ISSN-3139-6593 (Online)

Journal of Interdisciplinary Science &Technology

Agentic AI in Software Engineering: Opportunities, Risks, and a Human-Controlled Development Framework

Ekansh Tayade[1], Pranav Pagare[2], Darshan Aher[3], Smit Chaudhari[4]

[1,2,3] Computer Engineering, Sandip Institute of Technology and Research Center Nashik (SITRC), Maharashtra, India;

[4] Dharmsinh Desai University, Nadiad, Gujarat (DDU), India

ekanshshweta06@gmail.com, pranavnpagare@gmail.com, aherdarshan2803@gmail.com, smitchaudhari2601@gmail.com

***Corresponding author*- Ekansh Tayade**

DOI*: *10.5281/zenodo.22855949

Abstract: Agentic artificial intelligence is moving software engineering beyond prompt-based assistance toward systems that can plan tasks, invoke tools, modify repositories, execute tests, inspect results, and iterate with limited human intervention. This transition changes the engineering problem: the central question is no longer only whether an AI model can generate useful code, but whether an AI agent can act safely and reliably inside a software delivery system. This paper presents a structured review of agentic AI for software engineering and proposes a Human-Controlled Agentic Development Framework (HC-ADF). The review synthesizes foundational work on reasoning-and-acting systems, tool use, autonomous agents, multi-agent orchestration, coding agents, repository-level benchmarks, developer productivity, security, and AI governance. The proposed framework organizes agentic software development around six control stages: intent definition, planning, bounded execution, evidence-based verification, human authorization, and controlled release. It further introduces an autonomy ladder, risk-based approval gates, identity and tool permissions, sandboxing, traceability requirements, and a multidimensional evaluation model covering task success, code quality, security, change scope, verification effort, delivery stability, and human override behavior. The paper treats agentic AI as a software-delivery participant operating under explicit constraints rather than as an unrestricted autonomous programmer. Recent evidence from SWE-bench, SWE-Lancer, DORA, NIST, and OWASP shows substantial capability progress alongside persistent limitations in long-horizon reasoning, context handling, verification, security, and organizational control.

Keywords: Agentic AI, AI Agents, Software Engineering, Coding Agents, Autonomous Software Development, Human-in-the-Loop, LLM Agents, Software Security, AI Governance, SWE-bench, Software Delivery

I INTRODUCTION

Generative AI initially entered software engineering as an assistant: developers supplied a prompt or partial code and the model returned suggestions. Agentic AI extends this interaction model by coupling a foundation model with planning, memory, tool interfaces, execution environments, feedback loops, and policies. An agent can therefore move from answering a question to pursuing a software-engineering objective through multiple actions. Surveys of large-language-model agents describe this broader architecture as a combination of perception, reasoning, planning, memory, tool use, and action [1]. ReAct established an influential reasoning-and-acting pattern in which a language model alternates between task decomposition and environment interaction [2], while Toolformer demonstrated learned tool invocation [3]. Multi-agent frameworks such as AutoGen show how specialized agents, tools, and humans can be composed into collaborative workflows [4].

Software engineering is an unusually consequential environment for agentic systems because actions modify executable artifacts, dependencies, infrastructure, tests, and production systems. A coding agent may inspect a repository, create files, edit code, run commands, call APIs, execute tests, interpret failures, and produce a pull request. SWE-bench made this transition measurable by evaluating systems on real GitHub issues that require repository-level changes rather than isolated programming exercises [5]. SWE-bench Verified and SWE-Lancer extend evaluation toward more realistic engineering tasks and economic or managerial dimensions [6,7].

At the same time, increased autonomy introduces a control problem. A model can generate syntactically valid code that is semantically wrong, widen requested scope, misuse a tool, expose secrets, follow malicious repository instructions, or create changes that pass limited tests while violating architectural or product constraints. NIST notes that AI agents can perceive environments and take actions through tool-enabled scaffolding, creating security and reliability considerations beyond text-only systems [8]. OWASP's agentic-security work emphasizes risks such as goal hijacking, tool misuse, identity and privilege abuse, supply-chain vulnerabilities, unexpected code execution, memory poisoning, and cascading failures [9].

This paper asks: **How can software teams obtain the benefits of agentic AI while retaining meaningful human control over consequential engineering actions?** The contribution is a structured synthesis of the agentic software-engineering landscape and a Human-Controlled Agentic Development Framework (HC-ADF) connecting agent capability with explicit autonomy boundaries, evidence gates, and organizational governance.

II BACKGROUND AND CONCEPTUAL FOUNDATIONS

A. From LLM Assistance to Agency

The distinction between an assistant and an agent is primarily operational. An assistant usually returns an artifact or recommendation for a human to apply; an agent can select intermediate actions, interact with tools, observe outcomes, revise its plan, and continue toward a goal. ReAct [2] established a widely used reasoning-plus-action pattern. Toolformer [3] demonstrated tool selection and invocation. AgentBench framed agent evaluation as performance in interactive environments rather than only static text generation [10]. The 2024 survey by Wang et al. synthesizes agent architectures, applications, evaluation strategies, and open challenges [1].

B. Agentic Software Engineering

Agentic software engineering combines language-model reasoning with repository navigation, code editing, command execution, testing, documentation, version control, issue tracking, and deployment interfaces. CodeAct demonstrates an action interface in which agents execute code to interact with environments and revise actions based on observations [11]. OpenHands illustrates the practical direction in which agents can modify code, run commands, browse information, and interact with development environments [12]. SWE-agent investigates the importance of an agent-computer interface for repository-level engineering [13].

