World Journal of Advanced Research and Reviews
eISSN: 2581-9615 CODEN (USA): WJARAI
Cross Ref DOI: 10.30574/wjarr
Journal homepage: https://wjarr.com/

(RESEARCH ARTICLE)

Check for updates

SPEC-DRIVEN DEVELOPMENT FOR ENTERPRISE AGENTIC SOFTWARE ENGINEERING: A GOVERNANCE FRAMEWORK FOR RELIABLE AI-GENERATED SOFTWARE

Sachin Suryawanshi *

Even & Odd Minds LLC, United States
* Corresponding Author; Email: sachinmcm009@gmail.com

ORCID Details
Sachin Suryawanshi: https://orcid.org/0009-0001-8464-908X

World Journal of Advanced Research and Reviews, 2026, 31(03), 881–886

Publication history: Received on 31 July 2026; revised on 07 September 2026; accepted on 09 September 2026

Article DOI: https://doi.org/10.30574/wjarr.2026.31.3.2332

ABSTRACT

AI coding assistants are evolving into agents that can plan, edit repositories, run tools, test software, and prepare deployable changes. Natural-language prompts are useful for expressing intent, but they rarely encode the complete set of enterprise requirements that govern architecture, cybersecurity, reliability, and operations. This study proposes the Specification Governance and Validation Framework (SGVF), a design-science artifact that treats structured specifications as an external control plane for agentic software engineering. SGVF separates requirements into functional, architecture, security, and operational contracts; introduces a Specification Compliance Score (SCS) with mandatory control vetoes; classifies change risk to determine permissible autonomy; and records traceability evidence from business intent through deployment. The framework was evaluated analytically through six enterprise scenarios and qualitative comparison with traditional, prompt-based, vibe-coding, and ungoverned agentic approaches. The evaluation shows how SGVF produces explicit release decisions for authorization failures, architecture drift, vulnerable dependencies, operational regressions, privileged changes, and compliant low-risk work. The study does not claim measured productivity or defect-reduction gains. It provides a reproducible governance model and a basis for future empirical validation of specification-driven enterprise AI development.

Keywords: Agentic AI; Spec-Driven Development; Software Governance; AI Coding Agents; DevSecOps; Enterprise Architecture

1 INTRODUCTION

Large language models now support requirements, design, implementation, testing, maintenance, and software management. A systematic review of 395 studies shows broad adoption of LLM-based software engineering while identifying unresolved issues around reliability, evaluation, and workflow integration [1]. The next step is more consequential: coding agents can receive goal-level tasks, inspect repositories, modify multiple files, invoke tools, run tests, and iterate with less direct human intervention. SWE-bench demonstrates why repository-level work is difficult,

while ChatDev and MetaGPT show that structured workflows and role decomposition can improve multi-agent software development [2-4].

Enterprise risk increases as autonomy grows. A prompt may describe desired behavior but omit approved technology boundaries, identity rules, dependency policies, data classifications, performance objectives, or deployment restrictions. Security research has also shown that AI-generated code can contain exploitable weaknesses and that developers using AI assistants may produce less secure code while remaining confident in the result [5,6]. A functionally correct agent-generated change can therefore still be unacceptable.

Specification-Driven Development (SDD) offers a promising control mechanism by treating the specification as a primary artifact that guides planning and implementation. GitHub's Spec Kit operationalizes this idea through a specification, plan, tasks, implementation, and convergence workflow, and recent academic work frames SDD as a contract substrate for human-agent collaboration [7,8]. The remaining enterprise gap is enforcement. This paper asks: How can machine-verifiable specifications govern autonomous coding agents while preserving functional correctness, architectural consistency, security, operational reliability, and human accountability? The proposed answer is SGVF, which adds four enforceable specification contracts, compliance scoring, mandatory gates, risk-based autonomy, and auditable traceability.

2 RELATED WORK AND RESEARCH GAP

LLM-based software engineering increasingly spans the whole lifecycle [1]. Agent frameworks such as ChatDev and MetaGPT demonstrate that structured communication and operating procedures can reduce some inconsistencies in collaborative generation [3,4]. Requirements engineering research, however, continues to identify challenges in correctness, context, validation, and scaling LLM use to complex systems [9]. SDD directly addresses part of this problem by making specification artifacts persistent inputs to generation rather than disposable documentation [7,8].

