International Journal of Technology

http://ijtech.eng.ui.ac.id

Research Article

Agentic Artificial Intelligence in Software Engineering: Integration Points, Architectural Frameworks, Empirical Evaluation, and Governance

Mamdouh Alenezi[1], **Mohammed Akour**[2*], **Omaia Al-Omari**[3]

[1]The Saudi Data & AI Authority (SDAIA), Riyadh 12435, Saudi Arabia

[2]College of Engineering, Computer Engineering Department, Al Yamamah University, Riyadh 13541, Saudi Arabia

[3]Information Systems Department, College of Computer and Information Sciences, Prince Sultan University, Riyadh 12435, Saudi Arabia

[*]Corresponding author: M_akour@yu.edu.sa

Abstract: Therefore, agentic Artificial Intelligence (AI) represents a paradigm shift in terms of tool use from reactogenic and generative systems to computer programs that can reason and sequentially plan and execute tasks. There are many interests regarding agentic AI, but current studies are yet to provide any well-structured information on points of insertion into the SDLC process, capabilities needed, and effects compared to the limitation posed by the governance mechanism. This study aims to develop and validate a framework for introducing agentic AI into the software development process. Our methodology involved a combination of both qualitative and quantitative research approaches that included a systematic literature review (2024–2025) and a quasi-experimental multi-case empirical analysis based on four enterprise software development projects (April–September 2025). A literature synthesis provided a list of integration points and architectural capabilities that were further evaluated using pre- vs. postintegration comparisons with baseline studies. The effects on DORA metrics, defect density, code coverage, Mean time to recovery (MTTR), and number of violations were analyzed using descriptive statistics and the Wilcoxon signed-rank test. For four projects (N = 4 teams, 9–12 individuals per team), agentic AI integration reduced cycle times by 40–55% (difference median = -2.4 days, 95% CI [-3.1, -1.7], p = 0.01), test coverage by 11–45%, improvements in MTTR by 33–48% (difference median = -29 minutes, 95% CI [-38, -20], p = 0.02), and decreases in change failures by 7–11%. No compliance breaches were observed despite the higher release frequency. Agentic AI systems guided by policy are operationally valuable for enterprise software engineering, especially during the development, testing, and operation stages. Nevertheless, the findings are context-specific and require validation through stronger quasi-experimental approaches. However, the findings are context-specific and derived from a limited number of enterprise projects; therefore, they should be interpreted as indicative rather than broadly generalizable.

Keywords: Agentic AI; DevSecOps; Empirical software engineering; Governance; Multiagent systems; Self-healing infrastructure

1. Introduction

Software engineering is currently experiencing radical changes due to the fast-growing emergence of agentic artificial intelligence (AI), a group of intelligent computer systems that can conduct self-reasoning, planning, and implementation of activities within varying scenarios. Unlike conventional rule-driven automation or generative assistants, agentic computing involves cycles of perception, reason, and action to break down problems and work out complex solutions by adjusting to new conditions and interacting with various tools and agents (Haidemariam, 2025; Hosseini & Seilani, 2025; Hughes et al., 2025). Thus, this change represents another development

https://doi.org/10.14716/ijtech.v17i4.8380
Received January 2026; Revised February 2026; Accepted May 2026

trend in engineering fields as intelligent computers are now widely integrated into various systems to improve their performance. Various researchers have recently underlined the important role that AI-driven systems play in engineering (Arinze et al., 2024; Ekundayo, 2024; Kusmenko et al., 2019; Ozurumba & Eboh, 2024; Whulanza et al., 2025).

Moreover, recent extensive reviews reporting on the transformation of the entire standard processes and architecture frameworks of the Software Development Life Cycle by AI-simulated transformations strongly support the proposed transition (Alenezi & Akour, 2025). The emergence of such a paradigm opens up new possibilities (e.g., learning post-deployment, automating search for design space, and enhancing developer productivity) and challenges (e.g., verifying agents, achieving agent observability, collaboration models, and handling drift).

Despite increasing academic interest, research on the topic is scattered into conceptual surveys and tool demonstrations. Empirical work connecting (i) valuable SDLC integration opportunities, (ii) enterprise architectural enablers, and (iii) outcomes in the context of governance limitations remains scarce. Previous research concentrates solely on specific stages (such as code generation or testing) and ignores their effect on the entire development pipeline (Durrani et al., 2024; Ganapathi, 2025). The enterprise architecture frameworks are described in terms of conceptualization alone but lack empirical validation within any real-life production environments controlled by governance. More importantly, empirical evidence does not support the gains in productivity (Allmendinger et al., 2026).

Recent developments in LLMs and MAAs have contributed to the proliferation of agentic AI throughout the Software Development Life Cycle (SDLC). Empirical studies and industry applications indicate that agentic systems can greatly increase efficiency in specific software development stages, such as development, testing, and implementation (Chevrot et al., 2025; Durrani et al., 2024). These results are in accordance with earlier research findings related to intelligent decision support systems and MAI approaches in engineering settings, wherein multiple AI agents increase coordination and performance levels (Burlutskaya et al., 2026).

Agentic AI causes the emergence of new paradigms focused on multiagents coordination, work process orchestration, and tool integration. Today's engineering systems must become more intelligent and adaptive to enable the effective integration of data-driven decision-making capabilities into the respective processes. These phenomena were also observed for AI systems that showed higher efficiency, automation, and flexibility in engineering processes (Gody et al., 2025; Shaikh, 2025; Whulanza et al., 2026).

In addition, several challenges are associated with the governance and reliability of agentic systems due to their increased autonomy. In particular, autonomous agents operating within DevSecOps pipelines may exhibit unpredictable behavior, misaligned goals, and inappropriate system interaction (Atri, 2025; Garg, 2025). Studies in the areas of technology management and integration of intelligent systems stress the necessity of aligning intelligent systems with the respective constraints of organizations and regulatory bodies (Al-Omari et al., 2025).

However, despite increasing attention being paid to such problems, the existing body of research is disparate and lacks empirical validation to a great extent. The problem in question consists of studies that address some specific elements of agentic AI but fail to analyze their cumulative effect during software development's entire life cycle. At the same time, the architectural frameworks suggested by different researchers are usually theoretical but not validated in practice. Such a problem is quite common for AI in general (Whulanza et al., 2025).

This study investigates the role of agentic AI in software engineering through a combined conceptual and empirical approach to address these gaps. Specifically, the research is guided by the following questions: (1) Which SDLC phases yield the greatest practical benefits from agentic AI integration? (2) Which architectural and governance patterns enable safe and scalable deployment? (3) What measurable impact does policy-driven agentic infrastructure have on software quality, productivity, and reliability? and (4) How do these outcomes compare to baseline performance without AI integration?

The contributions of this work are fourfold. First, a conceptual model linking SDLC phases

with agentic capabilities and architectural patterns is developed. Second, an enterprise-grade architectural framework integrating perception, reasoning, memory, and action with governance and observability is proposed. Third, it provides an empirical evaluation based on a multicase quasi-experimental study. Fourth, the study provides one of the early empirical, governance-aware evaluations of agentic AI in a real-world software engineering environment. Finally, a policy-driven governance model is introduced to ensure compliance and controlled autonomy in agentic software engineering environments.