C. Why Software Engineering is a Suitable Agentic Testbed

Software engineering contains explicit artifacts, version histories, tests, build systems, issue descriptions, code review, and measurable delivery outcomes. These properties make it possible to evaluate agents with execution-based evidence rather than subjective text judgments. However, software correctness remains multi-dimensional: a patch can pass tests yet violate security assumptions, architecture, licensing requirements, performance constraints, or product intent. Benchmark success should therefore be interpreted as one signal within a broader assurance model.

III RELATED WORK

A. LLMs for software Engineering

Hou et al. conducted a large systematic literature review covering hundreds of studies on LLMs in software engineering, including code generation, testing, repair, documentation, and requirements work [14]. AlphaCode explored large-scale code generation through extensive sampling and filtering [15]. Empirical Copilot research found useful code-generation capability but also buggy, non-reproducible, or context-sensitive outputs [16]. These findings motivate a shift from generation quality alone toward end-to-end task reliability.

B. Repository Level coding Agents

SWE-bench contains 2,294 real-world software-engineering problems derived from GitHub issues and pull requests across 12 Python repositories [5]. Its design requires systems to understand repository context, modify multiple files, and satisfy execution-based tests. SWE-bench Verified was introduced to improve evaluation validity [6]. SWE-Lancer expands the task space to more than 1,400 real-world freelance software-engineering tasks with a combined nominal value of $1 million, including implementation and managerial decision tasks [7].

C. Organizational Evidence

DORA's 2025 research surveyed nearly 5,000 technology professionals and reported 90% AI use at work, more than 80% perceiving productivity improvements, and 59% reporting a positive influence on code quality [17]. The research frames AI as an amplifier of organizational strengths and weaknesses. DORA's 2026 analysis further describes a tension in which faster initial code generation can shift effort toward auditing and verification [18]. This supports the premise that agentic capability must be evaluated at the system level, not only by counting generated code or agent actions.

IV METHODOLOGY

This paper uses a structured narrative review and framework synthesis. Literature was organized into six evidence groups: (i) agent architectures and tool use, (ii) software-engineering agents and benchmarks, (iii) developer productivity and collaboration, (iv) security and failure modes, (v) governance and risk management, and (vi) organizational adoption. Priority was given to peer-reviewed papers, benchmark papers, standards, and primary research or institutional reports. The synthesis then mapped recurring findings into a control-oriented development model.

The framework follows three principles. First, **capability is separated from authority**: an agent may be technically capable of an action without being authorized to perform it. Second, **evidence must accompany consequential actions**: tests, static analysis, policy checks, and human review provide independent signals. Third, **autonomy should be risk-adaptive**: low-risk reversible actions can be more automated, while high-impact actions require stronger authorization.

V AGENTIC AI ACROSS THE SOFTWARE DEVELOPMENT LIFECYCLE

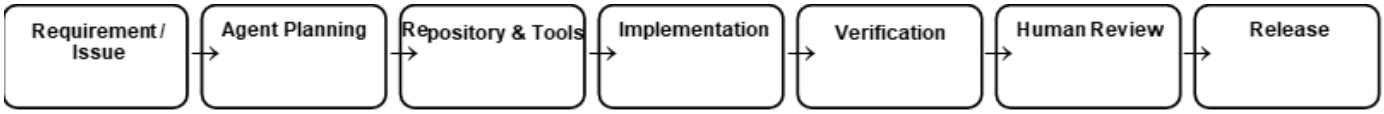


Fig. 1. Conceptual agentic software-engineering loop. Human authorization remains an explicit control point before consequential release actions

A. Requirements and Issue Analysis

Agents can summarize issue reports, identify acceptance criteria, inspect related tickets, retrieve repository conventions, and convert broad requests into implementation plans. The opportunity is reduced analysis friction. The risk is that an agent silently resolves ambiguity by inventing assumptions. A human-controlled workflow should therefore require assumptions, unresolved questions, affected components, and proposed acceptance criteria to be visible before implementation.

B. Planning and Repository Navigation

Repository-level work requires locating relevant modules, understanding dependencies, identifying tests, and estimating the change surface. SWE-bench demonstrates why this is harder than isolated code generation [5]. An agent should produce a bounded plan identifying intended files, expected behavior changes, tests, and constraints such as API compatibility. Plans should be versioned so reviewers can compare intended and actual scope.

C. Implementation and Tool Use

Agentic systems can use shell commands, code editors, package managers, test runners, issue trackers, search tools, and version-control operations. Tool use expands capability and the attack surface. Tool permissions should follow least privilege, with separate read, write, execute, network, and deployment capabilities. Sensitive credentials should not be directly exposed to model context when a scoped service identity can be used.

D. Testing and Self Repair

Agents can edit code, run tests, inspect failures, revise, and repeat. CodeAct illustrates this iterative pattern [11]. However, self-repair is not equivalent to independent verification. If the same agent writes code and designs its own tests, correlated blind spots can remain. Independent deterministic tests, static analysis, security scanning, and human review should remain outside the agent's authority boundary for higher-risk changes.