Security and governance research suggests that structure must extend beyond functional requirements. Studies of AI code generation have reported insecure outputs under security-sensitive conditions [5,6]. NIST SSDF recommends integrating security throughout the software lifecycle [10], and SP 800-218A adds generative-AI-specific secure development practices [11]. The NIST Generative AI Profile emphasizes lifecycle risk management, while OWASP identifies supply-chain risk, improper output handling, and excessive agency among important GenAI risks [12,13]. Existing work therefore provides strong pieces, but not a unified enterprise mechanism that binds functional intent, architecture, security, operations, autonomy, and release evidence into one decision path. SGVF is designed to fill that gap.

3 MATERIALS AND METHODS

This study uses design science research because its principal contribution is an engineering artifact. The method follows the problem identification, objective definition, design, demonstration, evaluation, and communication sequence described by Peffers et al. [14]. The problem was defined as governance loss when AI agents can translate broad intent directly into software changes. Solution objectives were derived from peer-reviewed software engineering research, contemporary SDD practice, and NIST/OWASP guidance: traceability, enforceable constraints, security-by-design checks, operational validation, risk-sensitive autonomy, and auditable decisions.

Evaluation used six predefined hypothetical enterprise scenarios: an API missing authorization, introduction of an unapproved cloud database, a latency regression, a vulnerable dependency, a compliant low-risk change, and modification of privileged identity configuration. Each scenario was traced through the framework's contracts, mandatory gates, compliance score, and autonomy rules. A qualitative comparison with traditional development, prompt-based AI coding, vibe coding, and ungoverned agentic development assessed whether each approach contains explicit mechanisms for traceability, architecture governance, security governance, operational governance, automated validation, auditability, and risk-based human oversight. This is an analytic architecture evaluation. No production deployment, survey, benchmark improvement, or defect-reduction result is claimed.

4 PROPOSED SPECIFICATION GOVERNANCE AND VALIDATION FRAMEWORK

SGVF places governance outside the coding model. Business intent enters a versioned specification registry and is decomposed into four contracts. Enterprise standards provide reusable policies. An AI agent may plan and generate code, tests, infrastructure-as-code, configuration, or documentation, but artifacts cannot advance until an external validation engine evaluates them against the contracts. Security and quality gates, risk classification, human approval where required, CI/CD, and runtime feedback complete the lifecycle shown in Figure 1.

