International Journal of Computer Information Systems and Industrial Management Applications
ISSN 2150-7988 Volume 18(17s) 2026 pp. 1248-1260
https://doi.org/10.70917/ijcisim-2026-4825
Article

Productivity vs. Compliance: The New Engineering Challenge of AI Coding Assistants in Regulated Codebases

Ashutosh Pal

Guru Jambheshwar University of Science & Technology, Hisar - Haryana (India) - GJUS&T
ashutosh1978pal@gmail.com

Abstract: Background: Large language model (LLM)-based coding assistants have moved from experimental developer utilities into everyday software engineering practice. Contemporary studies show that these tools can improve task completion, code comprehension, and routine implementation speed, while also introducing verification costs, usability limits, and security uncertainty. In regulated industries, the relevant question is not only whether AI assistance improves productivity, but whether AI-assisted changes can be governed as part of an auditable control environment. Objective: This article develops a codepath-aware governance framework for AI-assisted engineering in regulated codebases, with emphasis on financial services, payments, healthcare, and other domains where software changes may affect legal, operational, privacy, and audit obligations. Methods: A targeted integrative conceptual review was conducted using recent evidence from 2024-2026 across AI-assisted programming, code-generation security, secure software development, responsible AI governance, and regulated-sector technology risk. Sources were selected for recency, authenticity, publisher quality, regulatory relevance, and direct usefulness for software delivery governance. Results: The synthesis identifies a productivity-compliance asymmetry: AI tools can accelerate code production faster than many organizations can adapt review, provenance, and evidence practices. The article proposes a five-layer governance model comprising regulated scope mapping, AI assistance policy, provenance and accountability, reviewer and control-owner routing, and audit evidence generation. Conclusion: Regulated organizations should avoid both blanket bans and unconstrained adoption. A more defensible approach is controlled acceleration: AI assistance is permitted, but its behavior changes according to the regulatory sensitivity of the codepath being modified. The resulting engineering system should make AI-assisted changes useful, bounded, attributable, reviewable, and auditable.
Keywords: AI coding assistants, regulated codebases, software engineering governance, secure software development life cycle (SDLC), code provenance

1. Introduction

AI coding assistants have changed the practical texture of software engineering. Developers increasingly use assistants to generate boilerplate, summarize unfamiliar code, draft unit tests, migrate APIs, refactor legacy modules, and reason about compiler or runtime errors. Recent empirical and human-computer interaction studies report meaningful productivity effects, but they also show that AI assistance shifts rather than eliminates developer effort. The human engineer remains responsible for validating intent, context, security, maintainability, and downstream behavior [1]-[3].

This distinction becomes decisive in regulated codebases. In consumer software, an AI-generated edit that is incorrect may be a regression, a user-experience defect, or an availability incident. In payments, banking, insurance, healthcare, identity verification, and public-sector systems, the same edit can also be a compliance failure. A line of code may determine whether cardholder data is redacted before logging, whether a transaction is reversible or reportable, whether a sanctions-screening decision is preserved, whether a patient-

record access event is auditable, or whether a data-retention rule is correctly enforced. The software artifact is therefore part of the control environment, not merely part of the product.

The engineering literature has begun to describe both the promise and the limits of AI-assisted programming. Large-scale studies of programming assistants document usability challenges, context limitations, and developer concerns about generated output [3], [6]. Empirical evaluations of code generation continue to show that LLMs can support developers but require careful task framing and verification [4], [5]. Recent reviews of human-AI experience in development environments and GenAI-assisted code comprehension further show that benefits depend on task context, developer verification, and tool integration rather than model output alone [11], [12]. Security-focused work further indicates that AI-generated code can contain weaknesses or reproduce insecure patterns unless organizations deliberately design safeguards around its use [7]-[10].

Meanwhile, regulatory and standards bodies are sharpening expectations for technology risk, secure software development, operational resilience, and AI governance. NIST has extended secure software development guidance to generative AI and dual-use foundation models [15], while its Generative AI Profile to the AI Risk Management Framework emphasizes lifecycle governance, risk mapping, measurement, and management [16]. OWASP has also formalized risks particular to LLM applications, including prompt injection, sensitive-information disclosure, and excessive agency [17]. These developments do not directly prescribe how an engineer should use a code assistant inside an examined repository. They do, however, make clear that generative AI must be governed through repeatable organizational controls.