E. Pull Requests, Review and Release

Agents can generate pull requests, summaries, test reports, and change explanations. Review artifacts should make agent involvement visible and traceable: objective, tools used, files changed, commands executed, tests passed or failed, unresolved warnings, and approvals. Deployment authority should be separated from code-generation authority so that a coding agent cannot automatically promote its own output to production without satisfying policy gates.

VI OPPORTUNITIES

A. End to End Task Acceleration

Agentic systems can reduce coordination cost between code search, editing, testing, and documentation. DORA's 2025 findings indicate widespread perceived productivity benefits from AI-assisted development [17]. Productivity should therefore be measured at the task and delivery-system level rather than by lines of code or number of completions.

B. Software Maintenance and Technical- Debt Reduction

Agents are suitable for repetitive maintenance such as dependency updates, API migrations, test generation for

legacy modules, documentation, and stale-code identification. Because maintenance is repository-specific, agents can combine search, execution, and contextual reasoning. Automated cleanup should still be bounded by change-size limits and rollback mechanisms.

C. Test Generation and Defect Localization

Agents can generate tests, execute them, inspect stack traces, and propose repairs. This may increase iteration speed and coverage. The remaining challenge is test adequacy: passing generated tests does not prove all product or security requirements. Metamorphic testing, property-based testing, mutation testing, independent test suites, and human review can add assurance.

D. Knowledge Retrieval and Onboarding

Agents can retrieve repository documentation, architectural decisions, coding conventions, issue history, and examples while a developer works. This can reduce navigation cost. The principal requirement is provenance: retrieved context should be attributable to authoritative internal sources, with stale or conflicting documentation flagged rather than silently merged.

VII RISKS AND FAILURE MODELS

A. The Verification Bottle Neck

DORA's 2026 analysis reports that time saved in initial code generation can be reallocated to auditing and verification [18]. For agentic systems, the verification problem becomes more important because agents can produce changes faster than humans can inspect them. The objective should therefore be trustworthy throughput under explicit assurance constraints.

B. Security and the Expanded Attack Surface

Agentic systems introduce a larger attack surface because they can access tools, repositories, credentials, networks, and execution environments. NIST's 2025 work emphasizes understanding tool capabilities and limitations [8]. OWASP's guidance identifies threats that arise when models plan and act through tools and multi-step workflows [9]. Agent permissions should be explicit, scoped, auditable, revocable, and separated by environment.

C. Context and Architectural Drift

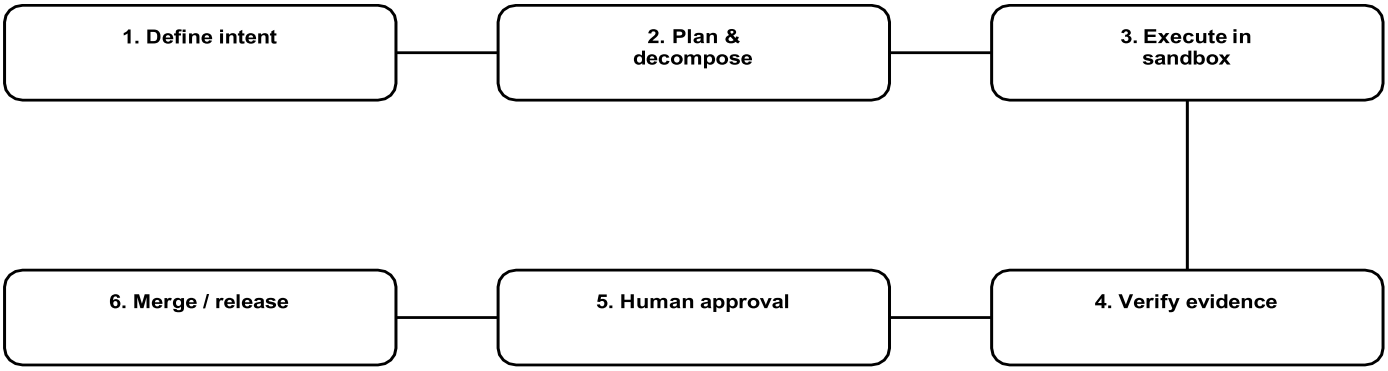
An agent may optimize for local task success while violating global architecture. It may duplicate functionality, introduce unapproved dependencies, or alter an interface without recognizing downstream consumers.

Risk	Agentic mechanism	Potential consequence	Primary control
Goal / prompt hijacking	Untrusted issue, README, web or tool content influences objectives	Unauthorized actions or data access	Context isolation, instruction hierarchy, approval gates
Tool misuse	Valid tool invoked unsafely	Data loss, destructive changes, policy violations	Least privilege, allowlists, dry-run mode
Identity & privilege abuse	Agent inherits broad credentials	Unauthorized repository/cloud actions	Scoped identities, short-lived credentials
Context / memory poisoning	Persistent memory stores malicious or stale information	Repeated incorrect behavior	Provenance, validation, bounded memory
Scope creep	Agent changes unrelated files or disables tests	Regressions and review difficulty	Specification boundaries, diff limits
Hallucinated implementation	Plausible but incorrect code/API use	Functional defects	Independent tests, static analysis, review
Cascading failure	Multiple agents propagate one wrong assumption	System-wide incorrect changes	Isolation, checkpoints, escalation
Automation over-trust	Human assumes agent output is correct	Reduced scrutiny, latent defects	Evidence-first review, calibrated trust

Table 1. Agentic AI risk-control matrix synthesized from agent, software-engineering, and security literature [8,9,17,18].