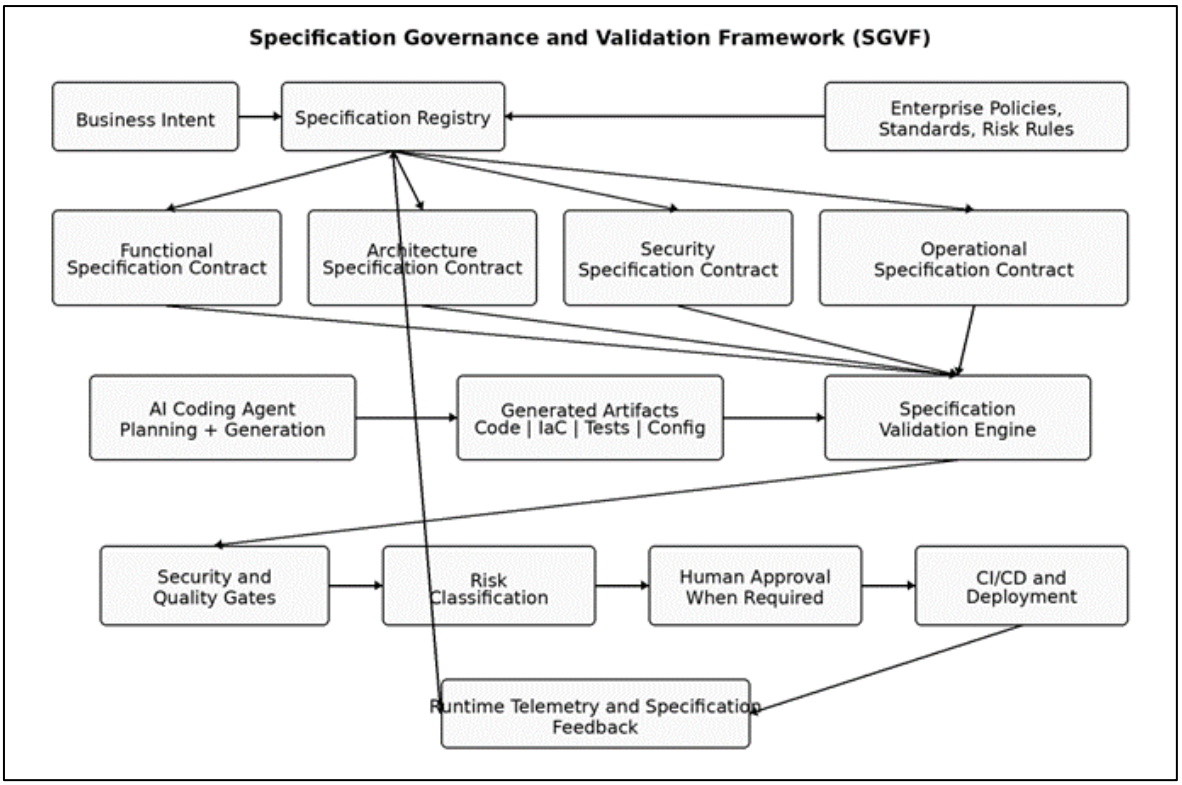

Figure 1: Specification Governance and Validation Framework (SGVF)

4.1 Four specification contracts

Table 1: SGVF contracts and representative controls

Contract	Purpose	Representative controls
Functional	Defines expected business behavior.	Business rules, API behavior, acceptance criteria, tests, invariants.
Architecture	Constrains solution structure and technology.	Service boundaries, approved dependencies, data stores, APIs, cloud services, prohibited choices.
Security	Protects identities, data, code, and supply chain.	Authentication, authorization, secrets, encryption, SAST, SCA, DAST, data classification.
Operational	Defines safe production behavior.	Performance, resilience, observability, scalability, cost, rollback, SLOs.

The separation prevents functional success from masking failures in another dimension. For example, passing tests cannot prove that a service uses an approved identity pattern, and secure code may still violate latency or resilience requirements. Rules are versioned so that each approval can be reconstructed against the specification and policy state used at decision time.

4.2 Compliance score, mandatory gates, and autonomy

Non-mandatory compliance is summarized by the Specification Compliance Score: $SCS = w_fF + w_aA + w_sS + w_oO$, where F, A, S, and O are normalized values from 0 to 1 and $w_f + w_a + w_s + w_o = 1$. A change is eligible only when SCS >= threshold T and every mandatory control passes. The mandatory condition is essential: a high aggregate score cannot compensate for exposed credentials, broken authorization, or another critical failure.

SGVF also assigns an Autonomy Risk Level. Low-risk changes such as documentation or isolated tests may progress automatically after validation. Medium-risk business-logic changes can require asynchronous review. High-risk database, public API, or infrastructure changes require explicit senior approval. Critical identity, secrets, payment, production security-policy, or destructive infrastructure changes require restricted automation and multi-party approval. Figure 2 shows the decision flow.