1.1 Research motivation

The immediate motivation for this article is the widening gap between tool adoption and compliance readiness. AI coding assistants are being incorporated into integrated development environments, code review workflows, repository platforms, and software delivery agents faster than many regulated enterprises can update their secure development lifecycles. A developer may receive an AI suggestion inside a file that implements a payment authorization rule, a customer due-diligence workflow, a clinical access-control check, or a data redaction utility. If the organization lacks codepath-aware governance, that suggestion is treated like any other suggestion even though its compliance significance is very different.

This creates an institutional blind spot. A conventional secure software development life cycle (SDLC) may require code review, static analysis, test evidence, and change approvals. Yet those controls often do not record whether AI assistance was used, what kind of assistance occurred, whether regulated context was included in prompts, whether generated code modified a control-relevant path, or whether reviewers understood that the change was AI-assisted. Productivity is visible in sprint velocity; AI provenance and compliance assurance are often invisible.

1.2 Problem statement

The problem addressed in this article is that regulated engineering organizations lack a practical, toolchain-level model for governing AI coding assistants according to the regulatory sensitivity of the code being changed. Existing research provides valuable evidence on productivity, usability, code generation, and security weaknesses. Regulatory documents provide obligations for data protection, operational resilience, payment security, and AI governance. However, these streams are rarely synthesized into an engineering governance framework that can operate inside the developer workflow.

A generic AI-use policy is insufficient. It may state that developers should not paste secrets into public models, that generated code must be reviewed, or that only approved tools may be used. Such policies are necessary but not enough. They do not distinguish between writing non-sensitive documentation and changing authentication logic; between generating a test helper and modifying a ledger; between refactoring an internal dashboard and changing patient-record access rules. In regulated codebases, risk is codepath-specific. Governance must therefore be codepath-aware.

1.3 Research question and objective

The guiding research question is: How can regulated engineering organizations adopt AI coding assistants while preserving evidence-grade control over codepaths that implement regulatory, privacy, security, financial, or clinical obligations?

The objective is to develop a conceptual framework for AI-assisted engineering governance that is independent of any single vendor or model provider. The framework is intended for engineering leaders, security architects, compliance teams, internal auditors, and practitioners designing AI-assisted software delivery systems in regulated environments.

1.4 Contributions of the article

This article makes four contributions. First, it reframes AI-assisted development in regulated industries as a productivity-compliance governance problem rather than a tooling adoption problem. Second, it defines regulated codepaths as units of governance that connect repository structure, data flow, and control obligations. Third, it proposes a five-layer codepath-aware governance model that can be embedded into developer tools, pull-request workflows, CI/CD pipelines, and audit evidence repositories. Fourth, it outlines implementation priorities and future research directions for organizations seeking to capture AI productivity gains without weakening compliance assurance.

2. Methodology

This article uses a targeted integrative conceptual review design. The purpose is not to conduct a statistical meta-analysis or a systematic literature review with exhaustive database coverage. Instead, the article synthesizes recent scholarly and authoritative evidence into a practitioner-relevant conceptual framework for regulated software engineering. This design is appropriate because AI-assisted coding in regulated codebases is an emerging and cross-disciplinary subject: the evidence base spans software engineering, human-computer interaction, security, responsible AI, secure development frameworks, and sector-specific regulatory guidance.

2.1 Study design

The methodological approach follows the logic of conceptual synthesis. Evidence is used to identify patterns, tensions, and design requirements rather than to estimate a single effect size. The article therefore gives equal attention to engineering workflow, governance architecture, auditability, and practical control implementation.

2.2 Evidence sources and selection logic

Sources were limited to the 2024-2026 period to meet the recency expectations of a forward-looking journal submission and to reflect the rapidly changing state of AI-assisted software development. The evidence base includes peer-reviewed articles from ACM, IEEE, Springer, and other reputable venues; official standards and guidance from NIST, OWASP, PCI Security Standards Council, and public authorities; and selected regulatory or supervisory publications relevant to financial services, operational resilience, AI, payments, and healthcare data protection.