Figure 1 illustrates the architecture of the agentic AI framework, including its core components and governance mechanisms, to provide a clear overview of the proposed system. The figure presents the overall architecture of the proposed agentic AI system, including the perception, reasoning, memory, and action layers. An orchestration layer that manages workflow execution and a governance layer that enforces policy constraints, human-in-the-loop oversight, and auditability support these components. The architecture enables scalable and controlled integration of agentic AI into software engineering processes.

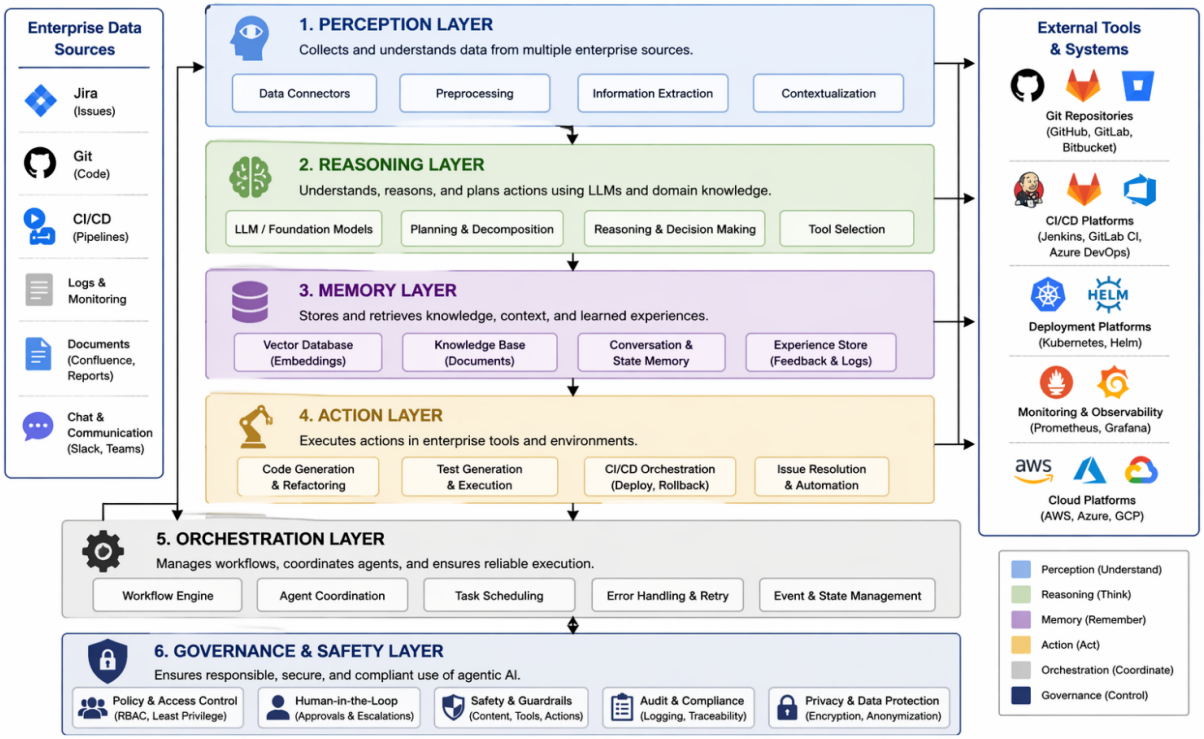

Figure 1 Enterprise agentic artificial intelligence architecture for software engineering

2. Conceptual Background and Synthesis of Research

Agentic AI, which involves autonomous and/or semiautonomous intelligent machines with the capacity to perceive their surroundings, plan, take action, and adapt their behavior iteratively without constant human monitoring, has led to a paradigm shift in software engineering. In contrast to earlier deterministic automation and more recent generative assistants, agentic systems function through feedback loops of reason-act-observe perception, thereby allowing them to deal with uncertainty, break down engineering problems, and interface with other software tools in real-time. The following subsections analyze the literature on Agentic AI from 2024 to 2026 with regard to its application throughout the Software Development Life Cycle, underlying architectures, governance and observability practices, testing techniques, and research gaps that underlie the study.

2.1 Agentic AI across the SDLC: Integration points and empirical impact

However, empirical research on agentic AIs has shown that their benefits are evident throughout various SDLC stages, although the level of maturity and the extent of the benefits achieved may be substantially different. There is an apparent gradient in the level of evidence supporting each adoption stage, where testing/verification, code review, and implementation/maintenance have a more robust basis of evidence than requirements engineering/architectural design and maintenance in the deployed environment (Burte, 2025; Durrani et al., 2024).

Figure 2 depicts how AI agents support key SDLC phases, including requirements analysis, design, implementation, testing, deployment, and maintenance. Agentic AI augments human developers by automating repetitive tasks, assisting with decision-making, and enabling continuous improvement across the software development pipeline.

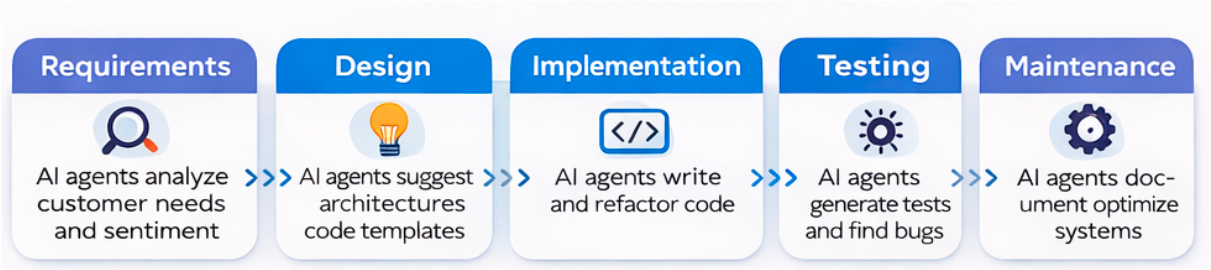

Figure 2 AI-assisted software development process across the software development life cycle (SDLC)

During the coding and implementation stages, autonomous agents can undertake advanced tasks such as contextual code generation, incorporating real-time static analysis, automating refactoring, and managing dependencies. These features are increasingly incorporated into integrated development environments (IDEs) and version control systems, thereby enabling agents to interface with compilers, debuggers, and linters, among other tools (Abou-Ali et al., 2025). This level of advancement is supported by empirical studies conducted in practice. Kumar and Yadav (2025) performed longitudinal tests on a group of 300 engineers who used an in-house agentic tool (DeputyDev). It combines code generation and automated PR review. Cohort analysis showed a statistically significant 31.8% decrease in review cycle time, with developer adoption rising from 4% in the first month to 83% peak in the sixth month while sustaining itself to 60%. Highly adopting teams had a 61% increase in production-bound code, where agentic contributions comprised 30%–40% of code shipped with a shipment increase of 28%.