Architecture-aware retrieval, repository policies, dependency rules, and mandatory human review of cross-cutting changes can reduce this risk.

VIII HUMAN CONTROLLED AGENTIC DEVELOPMENT FRAMEWORK (HC-ADF)



High-impact actions pause for evidence and explicit human authorization.

Fig. 2. HC-ADF operating loop. Each stage creates evidence that constrains the next stage; high-impact actions require explicit human authorization.

A. Intent Definition

Every agent task begins with an objective, scope, acceptance criteria, and prohibited actions. The intent specification is the contract against which later behavior is evaluated. Production tasks should also identify environment, data classification, rollback method, and maximum permitted change surface.

B. Planning and Decomposition

The agent generates a plan before material changes. The plan identifies affected components, expected files, dependencies, tests, and risks. For medium- and high-risk work, a human approves the plan before execution.

C. Bound Execution

Execution occurs in an isolated workspace with explicit permissions. The agent receives only the tools required. Commands are logged, network access restricted, secrets brokered through scoped identities, and destructive operations disabled unless separately authorized.

D. Evidence Based Verification

Verification combines deterministic and independent evidence: unit/integration tests, static analysis, dependency and secret scanning, policy checks, type checking, performance checks where applicable, and review against original intent. For security-sensitive changes, security review should be independent of the code-generating agent.

E. Human Authorization

Humans retain authority over irreversible or high-impact actions.

Approval should be evidence-based rather than a confirmation dialog. Reviewers should see the objective, plan, final diff, tool activity summary, verification results, unresolved issues, and risk classification.

F. Controlled Release and Learning

Release is separated from code generation. Approved changes move through conventional CI/CD controls, staged deployment, monitoring, and rollback. Post-release observations feed organizational learning: recurring agent failures should lead to better policies, tests, retrieval sources, tool permissions, or task templates.

IX AUTONOMY LADDER

Level	Description	Typical actions	Human control
L0 — Advisory	Agent recommends; no direct action	Explain code, propose patch, summarize issue	Review before every action
L1 — Assisted	Reversible workspace edits	Draft code/tests/docs	Human initiates and reviews
L2 — Bounded agent	Multi-step task in sandbox	Implement issue, run tests, prepare PR	Plan approval + PR review

L3 — Conditional autonomy	Policy-limited merge/deploy	Low-risk maintenance, routine updates	Policy gates + evidence + escalation

Table 2. Proposed autonomy ladder. These are governance categories proposed by this paper, not a universal industry standard.

X MEASUREMENT AND EVALUATION MODEL

A mature evaluation model should avoid a single metric such as lines of code, task completion time, or benchmark pass rate. HC-ADF proposes seven dimensions: task success, code quality, security, scope adherence, verification burden, delivery stability, and human intervention.

XI Security Architecture for Agentic Coding Systems

A secure agentic software-engineering architecture should combine model safeguards with conventional software security. The model is not a security boundary by itself. Stronger boundaries come from identity, permissions, sandboxing, network policy, artifact controls, CI gates, and human authorization.

A. Identity and Last Privilege

Each agent should operate under an identity narrower than a human administrator. Repository read/write access, deployment authority, cloud credentials, and production-data access should be separately granted.

Dimension	Example indicators	Interpretation
Task success	Issue resolution; acceptance-test pass rate	Did the agent achieve the intended objective?
Code quality	Defects; maintainability; static-analysis findings	Is the resulting software sustainable?
Security	SAST findings; secret exposure; dependency risk	Did agent actions introduce weaknesses?
Scope adherence	Unrequested files/lines; policy violations	Did the agent stay within boundaries?
Verification burden	Review time; failed checks; rework	How much human effort is required to trust output?
Delivery stability	Rollback; change-failure rate; incidents	Did local acceleration create downstream instability?
Human control	Override; escalation; approval latency	Does the workflow preserve meaningful authority?

Table 3. Multidimensional measurement model for agentic software engineering

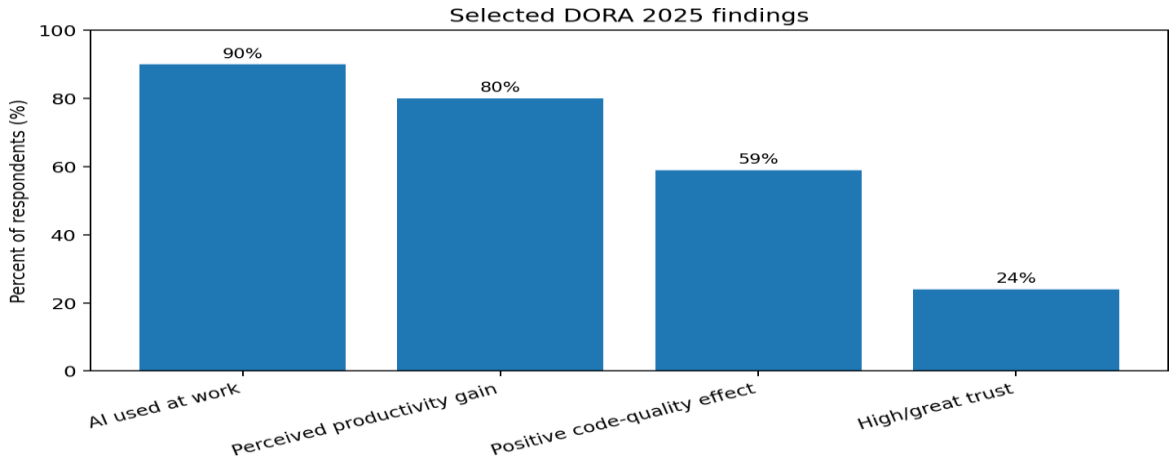


Fig. 3. Selected DORA 2025 survey findings. These values describe reported adoption/perceptions among surveyed technology professionals; they are not causal estimates [17].