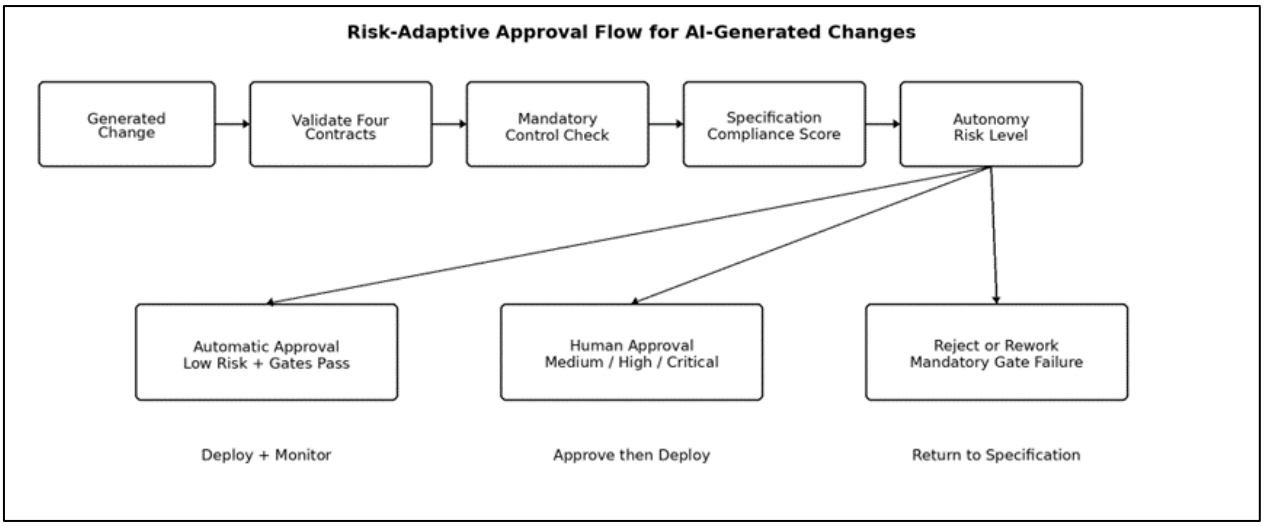

Figure 2: Risk-adaptive approval flow for AI-generated changes

4.3 Traceability and feedback

Every change carries an evidence bundle linking business intent, specification versions, agent plan, generated artifacts, validation output, mandatory gate results, SCS components, risk classification, human approvals, deployment identity, and relevant runtime telemetry. Runtime incidents and operational regressions can create new or strengthened specification rules. This makes traceability bidirectional and reduces the common drift between documentation and deployed behavior.

5 RESULTS AND DISCUSSION

5.1 Scenario evaluation

Table 2 summarizes the analytic walkthrough. The purpose is to test whether SGVF produces an explicit, explainable disposition for each failure mode, not to claim measured prevention rates.

Table 2: Scenario-based analytic evaluation

Scenario	Contract	Risk	SGVF disposition
API omits authorization	Security	High	Mandatory failure blocks release.
Unapproved cloud database	Architecture	High	Architecture drift is rejected.
Latency exceeds SLO	Operational	Medium/High	Performance rule requires rework or approved exception.
Vulnerable dependency	Security	High	Dependency policy blocks release.
Compliant isolated change	All	Low	Automatic approval is allowed if threshold and gates pass.
Privileged identity change	Security/Architecture	Critical	Restricted automation and multi-party approval.

Three results follow from the walkthrough. First, governance is multidimensional: functional correctness cannot authorize a change by itself. Second, mandatory vetoes preserve non-negotiable controls even when weighted compliance is high. Third, human oversight becomes selective. SGVF does not oppose automation and review; it allocates review according to risk while retaining objective validation for every change.

5.2 Comparative analysis and enterprise implications

Traditional development can achieve strong governance, but evidence and traceability are often assembled manually. Prompt-based coding provides faster implementation but usually lacks persistent architecture and operational contracts. Vibe coding further prioritizes interactive intent over durable specification, while ungoverned agents add autonomy without a corresponding enterprise control plane. SGVF's distinguishing feature is the externalized contract

system: the generation model proposes artifacts, but independent checks determine whether those artifacts satisfy the enterprise's current rules.

This design aligns with secure software development guidance because SGVF coordinates existing controls rather than replacing them. Static analysis, dependency scanning, secret detection, policy-as-code, architecture tests, functional test suites, and operational checks remain independent sources of evidence [10,11]. The framework aggregates those results into a release decision and therefore reduces dependence on the coding agent to judge its own output. Organizations must also govern the specifications themselves. Specification repositories need ownership, version control, access restrictions, provenance, and review. A coding agent may propose a specification change, but it should not silently weaken the security contract that evaluates its own work. Related zero-trust research for enterprise agentic AI likewise emphasizes continuous verification, scoped delegation, tool-level enforcement, runtime observability, and containment of consequential autonomous actions [15].

5.3 Limitations and future research

SGVF is a conceptual architecture and has not yet been validated in a production enterprise. SCS weights and thresholds require calibration, ambiguous specifications can create false confidence, and some architecture or operational requirements cannot be reduced to deterministic rules. Future research should implement SGVF in a reproducible repository, compare agentic development with and without the framework on benchmark tasks, measure policy-violation and defect-escape rates, examine review workload, and study safe specification evolution in long-lived systems.