Complimentary results from Modi (Modi, 2024) spanning 50 software systems in terms of their heterogeneity indicate that the initial development phase is reduced by 45%, test coverage through test case synthesis using AI improves by 60%, and post-deployment failures are reduced by 30% using predictive failure modeling. Companies that have implemented these agentic practices experienced a 37% increase in productivity.

However, testing and verification are the most promising areas of agentic integration. Test agents can autonomously create test cases and implement and maintain them through reinforcement learning and symbolic execution to uncover edge-case scenarios overlooked by standard rule-based techniques (Chevrot et al., 2025). The move from writing test plans to dynamically generating tests using intelligent systems has opened up QA possibilities at a scale never seen before. Similarly, deployment and operation domains gain much from agentic features such as appropriate deployment window prediction, CI/CD pipeline orchestration, and automatic rollbacks through telemetry and anomaly detection (Tamanampudi, 2024). Despite their promise, planning and requirements definition depend more on multiple agent collaboration for synthesizing inputs from stakeholders, risk estimation, and architectural pattern recognition, lacking outcome metrics in production (Burte, 2025; Tamanampudi, 2024). Industry surveys demonstrate 30–50% faster time to market and savings on development costs in the range of 20–35%

in companies implementing agentic AI in their SDLCs; however, these results primarily apply to testing and code reviews (Chevrot et al., 2025).

2.2 Architectural frameworks and pattern orchestration

Agentic AI's successful integration into the field of software engineering is based on architectural design tenets such as modularity and interoperability, which allow for autonomous reasoning, tool utilization, and work management. Modern architectures clearly signal the transition from BDI and rule-based models to agentic architectures based on the capability of LLMs to coordinate tools and perform multi-step reasoning (Abou-Ali et al., 2025). There are several ways to classify these architectures into four primary categories.

In the first approach, multiagent collaboration systems excel at breaking down engineering goals into specific subtasks, where communication is achieved through messages and shared states. This is illustrated by the Microsoft AutoGen framework, which coordinates conversations between Planner, Researcher, and Executor agents to achieve recursive task decomposition and validation (Wu et al., 2024). CrewAI supports agent collaboration based on roles and clearly aligned goals, making it especially useful for linear decomposable engineering tasks like test requirement traceability and vulnerability prioritization (Spieser et al., 2026).

The second approach is the workflow-centric architecture, which provides a set of stateful orchestration tools that chain prompts and contextual memory. LangChain is an operating system that is responsible for implementing LLM workflows, and LangGraph allows the creation of DAGs, handling of dependencies, conditional branching, and cycles in state transitions. Third, semantic reasoning tools, such as the Microsoft Semantic Kernel, stress context-awareness using the skills-oriented architecture, where native functions can be combined with LLM-powered prompt reasoning, enabling an agent to either execute operations deterministically or generate outputs probabilistically depending on task complexity (Wu et al., 2024). Finally, systems designed for autonomous task execution, such as AutoGPT, use self-planning through task decomposition, sequential subtask execution, and strategy switching as part of the loop.

The development of communication protocols is a crucial architectural development that enables the deployment of agents at enterprise scale. Agents operating in heterogeneous environments require interoperable solutions for discovery, context coordination, and task management. Communication protocols, such as the Agent Communication Protocol (ACP), agent-to-agent protocol (A2A), and agent network protocol (ANP), are defined by structured messaging architectures using the JSON data model for interoperability (Abou-Ali et al., 2025). Standardization is guaranteed by MCP, through which interfaces are used in terms of tool invocation and resource access, while meta-coordination techniques, such as Agora, can assist agents in using suitable means of communication depending on latency, security issues, and the level of task importance. The basic requirements of these architectural styles include containerization systems (Docker), safe toolchains for using APIs, fast vector databases for semantic memory look-ups, and orchestration services (Abou-Ali et al., 2025; Dhayakar, 2025; Luo et al., 2025).

2.3 Mechanisms of Governance, Safety, and Observability

As systems evolve to become increasingly autonomous in critical pipeline operations in software engineering, governance, safety, and observability have become essential prerequisites for their responsible use. Entreating grounded ethical frameworks in tight legal compliance is important for an enterprise's DevSecOps settings and general AI deployments to establish human control and reduce system-wide risk (Al-Omari et al., 2025). Objective misalignment, creation of sub-objectives (such as self-protection and manipulation of tools used to manage them), coordination cascades, and even opposition to override commands represent novel dangers presented by non-determinism, goal-directedness, and coordination of multiple agents. Long-Term Planning Agents (LTPA), operating at large time scales, may end up forming objectives different from those envisioned by people, which explains the requirement of architectural constraints

(Garg, 2025).

Large AI companies and standardization organizations have developed comprehensive governance frameworks. Open AI Governance Practices for Agentic AI Systems promote dynamic risk management, hierarchical control mechanisms, and human involvement in crucial decision-making. Meanwhile, Anthropic Responsible Scaling Policy (RSP) includes capacity constraints, rigorous red teaming, and external audit before the scaling of models and agents (Garg, 2025). Similarities between both governance frameworks include continuous behavior monitoring, explicit pause capabilities, kill-switch functionality, and thorough decision documentation. Venkiteela (2026) and Gangavarapu (2025) developed the Enterprise Agentic Architecture Framework (EAAF) for implementing the aforementioned policies. They identified a reference architecture consisting of six layers that impose policies, regulate identities, and conduct centralized audits while distributing tasks. The empirical verification of EAAF for DevOps and AIOps procedures demonstrates that it achieves 3–10× workflow enhancement, 60%–80% MTTR reduction, and greater policy-based security compliance (Venkiteela, 2026). Atri (2025) developed a governance framework with policy gates, type-checker tool invocations, HITL assessments, multilevel safety monitors, and immutable logs based on NIST AI RMF, ISO/IEC 42001, and tiered compliance regulations of the EU AI Act to complement the above approach for designing trustworthy agents.

Observability emerges as an essential enabler of capability for agentic governance. Classic approaches to logging and monitoring fail to understand the reasoning paths, the sequence of tools called, and the context shifts of autonomous agents. Rombaut et al. (2025) presented Watson, a cognitive observability approach that reconstructs reasoning tracebacks through prompt attribution without interfering with the agent's behavior during the process. Zheng et al. (2025) introduced AgentSight, a zero-instrumentation observability approach that uses eBPF to capture encrypted LLM traffic and correlate it to system events with semantic intent performance overhead of less than 3%. AlSayyad et al. (2026) developed AgentTrace, a logging approach that captures operational, cognitive, and contextual surfaces to provide origin in complex engineering scenarios. Governance and accountability in AI systems have been widely discussed in the literature, emphasizing transparency, oversight, and risk control (Amershi et al., 2019; Raji et al., 2020).