Short-lived credentials and environment-specific identities reduce the impact of compromised or misdirected agents

B. Tools and Network Controls

Tools should be categorized as read-only, reversible write, executable, external-network, or deployment actions. High-risk tools should require approval or policy evaluation. Network egress should be restricted by default for coding tasks, especially when repositories contain secrets or proprietary data.

C. Supply- Chain and Dependency Controls

Agents can install packages, alter dependency files, modify CI configuration, or interact with model/tool ecosystems. Dependency pinning, provenance verification, vulnerability scanning, code signing, and review of new components should remain part of the pipeline. OWASP specifically highlights supply-chain vulnerabilities as an agentic risk category [9].

D. Auditability and Forensics

An agentic system should produce an auditable operational trace covering task intent, model/tool actions, files touched, commands executed, test results, approvals, and release decisions. The goal is not to expose hidden model reasoning; it is to retain operational evidence sufficient to reconstruct what the system did and which actions were authorized.

X1I Human AI Collaboration Model

Human control does not require a person to inspect every token or intermediate action. A scalable model is evidence-centered supervision: humans define objectives, approve risk-sensitive plans, inspect material changes, and retain authority over irreversible actions, while agents handle high-volume reversible work and generate evidence.

Human responsibility	Agent responsibility	Shared artifact
Define intent and constraints	Interpret task and identify assumptions	Task specification
Approve risk level	Propose plan and change surface	Execution plan
Review consequential changes	Implement and test	Diff + test evidence
Authorize release	Prepare release artifacts	Release checklist
Monitor outcomes	Detect anomalies and summarize telemetry	Post-release report

Table 4. Division of responsibility in a human-controlled agentic development workflow

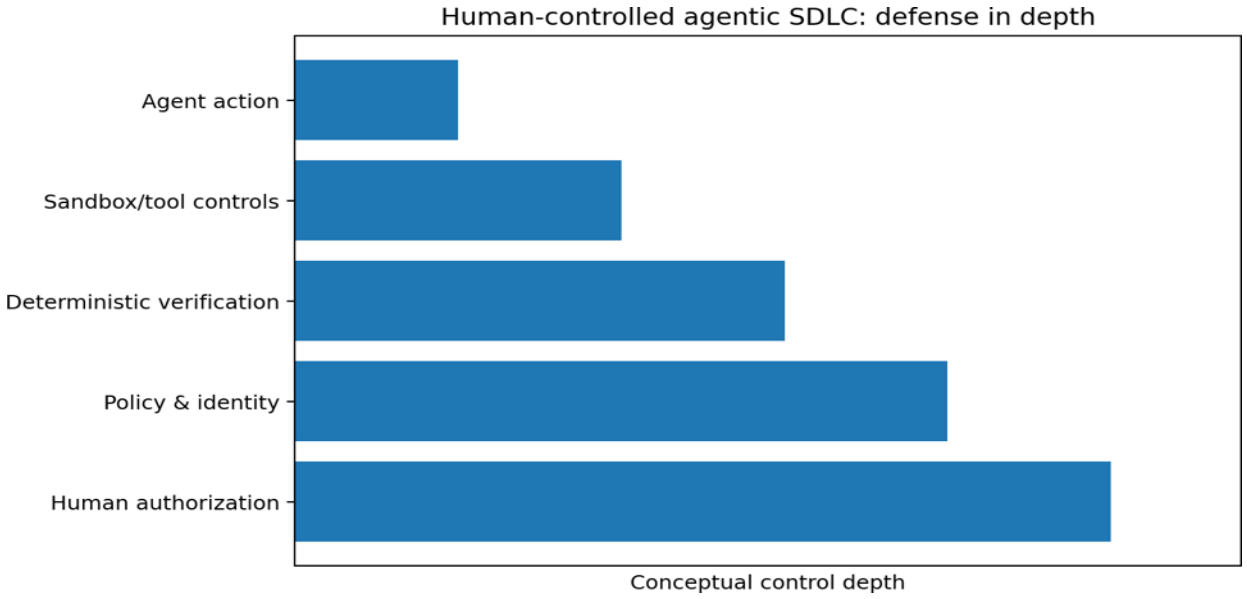


Fig. 4. Defense-in-depth control layers for agentic software development. Control depth is conceptual rather than a quantitative score

XIII Discussion

A. From Code Generation to Action Management

The central shift is from judging AI by generated text to judging an agent by the quality of its actions in an environment. A coding agent is a software actor with access to state, tools, and feedback. Evaluation must include action selection, scope control, recovery behavior, tool safety, and stopping behavior under uncertainty.

B. Capability Does Not Equal Authorization

A technically capable agent may be able to delete files, modify infrastructure, or deploy code. That does not mean those actions should be available by default. The distinction between capability and authority is central to HC-ADF and follows the least-privilege principle [39].