6 CONCLUSION

Agentic software engineering moves software governance closer to the point where AI systems plan and execute changes. Natural-language prompts alone are too narrow to represent the complete enterprise contract. SGVF addresses this gap with four specification contracts, a weighted compliance signal, mandatory vetoes, risk-adaptive autonomy, human approval, traceability evidence, and runtime feedback. The six scenario walkthroughs show how the framework can make release decisions explicit for common authorization, architecture, dependency, performance, and privilege risks while still permitting automation for compliant low-risk work. The central proposition is not that specifications make AI-generated software automatically safe, but that governed, testable, and enforceable specifications provide a stronger foundation for accountable enterprise agentic development.

STATEMENTS ON COMPLIANCE WITH ETHICAL STANDARDS

Acknowledgments

The author acknowledges the publicly available research, standards, and technical documentation cited in this study. No external research funding was received.

Disclosure of conflict of interest

The author declares no conflict of interest.

Data availability

No new empirical dataset was collected. The evaluation uses the framework definitions, scenarios, and public sources described in the manuscript.

REFERENCES

[1] Hou X, Zhao Y, Liu Y, Yang Z, Wang K, Li L, et al. Large language models for software engineering: A systematic literature review. ACM Trans Softw Eng Methodol. 2024;33(8):1-79. doi:10.1145/3695988.

[2] Jimenez CE, Yang J, Wettig A, Yao S, Pei K, Press O, et al. SWE-bench: Can language models resolve real-world GitHub issues? In: International Conference on Learning Representations; 2024.

[3] Qian C, Liu W, Liu H, Chen N, Dang Y, Li J, et al. ChatDev: Communicative agents for software development. Proc 62nd ACL. 2024:15174-15186. doi:10.18653/v1/2024.acl-long.810.

[4] Hong S, Zhuge M, Chen J, Zheng X, Cheng Y, Wang J, et al. MetaGPT: Meta programming for a multi-agent collaborative framework. In: International Conference on Learning Representations; 2024.

[5] Pearce H, Ahmad B, Tan B, Dolan-Gavitt B, Karri R. Asleep at the keyboard? Assessing the security of GitHub Copilot's code contributions. 2022 IEEE Symposium on Security and Privacy. doi:10.1109/SP46214.2022.9833571.

[6] Perry N, Srivastava M, Kumar D, Boneh D. Do users write more insecure code with AI assistants? Proc ACM CCS. 2023. doi:10.1145/3576915.3623157.

[7] GitHub. Specification-Driven Development, Spec Kit documentation [Internet]. 2026 [cited 2026 Sep 7]. Available from: https://github.com/github/spec-kit/blob/main/spec-driven.md

[8] Diaz J, Gayoso J, Cimminio A, Perez J. Spec-Driven Development for Agentic Software Engineering: Harnessing Human-Agent Teamwork. arXiv [Preprint]. 2026; arXiv:2609.00252.

[9] Norheim JJ, Rebentisch E, Xiao D, Draeger L, Kerbrat A, de Weck OL. Challenges in applying large language models to requirements engineering tasks. Des Sci. 2024;10:e16. doi:10.1017/dsj.2024.8.

[10] Scarfone K, Souppaya M, Dodson D. Secure Software Development Framework (SSDF) Version 1.1. NIST SP 800-218. 2022. doi:10.6028/NIST.SP.800-218.

[11] Booth H, Souppaya M, Vassilev A, Ogata M, Stanley M, Scarfone K. Secure Software Development Practices for Generative AI and Dual-Use Foundation Models. NIST SP 800-218A. 2024. doi:10.6028/NIST.SP.800-218A.

[12] Autio C, Schwartz R, Dunietz J, Jain S, Stanley M, Tabassi E, et al. Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile. NIST AI 600-1. 2024. doi:10.6028/NIST.AI.600-1.

[13] OWASP Gen AI Security Project. OWASP Top 10 for LLM and GenAI Applications 2025 [Internet]. 2025 [cited 2026 Sep 7]. Available from: https://genai.owasp.org/llm-top-10/

[14] Peffers K, Tuunanen T, Rothenberger MA, Chatterjee S. A design science research methodology for information systems research. J Manag Inf Syst. 2007;24(3):45-77. doi:10.2753/MIS0742-1222240302.

[15] Suryawanshi S. Security architecture for agentic AI in enterprise cloud environments: A zero-trust framework for secure autonomous systems. International Journal of Innovative Science and Research Technology. 2026;11(8). Available from: https://doi.org/10.38124/ijisrt/26aug1168