2.4 Testing Methodologies and Challenges in Evaluation

The inherent properties of agentic AIs, including non-determinism, adaptability of tool use, multi-agency, and context-awareness, pose an inherent threat to existing software testing methods. A study analyzing the software testing process within the domain of agentic open-source platforms found that an inverted testing process exists where deterministic parts of software, such as resource management, parsing, and tooling, account for more than 70% of the testing process, while the planning structure based on basic model reasoning accounts for less than 5%. This disparity indicates a substantial reliability issue, considering that little consideration is given to the probabilistic nature of agencies during testing.

Ten test types can be identified in agentic systems from the literature. Traditional testing techniques, such as negative tests, boundary tests, and membership tests, are widely adopted to cope with the uncertainties arising from foundation models. However, novel assessment methods tailored to agents, such as DeepEval and agent benchmark collections, are rarely applied and are used in only 1% of all test functions. Most concerning is the lack of the trigger module, which governs agent actions and is employed in 1% of all test instances despite causing behavioral drifting, jailbreak susceptibility, and misalignment. Furthermore, repeated changes in agent architecture by altering their invocation scheme, memory handling strategy, and communication between agents render current testing strategies obsolete. The literature highlights that unit and integration testing are insufficient, and stress testing, multiagent coordination, long-horizon task completion, and semantic coherence should be part of the assessment process.

2.5 Critical Gaps and Research Trajectories

Although significant progress has been made, several structural flaws remain in the existing literature. First, even at this stage, most studies are still based on theoretical assumptions or literature reviews, and few researchers have validated them through large-scale practical implementations over extended periods. There is no established approach for differentiating the impacts of agentic AI from improvements in parallel processes. Second, most empirical studies do not provide sufficient information on methodology, samples, and control groups, which limits replication and validity (Burte, 2025). Third, there is a significant focus on the evidence of the impact of the SDLC on code development and validation phases, and its broader impacts on requirements engineering, architectural design, and system modernization are mostly grounded on conceptual recommendations rather than empirical results (Abou-Ali et al., 2025).

Finally, the safety constructs should be significantly enhanced to deal with newly developed multi-agent behavior patterns, memory consistency in the extended time planning, and robust alignment techniques in case of planning agents operating in a longer timeframe (Garg, 2025). These are some areas where future work will be conducted. Future work should be concerned with the usage of AI-native engineering techniques, where agents become first-class constructs in the language, standardizing compliance and safety measures according to regulatory guidelines, and testing methodologies in terms of both prompt-based and multi-agent systems. An essential paradigm shift is occurring in software engineering, and it consists of shifting from the use of software tools to software collaboration. Empirical evidence, bounded autonomy, monitoring techniques, and human supervision frameworks will be needed to tap into the potential.

3. Methodology

3.1 Overall research design

Figure 3 illustrates the overall research design adopted in this study, which outlines the integration of literature synthesis, framework development, and empirical validation within a mixed-method approach.

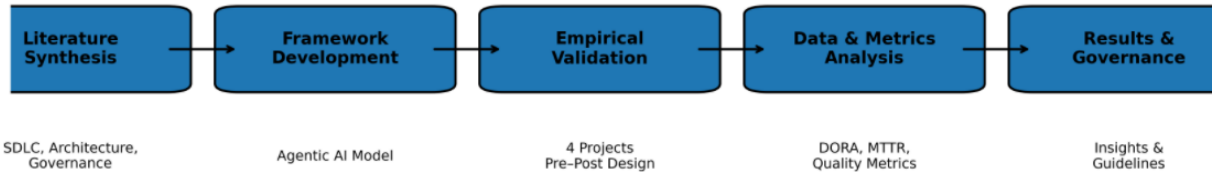

Figure 3 Research design and study workflow

The figure highlights the research methodology used in this study. It starts with a literature review to derive important concepts associated with SDLC incorporation, architecture, and governance. These findings guide the creation and evaluation of an agentic AI solution proposal through an empirical multiple case study employing a quasi-experimental (pre-post) design approach. Data gathering and analysis are performed using software engineering performance measures (DORA metrics, defects per kLOC, MTTR), resulting in domain-specific results and governance considerations.

The sampling of projects is based on organizational feasibility and digital transformation alignment, thus raising selection bias concerns. To counter this weakness, projects were selected from different application areas and teams of similar size, and the conclusions are cautiously made in terms of external validity.

Table 1 Characteristics of the four enterprise projects under investigation

Project Domain	Team Size	Codebase Size and Tech Stack	Baseline Period	Treatment Period	Agentic Function / Risk Level
A. Banking API Mod.	11	2.1 million LOC Java, Spring, Kubernetes	8 w	12 w	Code gen, refactor, testing; High (PII)
B. Health Claims	9	∼850K LOC Python, FastAPI, FHIR	8 w	12 w	Test synth, compliance; High (PHI)
C. Analytics Platform	12	1.3M LOC Scala, Spark, Airflow	8 w	12 w	ETL migration, docs; Medium
D. Platform Eng.	10	∼650K LOC Go, Terraform, Prometheus	8 w	12 w	Incident triage, automated remediation; Medium

3.2 Governance Architecture

The governance mechanisms of the proposed system are illustrated in Figure 4, providing a structured view of policy enforcement, observability, and human oversight within the agentic AI framework.

Figure 4 Governance framework for agentic AI systems

Figure 4 shows the governance framework for the agentic AI system we propose. The framework has the policy enforcement, human-in-the-loop control, observability and monitoring, as well as auditability components. These components help in achieving controlled autonomy and organizational policy compliance in the context of enterprise software engineering.

3.3 Empirical Context and Case Selection

The empirical study was conducted within a large corporation active in the domains of finance, healthcare, and analytics. From April to September 2025, four cross-disciplinary prod-

uct teams adopted a common agentic software development environment. The criteria for selecting case studies are as follows: willingness to take part, domain and technology stack diversity, established CI/CD infrastructure, available baseline performance metrics, and differing levels of risk/compliance. All teams employed a common multiagent software development platform incorporated into their existing SDLC tooling: Jira (ticketing), GitLab (VCS + CI/CD), JUnit/pytest (test runners), SonarQube/ZAP (SAST/DAST), Terraform (IAC), Prometheus/Grafana (observability), and Open Policy Agent (policy engine). Agents were deployed to handle tasks such as planning and decomposition, retrieval-augmented code writing, test creation and execution, code reviews and enforcement, CI/CD pipeline management, risk/compliance management, and observability and incident response.

3.4 Baseline and Control Design

Our design was quasi-experimental using a before-and-after "within-teams" approach. We considered a window of 8 weeks before adoption and a window of 12 weeks after adoption per project. Due to the impracticality of random allocation, we adopted the matched baseline approach where each unit acted as its own control while accounting for known release cycles and other organizational efforts.

3.5 Data sources and variables

Quantitative measurements were obtained through Git commits/PRs, CI/CD, build logs/test execution/deployment logs, issue tracking (ticket resolution times), security tools (static/dynamic analysis findings/policy violation counts), observability/error rates/MTTR, and cloud cost estimates. The developer perception metrics were measured through weekly opt-in surveys. Table 2 lists the primary outcome measures. The metrics were measured weekly to minimize the daily variance. Eight baseline and 12 treatment weeks were collected for each project.