C. Verification as a first-class Engineering Activity

The evidence reviewed suggests that agentic adoption may shift engineering effort rather than eliminate it. DORA's 2026 analysis describes creation-to-verification tension [18]. Teams should invest in tests, observability, static analysis, architecture rules, and review tooling so faster production of changes remains compatible with reliable delivery.

D. Organizational Readiness

DORA's 2025 work emphasizes that AI amplifies the underlying organizational system [17]. Teams with weak ownership, poor documentation, fragile CI/CD, unclear architecture, or low-quality platforms may experience downstream coordination problems as agent output increases. Agentic adoption should therefore be accompanied by improvements in repository quality, developer platforms, observability, and engineering policies.

XIV Practical Adoption Roadmap

Phase	Focus	Entry criteria	Primary evidence
1. Observe	Advisory-mode agents	Policy and logging	Task productivity + failure log
2. Sandbox	Reversible repository actions	Isolation and scoped tools	Patch quality + tests
3. Govern	Risk-based autonomy	Approval gates + identity controls	Policy compliance + overrides
4. Integrate	CI/CD and issue-system integration	Reliable verification pipeline	Delivery + stability metrics
5. Scale	Multiple teams/agents	Monitoring, rollback, incident response	Organization-level outcomes

Table 5. Suggested staged adoption roadmap

XV RESEARCH GAPS AND FUTURE DIRECTIONS

First, agentic software-engineering evaluation needs broader real-world benchmarks that combine technical correctness with security, maintainability, architecture, and operational outcomes. SWE-bench and SWE-Lancer provide important foundations, but benchmark success remains narrower than production responsibility [5,7].

Second, agent-to-agent collaboration requires methods for detecting correlated errors. Multi-agent systems can distribute work across planner, coder, tester, reviewer, and deployment agents, but multiple agents may share the same flawed assumptions. Independent verification and diverse evidence sources are therefore important.

Third, agent memory and context governance remain open problems. Persistent memory can improve continuity but can also preserve incorrect or malicious information. Provenance-aware memory, expiry policies, confidence metadata, and human correction mechanisms deserve deeper study.

Fourth, organizations need standardized metrics for verification burden. A system that completes tasks rapidly but requires extensive human correction may not improve end-to-end productivity. Future empirical work should measure contribution across task success, review time, rework, incidents, and delivery stability.

Finally, agentic software engineering raises questions about responsibility and accountability. As systems gain authority to modify production artifacts, organizations will need explicit ownership models describing who approves, who monitors, who can revoke access, and who investigates failures.

XVI CONCLUSION

Agentic AI represents a transition in software engineering from AI that suggests code to AI that can plan, act, test, and iterate inside development environments. The literature shows progress in tool use, autonomous agents, repository-level coding, and multi-agent orchestration, alongside persistent limitations in long-horizon reasoning, context management, correctness, verification, security, and trust [1, 5, 7, 14, 17, 18, 39, 40].

This paper proposed the Human-Controlled Agentic Development Framework (HC-ADF), treating human oversight as an architectural control rather than an after-the-fact review step. The framework combines explicit intent, bounded planning, least-privilege execution, deterministic and independent verification, human authorization, controlled release, and post-release learning. Its central principle is: agents may perform increasingly complex work, but authority must remain proportional to risk and supported by evidence.

The practical path toward agentic software engineering is therefore controlled autonomy: systems capable enough to reduce engineering friction, observable enough to audit, constrained enough to remain safe, and integrated enough to improve the software-delivery system rather than only the speed of code generation.

XVII REFERENCE

[1] Wang L, Ma C, Feng X, Zhang Z, Yang H, Zhang J, Chen Z, Tang J, Chen X, Lin Y, Zhao WX, Wei Z, Wen J. A survey on large language model based autonomous agents. Front Comput Sci. 2024; 18:186345. https://doi.org/10.1007/s11704-024-40231-1.

[2] Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K, et al. ReAct: Synergizing reasoning and acting in language models. arXiv [Preprint]. 2022. arXiv:2210.03629. https://doi.org/10.48550/arXiv.2210.03629.

[3] Schick T, Dwivedi-Yu J, Dessì R, Raileanu R, Lomeli M, Hambro E, Zettlemoyer L, Cancedda N, Scialom T. Toolformer: Language models can teach themselves to use tools. In: Advances in Neural Information Processing Systems 36 (NeurIPS 2023). 2023. p. 68539–68551. https://doi.org/10.52202/075280-2997.

[4] Wu Q, Bansal G, Zhang J, Wu Y, Li B, Zhu E, Jiang L, Zhang X, Zhang S, Liu J, Awadallah AH, White RW, Burger D, Wang C. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. arXiv [Preprint]. 2023. arXiv:2308.08155. https://doi.org/10.48550/arXiv.2308.08155.

[5] Wang H, Peng X, Cheng H, Huang Y, Gong M, Yang C, Liu Y, Lin J. ECom-Bench: Can LLM agent resolve real-world e-commerce customer support issues? In: Potdar S, Rojas-Barahona L, Montella S, editors. Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track; 2025 Nov; Suzhou, China. Association for Computational Linguistics; 2025. p. 276–284. https://doi.org/10.18653/v1/2025.emnlp-industry.19.

[6] Ifrah S. Introduction to Azure AI and OpenAI. In: Getting Started with Azure OpenAI. Berkeley (CA): Apress; 2024. p. 1–32. https://doi.org/10.1007/979-8-8688-0599-8_1.