Table 1: Evidence domains used for the integrative conceptual synthesis

Evidence domain	Representative source type	Contribution to the manuscript
AI-assisted programming productivity and developer experience	ACM CHI, ACM EICS, ICSE, Communications of the ACM, and systematic reviews	Establishes the productivity promise, human verification burden, and usability limits of AI programming assistants [1]-[6], [11], [12].

Code-generation security and vulnerability evidence	IEEE Security and Privacy, IEEE workshops, ACM TOSEM, Empirical Software Engineering	Supports the argument that AI-generated code requires security review, guardrails, and empirical validation rather than default trust [7]-[10], [25].
Secure software development and generative AI risk governance	NIST SSDF profile, NIST Generative AI Profile, OWASP LLM guidance	Provides lifecycle governance concepts for provenance, control design, risk mapping, and AI-specific security threats [15]-[17].
Regulated-sector obligations	PCI DSS, HHS HIPAA Security Rule materials, Treasury, FSB, Basel Committee, EU AI Act, DORA guidance	Defines the regulatory context in which software changes may affect payment security, healthcare privacy, financial stability, AI governance, and ICT operational resilience [18]-[24].
Responsible AI and regulation-to-practice translation	ACM responsible AI and human-centered regulation studies	Informs the need to translate broad regulatory expectations into usable guidance for technical and non-technical roles [13], [14].

2.3 Inclusion and exclusion criteria

Sources were included when they met at least three criteria: publication or release between 2024 and 2026; relevance to AI-assisted programming, code generation, secure SDLC, responsible AI governance, or regulated technology risk; publication through a reputable academic venue, recognized standards body, or official public authority; direct usefulness for understanding AI-assisted engineering in regulated software environments; and availability of a stable DOI, publisher page, official document record, or equivalent authenticity marker.

Sources were excluded when they were vendor marketing materials without methodological detail, unverifiable web commentary, duplicate summaries of primary sources, documents outside the 2024-2026 window, or materials that did not contribute directly to the governance framework. Because the article focuses on contemporary practice, older foundational work on software engineering governance, compliance management, and platform tooling is acknowledged as intellectually relevant but not included in the formal reference list.

2.4 Analytical procedure

The analysis followed a seven-step workflow. First, the regulated-engineering problem was framed through the tension between AI-assisted productivity and compliance assurance. Second, evidence domains were selected to cover productivity, developer experience, security, secure development, regulation, and responsible AI governance. Third, sources were screened for quality and relevance. Fourth, themes were extracted concerning AI output reliability, verification cost, vulnerability risk, provenance, auditability, and code review. Fifth, the themes were synthesized into a codepath-aware governance model. Sixth, the model was mapped to concrete engineering controls and audit evidence. Seventh, limitations and future research needs were identified.

Figure 1: Methodological workflow for the integrative conceptual synthesis.

3. Results: Conceptual Synthesis

The synthesis produced five principal findings. First, AI coding assistants create a productivity-compliance asymmetry: code production can accelerate faster than review, provenance, and evidence practices adapt. Second, regulated codepaths require differentiated treatment because they implement controls rather than ordinary application behavior. Third, AI-generated code security findings justify control-based adoption rather than trust-based adoption. Fourth, codepath-aware governance can translate broad AI risk principles into enforceable engineering workflow. Fifth, audit evidence must be generated as a by-product of development rather than reconstructed after release.

3.1 AI coding assistants create a productivity-compliance asymmetry

Productivity studies demonstrate why AI-assisted development cannot be ignored. Weber et al. report significant productivity gains from programming with large language models, while Mozannar et al. show that AI-assisted programming changes user behavior and imposes distinct interaction costs [1], [2]. Large-scale survey evidence further indicates that developers value AI assistance but encounter problems related to correctness, integration into workflow, and trust calibration [3], [6].

For regulated organizations, the critical implication is not simply that developers may complete routine tasks faster. The deeper implication is that the rate of change entering review pipelines may increase, while control practices remain tuned for pre-AI development velocity. Code review queues, security scans, test coverage, domain-expert

availability, and audit documentation may become the new bottlenecks. When production speed increases without corresponding improvements in assurance, the organization experiences a productivity-compliance asymmetry.