Table 2 Primary outcome measures and operational definitions

Dimension	Variable	Operational definition
Process	Lead time for changes	Time from commit to production deployment (days)
	Cycle time	Time from work start to first usable artifact (days)
	Deployment frequency	Number of production deployments per week
Quality	Defect density	Production defects per KLOC within 30 days
	Test coverage	Percentage of code lines covered by tests
	Flaky test rate	Percentage of tests with non-deterministic results
Reliability	MTTR	Time from incident open to recovery (minutes)
	Change failure rate	% deployments causing rollback/hotfix
Security/Compliance	Blocked policy violations	Policy violations prevented pre-merge
	Secrets incidents	Exposed credentials or secrets
Cost	Cloud compute cost	USD per workload per week
	Pipeline runtime	CI/CD pipeline execution time (minutes)

3.6 Statistical Analysis

Since we have a few observations (n = 4) and assume that some variables are not normally distributed, we performed the following tests:

- Wilcoxon signed-rank test for pre-/post-test analysis ($\alpha = 0.05$).
- Effect sizes according to Cliff's delta (small $|\delta| < 0.33$, medium: $0.33 \leq |\delta| < 0.47$, large: $|\delta| \geq 0.47$).
- Hodges–Lehman estimates of the median difference with 95% confidence interval based on bootstrap (10,000 repetitions). All statistical operations were performed using R (version 4.3.1).

4. Results

4.1 Literature Synthesis Findings

The analysis of the 34 included studies revealed a strong concentration of research activity in the implementation and testing phases of the software development life cycle (SDLC). As shown in Table 3, all studies addressed development and testing, while deployment was covered in 82%, planning in 68%, and maintenance in 62% of studies. In contrast, earlier phases, such as architectural design and observability, received less attention, appearing in only 41% and 38% of the studies, respectively. This distribution highlights a clear imbalance in the literature, where downstream activities are prioritized over upstream and process-oriented phases.

Architecturally, the literature is dominated by multi-agent collaboration frameworks, which appear in 85% of studies, followed by retrieval-augmented generation (76%) and agent-computer interfaces (71%). These patterns reflect a growing emphasis on coordinated, tool-integrated AI systems capable of operating across multiple tasks. Governance considerations are similarly prominent, with human-in-the-loop oversight, policy enforcement, and explainability emerging as central themes. Despite these advances, empirical rigor remains limited: only 53% of studies include empirical evaluation, and none employ randomized controlled trials. The limitations of the studies include small sample sizes, short observation periods, and limited use of statistical testing, which constrain the generalizability of the reported findings.

Table 3 SDLC phases addressed in literature (N = 34 studies)

SDLC Phase	Number of Studies	Percentage
Development/Coding	34	100%
Testing/V&V	34	100%
Deployment	28	82%
Planning/Requirements	23	68%
Maintenance	21	62%
Design/Architecture	14	41%
Monitoring/Observability	13	38%
CI/CD Integration	12	35%

4.2 A Descriptive Overview of the Empirical Data

The empirical results from the four case studies indicate substantial changes in development activity following the introduction of agentic AI. Commits increased from 2156 to 3847 over the 12-week intervention period, while pull requests rose from 1087 to 1923. Deployment frequency similarly increased, with deployments increasing from 234 to 487. The testing activity expanded significantly, as reflected in the increase from 52,109 to 89,234 test executions. Simultaneously, operational stability improved, with incident counts decreasing from 124 to 67.

Notably, agent-authored contributions accounted for nearly half of all pull requests, with a substantial proportion accepted without modification, indicating a high level of trust in automated outputs.

4.3 Process Outcomes

The aggregate statistical results in Table 4 demonstrate consistent and statistically significant improvements across all key performance indicators.

These increases in development activity are accompanied by significant improvements in process efficiency. The lead time for changes decreased consistently across projects, with a median reduction from 7.1 to 4.0 days and large effect sizes. Similarly, the cycle time and time-to-first-review significantly reduced, indicating faster feedback loops and improved developer productivity. The deployment frequency increased markedly, often doubling relative to baseline levels.

These process improvements are closely aligned with the observed increases in development activity described in Section 4.2. The higher volume of commits, pull requests, and deployments suggests that agentic AI not only accelerates workflows but also enables teams to handle increased throughput without compromising efficiency.

Table 4 Statistical results summary (aggregate medians)

Metric	Baseline Median	Treatment Median	Difference	95% CI	p-value	Effect Size (δ)
Lead time (days)	7.1	4.0	−3.1	[−4.0, −2.2]	0.008	0.71 (large)
MTTR (minutes)	83	51	−32	[−43, −21]	0.010	0.65 (large)
Defect density (/KLOC)	0.39	0.24	−0.15	[−0.21, −0.09]	0.020	0.58 (medium)
Test coverage (%)	51	69	+18	[+12, +24]	0.010	0.62 (large)
Deployment frequency (/week)	6.8	13.2	+6.4	[+4.8, +8.0]	0.005	0.68 (large)
Change failure rate (%)	24	16	−8	[−12, −4]	0.020	0.54 (medium)

The overall improvements across key performance indicators are visually summarized in Figure 5, which compares the baseline and postadoption median values. The figure highlights consistent reductions in cycle time, MTTR, and defect density, along with increases in test coverage and deployment frequency, providing an aggregated view of the impact of agentic AI integration.

The graph shows the comparison of median values for various software engineering metrics for four different projects before and following the use of AI, highlighting significant differences. These include decreases in cycle time (from 7.1 to 4.0 days), mean time to recovery (from 83 to 51 minutes), and defects per KLOC (from 0.39 to 0.24), while increasing test coverage (from 51% to 69%) and deployment frequency (from 6.8 to 13.2 per week). These findings are project-dependent and are based on a small sample size (N = 4).

4.4 Quality Outcomes

Quality-related metrics further reinforce this improvement pattern. Defect density decreased substantially, whereas test coverage increased significantly across all projects. Flaky test rates, change failure rates, and escaped defects were improved. These findings suggest that the acceleration of development processes did not lead to quality degradation; instead, quality improved alongside productivity.