[7] Sun Y, Zhao Y, Wang Y, Du Y, Ma Z, Wang J, Zhang M, Zhang K, Huang Z. SWE-Mutation: Can LLMs generate reliable test suites in software engineering? In: Liakata M, Moreira VP, Zhang J, Jurgens D, editors. Findings of the Association for Computational Linguistics: ACL 2026; 2026 Jul; San Diego, CA. Association for Computational Linguistics; 2026. p. 39651–39674. https://doi.org/10.18653/v1/2026.findings-acl.1976

[8] Kung DC, Bhambhani H, Nwokoro S, Okasha W, Kambalakatta R, Sankuratri P. Lessons learned from software engineering multi-agent systems. In: Proceedings 27th Annual International Computer Software and Applications Conference. COMPAC 2003. 2003. p. 50-55. https://doi.org/10.1109/CMPSAC.2003.1245321.

[9] OWASP GenAI Security Project. OWASP Top 10 agentic applications 2026: AI agent testing guide [Internet]. CISO Platform. 2026 [cited 2026 Sep 20]. Available from: https://www.cisoplatform.com/profiles/blogs/owasp-top-10-agentic-applications-2026-ai-agent-testing-guide.

[10] Samuel V, Zou HP, Zhou Y, Chaudhari S, Kalyan A, Rajpurohit T, Deshpande A, Narasimhan KR, Murahari V. PersonaGym: Evaluating persona agents and LLMs. In: Christodoulopoulos C, Chakraborty T, Rose C, Peng V, editors. Findings of the Association for Computational Linguistics: EMNLP 2025; 2025 Nov; Suzhou, China. Association for Computational Linguistics; 2025. p. 6999–7022. https://doi.org/10.18653/v1/2025.findings-emnlp.368.

[11] X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, and H. Ji, "Executable Code Actions Elicit Better LLM Agents," *Proceedings of ICML*, PMLR 235, 50208–50232, 2024.

[12] Bruzzone F, Cazzola W, Favalli L. Code less to code more: Streamlining Language Server Protocol and type system development for language families. J Syst Softw. 2025; 231:112554. https://doi.org/10.1016/j.jss.2025.112554.

[13] Yang J, Jimenez C, Wettig A, Lieret K, Yao S, Narasimhan K, Press O. SWE-agent: Agent-computer interfaces enable automated software engineering. In: Advances in Neural Information Processing Systems 37 (NeurIPS 2024); 2024 Dec 10–15; Vancouver, Canada. Neural Information Processing Systems Foundation; 2024. p. 50528–50652. https://doi.org/10.52202/079017-1601.

[14] Hou X, Zhao Y, Liu Y, Yang Z, Wang K, Li L, Luo X, Lo D, Grundy J, Wang H. Large language models for software engineering: A systematic literature review. ACM Trans Softw Eng Methodol. 2024;33(8):220. https://doi.org/10.1145/3695988

[15] Hou X, Zhao Y, Liu Y, Yang Z, Wang K, Li L, Luo X, Lo D, Grundy J, Wang H. Large language models for software engineering: A systematic literature review. ACM Trans Softw Eng Methodol. 2024;33(8):220. https://doi.org/10.1145/3695988

[16] Moradi Dakhel A, Majdinasab V, Nikanjam A, Khomh F, Desmarais MC, Jiang ZM. GitHub Copilot AI pair programmer: Asset or liability? J Syst Softw. 2023; 203:111734. https://doi.org/10.1016/j.jss.2023.111734

[17] Haque EA. Towards trustworthy AI-assisted software development. In: 2025 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC); 2025 Oct 19–24; Raleigh, NC. IEEE; 2025. p. 431–432. https://doi.org/10.1109/VL-HCC65237.2025.00062

[18] Neema, Aarti, and Rashid Khan. "Crafting Effective AI Adoption Strategies." 1-23. Accessed September 20, 2026. https://doi.org/10.1002/9781394234028.ch1.

[19] Aaron, Lynn, et al. "AI Capabilities." State University of New York Press, 2024, pp. 4–4. SUNY FACT2 Guide, Second Edition, http://www.jstor.org/stable/jj.20522984.10.

[20] National Institute of Standards and Technology. Artificial Intelligence Risk Management Framework (AI RMF 1.0). Gaithersburg (MD): U.S. Department of Commerce; 2023 Jan. NIST Special Publication (SP) 100-1. https://doi.org/10.6028/NIST.AI.100-1

[21] Ouyang L, Wu J, Jiang X, Almeida D, Wainwright C, Mishkin P, et al. Training language models to follow instructions with human feedback. In: Advances in Neural Information Processing Systems 35 (NeurIPS 2022); 2022 Nov 28–Dec 9; New Orleans, LA. Neural Information Processing Systems Foundation; 2022. p. 27730–27744. https://doi.org/10.52202/068431-2011