This asymmetry is visible in routine scenarios. An assistant can quickly generate a logging change, but it cannot independently know whether the log line might expose cardholder data. It can refactor an authorization function, but it may not understand that a particular branch is relied on by an internal access-control attestation. It can simplify an error-handling path, but it may remove an audit event needed to reconstruct a payment dispute. The code may compile and tests may pass, while compliance-relevant intent is lost.

3.2 Regulated codepaths are not ordinary codepaths

A regulated codepath is a portion of a software system whose behavior is materially connected to a regulatory, contractual, privacy, financial, security, clinical, audit, or operational-resilience obligation. The unit of analysis is broader than a file and narrower than an entire enterprise system. It can be a service, endpoint, library, schema, data pipeline, configuration module, authorization check, audit emitter, redaction function, or workflow branch.

In payment environments, codepaths may be in scope because they store, process, or transmit cardholder data, or because they support required security controls under PCI DSS v4.0.1 [18]. In healthcare environments, codepaths may be in scope because they access or protect electronic protected health information (PHI) and therefore relate to confidentiality, integrity, and availability obligations under HIPAA Security Rule expectations and proposed strengthening of cybersecurity safeguards [19]. In financial services, codepaths may support model governance, customer onboarding, fraud prevention, operational resilience, ICT risk management, or supervisory reporting, all of which have attracted intensified policy attention in recent AI and digitalisation publications [20]-[22], [24].

This codepath framing matters because the same AI assistance action has different risk meanings in different locations. A generated helper function in a non-sensitive internal tool may create ordinary maintainability risk. A generated helper function in a payment tokenization module may affect data security and certification posture. A test generated for a public API may improve coverage. A test generated for sanctions-screening logic may become part of evidence used to justify a release. Governance should therefore attach to codepaths, not merely to users or tools.

3.3 Security evidence supports control-based adoption

Security research cautions against assuming that AI-generated code is safe merely because it appears plausible. Hamer et al. compare vulnerabilities in ChatGPT-generated code with Stack Overflow answers, showing that AI code generation can participate in familiar patterns of insecure reuse [7]. Natella et al. frame AI code generators as both potential security aids and sources of security risk, depending on how they are applied and validated [8]. Fu et al. provide empirical evidence of security weaknesses in Copilot-generated code in GitHub projects [9], and Sajadi et al. examine whether LLMs consider security when responding to programming questions [10].

These findings do not imply that AI-generated code is inherently worse than human-written code. The proper conclusion is more precise: generated code requires a control environment that assumes plausible but unverified output. AI can accelerate drafting, but it should not bypass threat modeling, secure coding review, test design, static analysis, secrets scanning, dependency review, and domain-specific validation. Emerging

work on using LLMs to enhance static analysis further suggests that AI can strengthen defense when it is positioned as an assurance aid rather than an unconstrained code author [25].

3.4 Codepath-aware governance translates AI risk into engineering workflow

The operational relationship among scope, assistance, provenance, review, and evidence is summarized in Figure 2. The model is deliberately tool-agnostic so that it can be implemented through different IDEs, repository platforms, CI/CD systems, and audit-evidence repositories.