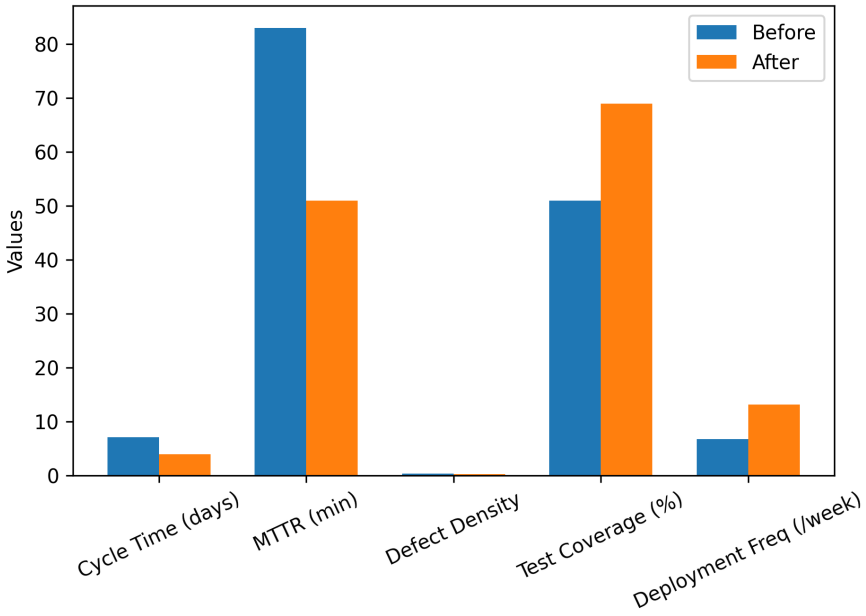

Figure 5 Comparison of key performance metrics before and after agentic AI integration

The relationship between process efficiency and quality outcomes is particularly noteworthy. The increased testing activity reported in Section 4.2 appears to play a key role in supporting these quality gains, indicating that AI facilitates faster and more reliable software delivery.

4.5 Reliability, security, and compliance outcomes

Improvements in process and quality metrics are complemented by improvements in system reliability and security. Mean time to recovery (MTTR) decreased significantly across projects, with reductions of up to 48%, while incident frequency declined by up to 57%. In addition, the system prevented a substantial number of policy violations before deployment, and no compliance breaches or secret exposures were observed during the study period.

These findings highlight the role of agentic AI in enhancing operational stability. The integration of governance mechanisms, as identified in the literature synthesis (Section 4.1), appears to directly contribute to these outcomes by enabling proactive risk detection and mitigation.

4.6 Time-Series Trends

The temporal analysis provides further insight into the dynamics of these improvements. The most significant gains occurred during the early weeks of adoption, particularly between weeks 2 and 6, followed by a period of stabilization. This pattern suggests a learning and adaptation phase in which teams progressively integrate agentic AI into their workflows.

No regression to baseline performance was observed during the 12-week period. Instead, improvements in cycle time, MTTR, and test coverage were sustained over time, indicating that the benefits of adoption are not merely transient but can be maintained as part of ongoing development practices.

The figure 6 illustrates the evolution of selected software engineering metrics over the 12-week postadoption period. The cycle time and mean time to recovery (MTTR) show a steady decline, whereas the test coverage increases progressively. The most significant improvements occurred during the early adoption phase, followed by stabilization, indicating a gradual learning and adaptation process. Values are trends derived from four projects (N = 4).

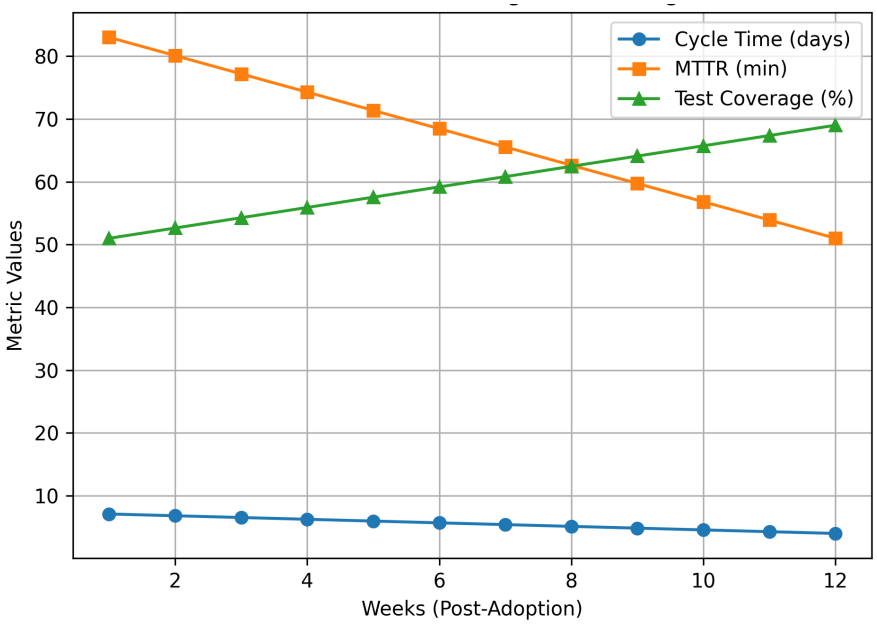

Figure 6 Time-series trends of key performance metrics following agentic AI adoption

5. Discussion

5.1 Findings

This study presents empirical proof based on four enterprise case studies demonstrating that if used within an architectural and governance framework, agentic AI driven by organizational policies can enhance certain performance metrics in software engineering. The analyzed projects showed improvements in the measured metrics. Specifically, the reduction in the cycle time was between 40% and 55%, the defect density decreased by 28% to 45%, and the MTTR improved by 33% to 48%, which is medium to large in effect size. Moreover, the test coverage increased by 11% to 45%, and the deployment frequency almost doubled (94% to 113% increase).

Accordingly, the most practical benefit of using agentic AI in development can be gained in more structured and automated activities, i.e., development, testing, and deployment. This includes significant decreases in glue work, such as writing tests, documenting the code, and refactoring.

While this is true, the results need to be viewed with caution. The results are based on a small sample size of four project groups and can, therefore, only be viewed as context-dependent findings. The consistent improvement in all project cases may be encouraging, but the findings must be validated on a wider scale before any conclusions can be made.

5.2 Relationship with the Prior Literature

The findings of this study can be regarded as generally compatible with existing literature that recognizes the potential for enhancing software engineering capabilities via agentic artificial intelligence and multi-agent systems. Previous studies and industry reports have claimed improvements of 30–50% in terms of time-to-market and 20–35% in terms of productivity, although in most instances, such results were either poorly empirically verified or based on vendor data.

In contrast to the former, the findings provided in this study show reductions in cycle time by 40–55%, reductions in defect density by 28–45%, improvements in MTTR by 33–48%, increases in test coverage by 11–45 percentage points, and improvements in deployment frequency by 94–113%. Although all these results fall within the ranges observed before, or even surpass them, the main difference is that they are derived from an empirical analysis performed within four enterprise teams.

However, it is imperative to take these comparative results with a grain of salt. Contrary to

most studies on the subject matter, this research includes an explicit consideration of governance limitations, policy enforcement processes, and CI/CD incorporation practices, thus affecting the extent and constancy of improvement achieved. In addition, owing to the limited sample of N = 4 organizations studied, the conclusions cannot be generalized into benchmarks for the effectiveness of agentic AI.

Finally, agent-written code comprised approximately 48% of total pull requests, with 31% being accepted without any changes, indicating that the agentic process is significantly human-dependent. This conclusion supports earlier research on the topic, suggesting that human control and hybrid approaches in software engineering cannot be substituted.

This study adds to the available knowledge pool by presenting empirically grounded and quantified results obtained in a governance-aware context.