[22] Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, Agarwal S, Herbert-Voss A, Krueger G, Henighan T, Child R, Ramesh A, Ziegler D, Wu J, Winter C, Hesse C, Chen M, Sigler E, Litwin M, Gray S, Chess B, Clark J, Berner C, McCandlish S, Radford A, Sutskever I, Amodei D. Language models are few-shot learners. In: Larochelle H, Ranzato M, Hadsell R, Balcan MF, Lin H, editors. Advances in Neural Information Processing Systems 33 (NeurIPS 2020). Curran Associates, Inc.; 2020. p. 1877–1901. Available from: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[23] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, et al. Attention is all you need. In: Guyon I, Von Luxburg U, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R, editors. Advances in Neural Information Processing Systems 30 (NIPS 2017). Curran Associates, Inc.; 2017. p. 5998–6008. Available from: https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[24] Wei J, Wang X, Schuurmans D, Bosma M, Ichter B, Xia F, Chi E, Le QV, Zhou D. Chain-of-thought prompting elicits reasoning in large language models. In: Advances in Neural Information Processing Systems 35 (NeurIPS 2022); 2022 Nov 28–Dec 9; New Orleans, LA. Neural Information Processing Systems Foundation; 2022. p. 24824–24837. https://doi.org/10.52202/068431-1800

[25] Barke S, James MB, Polikarpova N. Grounded Copilot: How programmers interact with code-generating models. Proc ACM Program Lang. 2023;7(OOPSLA1):78. https://doi.org/10.1145/3586030

[26] Xu FF, Alon U, Neubig G, Hellendoorn VJ. A systematic evaluation of large language models of code. In: Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming (MAPS 2022); 2022 Jun 13; San Diego, CA. New York (NY): Association for Computing Machinery; 2022. p. 1–10. https://doi.org/10.1145/3520312.3534862

[27] Sharma A. Analyzing redundancy in code-trained language models [master's thesis]. Ames (IA): Iowa State University; 2024. https://doi.org/10.31274/td-20250502-3

[28] Allamanis M, Barr ET, Devanbu P, Sutton C. A survey of machine learning for big code and naturalness. ACM Comput Surv. 2018;51(4):Art. 81, 37 p. https://doi.org/10.1145/3212695

[29] Ebert C, Louridas P. Generative AI for software practitioners. IEEE Softw. 2023;40(4):30–38. https://doi.org/10.1109/MS.2023.3265877

[30] Varma DK, Parmar R, Bhat S, Dutta P. MLOps for enterprise AI: A systematic literature review of frameworks, interpretability practices, and organisational readiness. Future Gener Comput Syst. 2027;187:108807. https://doi.org/10.1016/j.future.2026.108807

[31] Rajbhar SH. Impact of generative AI on software development productivity. Myresearchgo. 2026 Jun;2(6):54–62. https://doi.org/10.64448/myresearchgo.vol.2.issue.6.07

[32] Song F, Agarwal A, Wen W. The impact of generative AI on collaborative open-source software development: Evidence from GitHub Copilot. Inf Syst Res. 2026 Aug 24. Epub ahead of print. https://doi.org/10.1287/isre.2024.1361

[33] Leander KR. The future of developer experience. In: Developer experience unleashed. Berkeley (CA): Apress; 2025. p. 277–336. https://doi.org/10.1007/979-8-8688-0242-3_11

[34] Wienholt N. Prompt engineering with AI coding agents. In: GitHub Copilot and AI coding tools in practice. Berkeley (CA): Apress; 2025. p. 49–93. https://doi.org/10.1007/979-8-8688-1784-7_4

[35] Hosseini S, Seilani H. The role of agentic AI in shaping a smart future: A systematic review. Array. 2025;26:100399. https://doi.org/10.1016/j.array.2025.100399

[36] Somwanshi SK, Tharayil AS, Budhewar AS, Velu G, Thamizanban D, Gayathri KC, Dasarwar AN, Patil PG, Somwanshi SK, Patil BK, Perumal D, Karwande VS. Beyond performance benchmarks: A safety-critical evaluation paradigm for reliability and trust in clinical language models. In: Ortiz-Rodriguez F, Kansal M, Kumar Dhanaraj R, Khan F, editors. Medical LLMs for clinical safety assessment. Hershey (PA): IGI Global Scientific Publishing; 2026. p. 173–200. https://doi.org/10.4018/979-8-3373-7862-6.ch007

[37] Pressman RS, Maxim BR. Software engineering: a practitioner's approach. 8th ed. New York (NY): McGraw-Hill Education; 2015. 914 p. ISBN: 978-1-259-25315-7.

[38] Schwaber K, Sutherland J. The Scrum guide. In: Software in 30 days: how agile managers beat the odds, delight their customers, and leave competitors in the dust. Hoboken (NJ): John Wiley & Sons; 2012. Appendix 2; p. 133–152. https://doi.org/10.1002/9781119203278.app2

[39] Tharayil AS, Budhewar AS, Andy A, Velu G, Surendra AV, Sunkari V, et al. Explainable and multi-source AI frameworks for adaptive biosurveillance in public health and environmental systems. In: Ogwu M, editor. AI-enhanced biosurveillance: frameworks, practices, and futures [Internet]. Hershey (PA): IGI Global Scientific Publishing; 2027 [cited 2026 Sep 20]. p. 139–168. Available from: https://doi.org/10.4018/979-8-3373-9033-8.ch006

[40] Li H, Zhang H, Zhang JM, Lou Y, Ray B, Zimmermann T, Hassan AE. Agentic software engineering (SE 3.0): The rise of AI teammates. In: Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2 (KDD '26); 2026; Republic of Korea. New York (NY): Association for Computing Machinery; 2026. p. 13417–13418. https://doi.org/10.1016/3770855.3818242