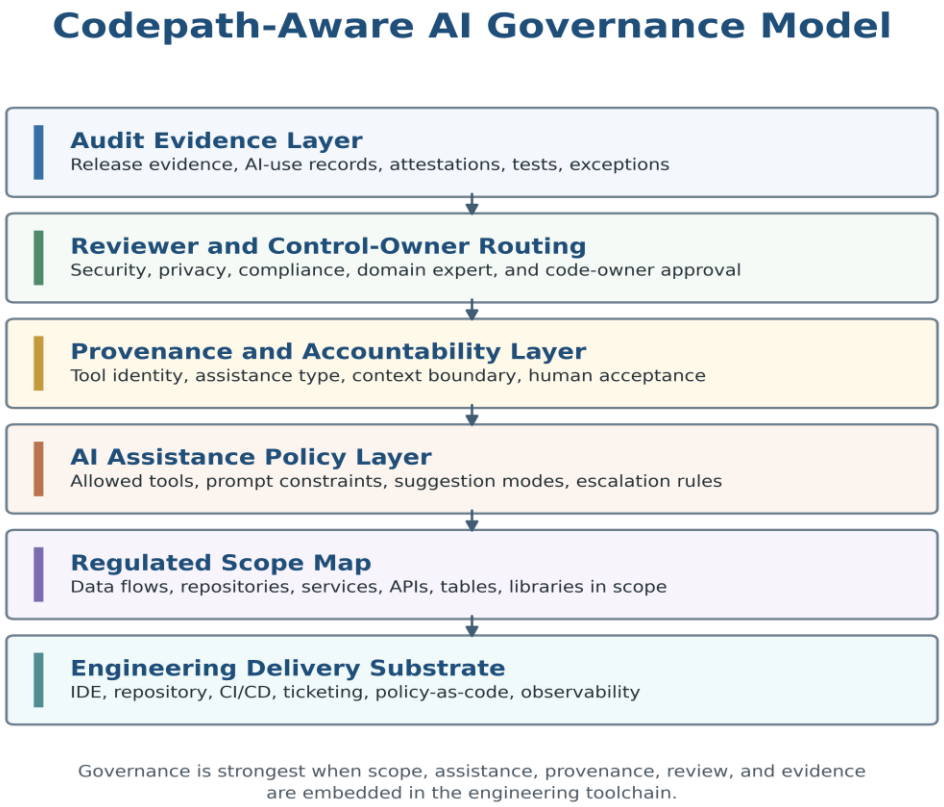

Figure 2: Codepath-aware AI governance model for regulated software engineering.

Responsible AI and regulatory guidance often operate at the level of organizational principles: accountability, transparency, human oversight, risk management, documentation, and lifecycle governance [13], [14], [16]. These principles are necessary but insufficient unless translated into developer workflow. A software engineer needs to know whether AI assistance is allowed in a particular repository, what context may be shared, whether generated suggestions require an AI-use tag, which reviewer must approve the change, and what evidence must be retained.

The proposed codepath-aware governance model contains five operational layers built on the engineering delivery substrate. The regulated scope map identifies where control-relevant code exists. The AI assistance policy layer determines what assistance is permitted in that scope. The provenance and accountability layer records how AI contributed and which human accepted responsibility. The reviewer and control-owner routing layer ensures that sensitive changes reach qualified approvers. The audit evidence layer preserves the records needed to demonstrate that the process operated effectively.

3.5 Mapping regulated codepath risk to AI assistance controls

The practical value of the model lies in differentiated controls. Regulated organizations already classify data, systems, vendors, and production access according to risk. AI-assisted development should extend the same

Table 2: Mapping regulated codepath categories to AI assistance controls

Regulated codepath category	Representative AI-assisted failure mode	Minimum control posture	Evidence expected at review or audit
Payment and cardholder-data handling	Generated logging, serialization, or error-handling code exposes	Approved AI tools only; no sensitive prompt context; mandatory	AI-use attestation, redaction test results, security scan output,

	cardholder data or weakens redaction.	security review; PCI-aware test cases; data loss prevention (DLP) and secrets scanning.	code-owner approval, release record [18].
Money movement and ledger integrity	Generated refactor changes idempotency, reconciliation, rounding, reversal, or settlement behavior.	AI suggestions allowed only with domain-expert review; deterministic tests; reconciliation checks; change rationale required.	Ledger-impact analysis, test artifacts, reviewer signoff, deployment approval, exception record if controls are bypassed.
Know Your Customer (KYC), Anti-Money Laundering (AML), fraud, and sanctions workflows	Generated conditional logic changes verification thresholds, audit events, or decision traceability.	Scope-aware reviewer routing; model/policy owner review; audit-event validation; production monitoring after release.	Control-owner approval, decision-trace tests, audit-log verification, monitoring ticket, regulatory-impact note [20]-[22].
Patient records and PHI access	Generated access-control or caching code exposes protected health information or weakens availability safeguards.	Restricted context sharing; privacy and security review; access-control regression tests; PHI-safe logging validation.	Privacy review evidence, access-control test results, audit-log evidence, HIPAA-security mapping [19].
Authentication, authorization, and cryptography	Generated simplification weakens authorization checks, token validation, session handling, or cryptographic configuration.	AI explanation and test generation permitted; direct generated changes require senior security approval; threat model update.	Threat-model delta, secure-code review, static analysis, dependency review, penetration-test ticket if material [7]-[10], [17].
Audit logging, retention, and reporting	Generated cleanup removes control events, reduces retention, changes schema semantics, or disrupts reporting lineage.	Mandatory provenance; control-owner routing; schema compatibility checks; audit-log replay tests.	Before/after audit-event map, retention-control validation, reviewer approval, release evidence package.