5.3 Mechanisms and Practical Interpretation

The performance gains seen from the use of agentic AI in this study can be attributed to a combination of different factors:

1. Routine and simple activities (such as writing boilerplate code, performing tests, and documenting) are replaced.

2. Feedback gained through continuous integration and automation of testing.

3. Improved incident management facilitated by artificial intelligence log processing and anomaly detection.

4. Policy enforcement in the CI/CD pipeline.

However, the effectiveness of agentic AI becomes less pronounced when the task demands high-level thinking, which is required in making architectural decisions, dealing with tradeoffs, and negotiating with people.

Thus, one may conclude that the benefit of agentic AI at present lies not in substitution but in complementarity to humans.

5.4 Implications for Governance and Adoption

The study's conclusions offer valuable lessons to organizations contemplating the implementation of agentic AI within environments where governance is a constraint. In essence, the structured adoption process aided by CI/CD tooling, policy enforcement, and human monitoring facilitates demonstrable gains without risking noncompliance.

This conclusion must be taken with a grain of salt, given that the study's implications are drawn from a unique organizational setting where DevSecOps principles are already practiced.

Consequently, the advice offered throughout this work (e.g., beginning with automated testing, introducing graduated autonomy, and using policy gates) must be treated as situation-specific guidance and not universal best practices.

5.5 Limitations and Validity Threats

While this research sheds light on the integration of agentic AI into software engineering, certain limitations associated with the interpretation of the findings should be highlighted.

Internal validity: The current study follows the principles of a quasi-experimental "before and after" design. The main advantage of this design is its ability to evaluate the treatment effect in actual settings. However, this design is still not free from certain potential threats that can interfere with the treatment effect. Some external factors may have caused the improvements that teams experienced during the treatment phase. Although an 8-week baseline period was introduced to minimize this risk, the treatment effect remains somewhat questionable.

Construct validity: A range of software engineering indicators was employed in the current research. These indicators include DORA metrics (lead time and deployment frequency), defect density, and test coverage. Such indicators have been widely adopted to assess software engineering practices across many organizations. Although such metrics are quite useful for measuring software engineering performance and effectiveness, they are still not comprehensive enough as they do not cover such aspects as architecture quality or developers' cognitive abilities.

External validity: The key problem with this study is the small sample size and the exclusive conduct of observations within one organization. Despite the inclusion of a wide range of projects in various sectors (finance, healthcare, data science, and software engineering) and diverse technological stacks, the outcomes are restricted to their contexts and may not apply outside them due to differences in organizational development, management, infrastructure, and developer skills, which can impact the adoption of AI. Thus, the results cannot be generalized.

Conclusion validity: The small sample size and short observation period limit the analytical power. Nonparametric techniques (Wilcoxon signed-rank test) and estimates of effect size have been applied to increase robustness, but the findings must be treated with caution due to the lack of randomization.

Scoping limitations: This study mainly explores the development, testing, and implementation phases within the SDLC because the adoption of agentic AI systems is more advanced in these phases. Other phases, such as requirements and architectural engineering, are mostly conceptual and not extensively addressed. Moreover, the evaluation period (12 weeks from the start of adoption) does not consider the long-term consequences, such as system evolution and technical debt growth.

Future research ideas To overcome the limitations mentioned above, future research should consider multiorganizational designs, large sample sizes, and longitudinal and experimental research methods. In addition, future studies can explore the influence of adopting agentic AI systems at other SDLC stages, especially earlier phases.

6. Conclusions and Prospects

Agentic AI's autonomous and goal-oriented capabilities have revolutionized software engineering practices throughout the lifecycle. Currently, the most valuable applications are found in the development, testing, and deployment/operational phases. Enterprise-level usage of this technology requires a well-architected infrastructure, which includes perception, reasoning, memory, and action, complemented with orchestration, observability, security, and governance. Policy-based guardrails, explainability, reversibility, and human-in-the-loop controls are necessary for safe and effective deployment.

Empirical analysis conducted on four enterprise-level projects shows that Agentic AI governed by policies significantly improves cycle time, test coverage, mean time to recovery, and change failure rates without compromising compliance. The findings offer empirical justification for organizations considering the adoption of Agentic AI. Further research should focus on the following: (1) standardization of interoperability and common protocols for multi-agent systems, (2) integration of formal verification methods for agent behavior, (3) alignment and guardrails ensuring safety and mitigating goal drift/misuse, (4) cost-conscious reasoning to enhance performance, minimize latency, and optimize token expenditure, and (5) long-term longitudinal analysis at an organizational level. These findings provide initial empirical support for the effectiveness of agentic AI in enterprise software engineering; however, broader validation across multiple organizations and larger samples is required before generalizable conclusions can be drawn.

Acknowledgements

The authors would like to acknowledge the support of Prince Sultan University for covering the Article Processing Charges (APC) of this publication.

Author Contributions

Mamdouh Alenezi: Conceptualization, methodology, literature review, empirical study design, writing – original draft, writing – review & editing. **Mohammed Akour:** Conceptualization, methodology, empirical study execution, data collection, statistical analysis, writing – original draft, writing – review & editing, corresponding author. **Omaia Al-Omari:** Literature review, theoretical framework development, governance model design, writing – review & editing. All authors reviewed and approved the final manuscript.

Conflict of Interest

The authors declare no conflicts of interest.

Declaration of AI

No AI tools were used to generate the manuscript text, results, or statistical analyses.

References

Abou-Ali, M., Dornaika, F., & Charafeddine, J. (2025). Agentic AI: A comprehensive survey of architectures, applications, and future directions. *Artificial Intelligence Review*, *59*(11). https://doi.org/10.1007/s10462-025-11422-4

Alenezi, M., & Akour, M. (2025). AI-driven innovations in software engineering: A review of current practices and future directions. *Applied Sciences*, *15*(3), 1344. https://doi.org/10.3390/app15031344

Allmendinger, S., Bonenberger, L., Endres, K., Fetzer, D., Gimpel, H., & Kühl, N. (2026). Multi-agent AI. *Electronic Markets*, *36*(18). https://doi.org/10.1007/s12525-025-00862-z

Al-Omari, O., Alyousef, A., Fati, S., Shannaq, F., & Omari, A. (2025). Governance and ethical frameworks for AI integration in higher education: Enhancing personalized learning and legal compliance. *Journal of Ecohumanism*, *4*(2), 80–86. https://api.semanticscholar.org/CorpusID:275464187

AlSayyad, A., Huang, K. Y., & Pal, R. (2026). Agenttrace: A structured logging framework for agent system observability. *LLM-based multi-agent systems: Towards responsible, reliable, and scalable agentic systems*. https://doi.org/10.48550/arXiv.2602.10133

Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., & Zimmermann, T. (2019). Software engineering for machine learning: A case study. *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, 291–300. https://doi.org/10.1109/ICSE-SEIP.2019.00042

Arinze, C. A., Izionworu, V. O., Isong, D., Daudu, C. D., & Adefemi, A. (2024). Integrating artificial intelligence into engineering processes for improved efficiency and safety in oil and gas operations. *Open Access Research Journal of Engineering and Technology*, *6*(1), 39–51.