logic to codepaths. Table 2 summarizes representative codepath categories, AI failure modes, required controls, and evidence artifacts.

4. Discussion

The central implication of the analysis is that productivity and compliance should not be treated as opposing objectives. The false choice is a binary one: either ban AI coding assistants in regulated codebases or allow them everywhere in the name of velocity. Neither position is durable. A blanket ban may be administratively simple, but it can drive use into unsanctioned channels, reduce transparency, and prevent legitimate productivity and assurance benefits. Unconstrained adoption may produce short-term delivery gains, but it creates undocumented risk when AI-assisted changes touch control-relevant codepaths.

The more defensible operating model is controlled acceleration. In this model, AI assistance remains available to engineers, but the toolchain responds to the sensitivity of the codepath. Low-risk code may permit broad drafting and refactoring assistance. Regulated-adjacent code may require provenance tags, targeted tests, and code-owner review. Control-critical code may restrict AI to explanation, test generation, or review support unless an expert approves generated modifications. The user experience is therefore not uniform. It is governed by scope.

4.1 From policy documents to engineering controls

Many regulated enterprises begin AI governance with policy language. Policies are necessary because they define approved tools, acceptable use, prohibited disclosures, human accountability, and escalation channels. However, policy language alone does not prevent a developer from accepting an unsafe suggestion in a sensitive file. Mature governance requires implementation in the engineering system itself: repository labels, CODEOWNERS rules, CI gates, prompt-boundary controls, data-loss-prevention checks, pull-request templates, automated reviewer routing, and release evidence retention.

NIST SP 800-218A is useful because it reinforces the idea that generative AI should be integrated into secure software development practices rather than handled as an afterthought [15]. The NIST Generative AI Profile similarly emphasizes lifecycle risk governance [16]. For engineering organizations, the practical translation is direct: AI-assisted development should have an inventory, scope map, risk tiering, review protocol, monitoring process, and evidence trail. The strongest control is not a paragraph in a policy; it is a workflow that makes the compliant path the easiest path.

4.2 What future examiners and auditors are likely to ask

Regulators and auditors may not require deep model introspection for every code assistant. They are more likely to ask questions that map to existing technology-risk disciplines. Which AI tools are approved for engineering use? Which repositories or codepaths can those tools access? How does the organization prevent sensitive data from being submitted to inappropriate models? Which production changes were AI-assisted? What human reviewed and accepted the change? What tests, scans, and approvals supported release? What exceptions were granted and who approved them?

These questions are consistent with trends in payment security, healthcare cybersecurity, financial-services AI risk, and operational resilience. PCI DSS v4.0.1 continues to frame payment security through explicit requirements and testing procedures [18]. HHS materials emphasize strengthened cybersecurity expectations for electronic protected health information [19]. The U.S. Treasury, Basel Committee, and Financial Stability Board have each highlighted AI-related governance, model, operational, and concentration risks in financial services [20]-[22]. The EU AI Act and DORA-related operational-resilience guidance further illustrate the direction of travel: regulated institutions are expected to demonstrate governance, documentation, oversight, and resilience rather than merely assert them [23], [24].

4.3 Provenance is not blame; it is accountability infrastructure

A common cultural barrier is the fear that AI-use disclosure will be interpreted as developer weakness or misconduct. That interpretation should be avoided. Provenance is not a mechanism for blame. It is accountability infrastructure. Engineering organizations already track commit authors, reviewers, ticket links, build artifacts, deployment approvals, and incident records. AI-use provenance extends the same logic to a new class of development assistance.

A lightweight provenance record should capture the approved tool used, the type of assistance, the codepath risk tier, whether regulated context was included or excluded, the human who accepted the final change, and any additional controls triggered by scope. The organization need not store raw prompts containing sensitive information. In fact, storing sensitive prompts may create new risk. Metadata, attestations, hashes, prompt-class labels, and tool-use logs can often provide sufficient evidence without creating unnecessary data exposure.

4.4 Reviewer routing should reflect control ownership

Ordinary code review often routes by repository ownership. Regulated code review should also route by control ownership. A change to a redaction utility should reach the team accountable for data protection and logging controls. A change to ledger logic should reach engineers who understand reconciliation and settlement semantics. A change to PHI access logic should reach privacy-aware security reviewers. A change to model-assisted fraud triage should reach governance stakeholders who understand both technical and policy implications.

This does not mean every regulated change must become bureaucratic. It means that routing must be proportionate. The goal is to avoid both under-review and over-review. Under-review allows material control changes to ship without qualified oversight. Over-review slows ordinary work and causes teams to route

around the process. Codepath-aware automation can reduce this burden by escalating only when sensitive files, directories, services, schemas, or data flows are affected.

4.5 AI can strengthen assurance when positioned correctly

The argument is not that AI coding assistants are only a risk. AI can also strengthen assurance. Assistants can help developers understand legacy control logic, generate negative test cases, identify missing edge cases, explain complex authorization paths, draft threat-model prompts, compare policy requirements with tests, and improve documentation. Research on LLM-assisted static analysis points toward the possibility of using AI to augment vulnerability detection rather than merely generate code [25].

The distinction is role design. An unconstrained assistant that directly edits critical code without context-aware controls is risky. An assistant that helps produce tests, summarize change impact, explain control flow, or support reviewer analysis can improve assurance. Regulated organizations should therefore design AI adoption patterns around both productivity and verification.

4.6 Implementation roadmap

A realistic adoption roadmap begins with visibility before enforcement. Organizations should first understand what tools are used, where they are used, and which repositories are most sensitive. They can then introduce scope labels, AI-use attestations, and reviewer routing. Only after the basic map exists should organizations add stricter gates, exception workflows, and evidence automation. Table 3 summarizes an implementation roadmap for codepath-aware governance.

Table 3: Implementation roadmap for codepath-aware AI governance in regulated codebases

Implementation phase	Primary objective	Engineering actions	Risk and compliance output
0-90 days: Visibility and policy baseline	Establish approved use and discover current practice.	Create tool inventory; define approved assistants; publish prompt and data-use rules; identify high-risk repositories; update PR templates with optional AI-use disclosure.	AI tool register, acceptable-use policy, preliminary regulated repository inventory, initial training record.
3-6 months: Scope-aware controls	Move from generic policy to differentiated engineering workflow.	Classify codepaths; add CODEOWNERS rules; configure DLP/secrets scanning; require AI provenance tags for regulated paths; route sensitive changes to qualified reviewers.	Regulated scope map, reviewer routing matrix, provenance metadata schema, exception process.
6-12 months: Evidence automation	Generate audit evidence as a normal by-product of delivery.	Integrate AI-use metadata with CI/CD; preserve test and scan artifacts; link tickets to release approvals; build dashboards for AI-	Release evidence packages, control attestations, audit-ready change records, periodic compliance reports.

		assisted regulated changes.	
12+ months: Assurance optimization	Use AI to strengthen verification while reducing manual burden.	Deploy AI-assisted test generation for regulated paths; use LLMs to support static-analysis triage; benchmark defect and review outcomes; refine policy based on incidents and metrics.	Measured productivity and assurance outcomes, model/tool performance reports, governance improvement backlog.