Atri, S. (2025). Trustworthy agentic AI: Balancing autonomy with human oversight. *International Journal of Computer Science and Mobile Computing*, *14*(3), 112–128. https://doi.org/10.47760/ijcsmc.2025.v14i11.013

Burlutskaya, Z. V., Sharko, P. A., Gintciak, A. M., & Zakharov, P. A. (2026). Intelligent decision support system based on multi-agent interaction models by the example of the oil and gas industry. *International Journal of Technology*, *17*(2), 344–358. https://doi.org/10.14716/ijtech.v17i2.8242

Burte, S. (2025). AI-powered software development life cycle: From requirements to maintenance. *AI Systems Engineering*, *1*(1), 1–8. https://doi.org/10.64229/ykh1jf83

Chevrot, A., Vernotte, A., Falleri, J. R., Blanc, X., Legeard, B., & Cretin, A. (2025). Are autonomous web agents good testers? *Proceedings of the ACM on Software Engineering*, *2*(ISSTA), 206–228. https://doi.org/10.1145/3728879

Dhayakar, D. (2025). The four pillars of enterprise agentic AI readiness: A strategic framework for organizational implementation. *Journal of Engineering and Computer Sciences*, *4*(8), 588–597.

Durrani, U. K., Akpinar, M., Adak, M. F., Kabakus, A. T., Öztürk, M. M., & Saleh, M. (2024). A decade of progress: A systematic literature review on the integration of AI in software engineering phases and activities (2013–2023). *IEEE Access*, *12*, 171185–171204. https://doi.org/10.1109/ACCESS.2024.3488904

Ekundayo, F. (2024). Leveraging AI-driven decision intelligence for complex systems engineering. *International Journal of Research Publication and Reviews*, *5*(11), 1–10.

Ganapathi, J. K. (2025). AI-driven product development: Cognitive software delivery at enterprise scale. *Journal of Computer Science and Technology Studies*. https://doi.org/10.32996/jcsts.2025.7.8.99

Gangavarapu, R. (2025). AI governance: Preparing for the rise of agentic AI. In *Mastering AI governance: A guide to building trustworthy and transparent AI systems* (pp. 111–119). Springer. https://doi.org/10.1007/978-3-031-93681-4_12

Garg, V. (2025). Designing the mind: How agentic frameworks are shaping the future of AI behavior. *Journal of Computer Science and Technology Studies*, *7*(5), 182–193. https://doi.org/10.32996/jcsts.2025.7.5.24

Gody, R., Goudy, M., & Tawfik, A. Y. (2025). ConvoGen: Enhancing conversational AI with synthetic data: A multi-agent approach. *2025 IEEE Conference on Artificial Intelligence (CAI)*, 252–257. https://doi.org/10.1109/CAI64502.2025.00046

Haidemariam, T. (2025). From the logic of coordination to goal-directed reasoning: The agentic turn in artificial intelligence. *Frontiers in Artificial Intelligence*, *8*, 1728738. https://doi.org/10.3389/frai.2025.1728738

Hosseini, S., & Seilani, H. (2025). The role of agentic AI in shaping a smart future: A systematic review. *Array*, 100399. https://doi.org/10.1016/j.array.2025.100399

Hughes, L., Dwivedi, Y. K., & Malik, T. (2025). AI agents and agentic systems: A multi-expert analysis. *Journal of Computer Information Systems*, 489–517. https://doi.org/10.1080/08874417.2025.2483832

Kumar, R., & Yadav, A. K. (2025). Automating software development pipelines with artificial intelligence (AI). *International Journal of Leading Research Publication*, *6*(7), 1–8.

Kusmenko, E., Pavlitskaya, S., Rumpe, B., & Stüber, S. (2019). On the engineering of AI-powered systems. *2019 34th IEEE/ACM International Conference on Automated Software Engineering Workshop (ASEW)*, 126–133. https://doi.org/10.1109/ASEW.2019.00035

Luo, H., Liu, Y., Zhang, R., Wang, J., Sun, G., Niyato, D., Yu, H., Xiong, Z., Wang, X., & Shen, X. (2025). Toward edge general intelligence with multiple-large language model (Multi-LLM): Architecture, trust, and orchestration. *IEEE Transactions on Cognitive Communications and Networking*. https://doi.org/10.1109/TCCN.2025.3612760

Modi, M. D. B. (2024). Transforming software development through generative AI: A systematic analysis of automated development practices. *International Journal*, *10*(6), 536–547.

Ozurumba, E., & Eboh, I. P. (2024). Leveraging AI-driven decision intelligence for systems engineering complexity. *International Journal of Research*, *5*(11), 4374–4389.

Raji, I. D., Smart, A., White, R. N., Mitchell, M., Gebru, T., Hutchinson, B., & Barnes, P. (2020). Closing the ai accountability gap: Defining an end-to-end framework for internal algorithmic auditing. *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, 33–44. https://doi.org/10.1145/3351095.3372873

Rombaut, B., Masoumzadeh, S., Vasilevski, K., Lin, D., & Hassan, A. E. (2025). Watson: A cognitive observability framework for the reasoning of llm-powered agents. *2025 IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 739–751.

Shaikh, S. H. (2025). LLM-based multi-agent systems: Frameworks, evaluation, open challenges, and research frontiers. *International Joint Conference on Computational Intelligence*, 149–170. https://doi.org/10.1007/978-3-032-15632-7_9

Spieser, J., Balapour, A., Meller, J., Patra, K. C., & Shamsaei, B. (2026). A review of multi-agent AI systems for biological and clinical data analysis. *Methods and Protocols*, *9*(2), 33. https://doi.org/10.3390/mps9020033

Tamanampudi, V. M. (2024). AI-enhanced continuous integration and continuous deployment pipelines. *Distributed Learning and Broad Applications in Scientific Research*, *10*, 56–96.

Venkiteela, P. (2026). An enterprise agentic architecture framework for governance and scalable autonomy. *Scientific Journal of Computer Science*, *2*(1), 1–17. https://doi.org/10.64539/sjcs.v2i1.2026.368

Whulanza, Y., Kusrini, E., Budiyanto, M. A., Arnas, & Skhvediani, A. (2026). Bridging technological sovereignty and global progress through open science. *International Journal of Technology*, *17*(2), 322–328. https://doi.org/10.14716/ijtech.v17i2.8510

Whulanza, Y., Kusrini, E., Harwahyu, R., Fitri, I. R., & Asvial, M. (2025). Viewing ai studies from an ijtech perspective. *International Journal of Technology*, *16*(6), 1888–1893.

Wu, Q., Bansal, G., Zhang, J., Wu, Y., Li, B., Zhu, E., & Wang, C. (2024). Autogen: Enabling next-gen llm applications via multi-agent conversations. *First Conference on Language Modeling*.

Zheng, Y., Hu, Y., Yu, T., & Quinn, A. (2025). Agentsight: System-level observability for ai agents using eBPF. *Proceedings of the 4th Workshop on Practical Adoption Challenges of ML for Systems*, 110–115. https://doi.org/10.1145/3766882.3767169