4.7 Metrics for accountable adoption

A codepath-aware program should be evaluated through both productivity and assurance metrics. Productivity metrics may include cycle time, review latency, test-generation time, developer satisfaction, and time to remediate defects. Assurance metrics should include the share of regulated changes with AI-use provenance, percentage of sensitive changes routed to correct reviewers, test coverage for control-relevant paths, number of AI-assisted defects discovered before release, policy exceptions, prompt-data violations, and audit-evidence completeness. Mature organizations should avoid celebrating velocity alone. The more meaningful outcome is reduced time to safe, reviewable, and compliant change.

5. Conclusion

AI coding assistants are becoming part of the normal software engineering environment. Their productivity potential is real, but regulated software delivery requires a broader standard than speed. In codebases that govern payment data, money movement, KYC and AML workflows, patient records, audit logs, authorization logic, retention rules, and operational resilience, an AI-generated edit can affect both system behavior and compliance posture.

This article argued that regulated organizations should not choose between blanket prohibition and unrestricted adoption. The stronger approach is codepath-aware governance: the engineering toolchain understands which codepaths are regulated, adapts the permitted AI assistance accordingly, records provenance, routes review to qualified control owners, and preserves release evidence. This model converts AI use from an informal developer behavior into a governed component of the secure SDLC.

The central insight is that compliance should be treated as a design constraint for trustworthy acceleration. Organizations that embed governance into developer workflow can capture AI productivity benefits without relying on after-the-fact manual reconstruction. In the next phase of AI-assisted engineering, competitive advantage in regulated industries will belong not to the firms that adopt assistants the fastest, but to those that can prove their adoption is bounded, accountable, and auditable.

6. Limitations

This manuscript is a conceptual integrative review rather than an empirical field study. It does not analyze private repository data, developer telemetry, audit findings, production incident records, or controlled experiments inside regulated institutions. The proposed framework should therefore be viewed as a governance model requiring empirical validation rather than as proof of operational effectiveness.

The reference window is intentionally limited to 2024-2026. This satisfies the recency requirement for the manuscript but excludes older foundational work on secure software development, compliance engineering, technology risk management, and human factors in programming. The analysis also does not constitute legal advice. Regulatory obligations differ across jurisdictions, institution types, data architectures, outsourcing arrangements, and contractual commitments.

The article is vendor-neutral and does not compare specific AI coding assistants, model providers, repository platforms, or IDE integrations. Vendor-specific evaluation would require technical testing, contractual review, data-processing analysis, and operational assessment that are outside the scope of this conceptual manuscript.

7. Future Directions

Future research should empirically test the codepath-aware governance model in regulated engineering organizations. Useful study designs include controlled comparisons of AI-assisted and non-AI-assisted changes in regulated repositories, longitudinal analysis of review outcomes, and incident-based studies of AI-assisted defects discovered before and after release. Researchers should measure not only code throughput, but also evidence completeness, reviewer burden, defect severity, control breakage, and remediation time.

A second research direction is the development of standard AI provenance schemas for software delivery. Such schemas should capture tool identity, assistance type, context boundary, codepath risk tier, human acceptance, reviewer routing, and evidence artifacts without requiring storage of sensitive prompt content. Standardization would help internal audit, external assurance, and cross-organization comparability.

A third direction is the construction of regulated-code benchmarks. Existing code-generation benchmarks rarely model the compliance significance of redaction, audit logging, sanctions screening, ledger integrity, PHI

access, retention, or operational resilience. Benchmark tasks that include control semantics would allow researchers to evaluate whether AI assistants preserve or weaken compliance-relevant behavior.

Finally, future work should examine how AI can strengthen assurance rather than only accelerate implementation. Promising areas include AI-assisted test generation for control paths, summarization of regulatory impact in pull requests, static-analysis triage, documentation of audit-event lineage, and continuous verification of regulated data flows.

REFERENCES

1. T. Weber, M. Brandmaier, A. Schmidt, and S. Mayer, "Significant Productivity Gains through Programming with Large Language Models," Proceedings of the ACM on Human-Computer Interaction, vol. 8, no. EICS, Article 256, pp. 1-29, 2024, doi: 10.1145/3661145.
2. H. Mozannar, G. Bansal, A. Fourney, and E. Horvitz, "Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming," in Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI 2024), 2024, pp. 1-16, doi: 10.1145/3613904.3641936.
3. J. T. Liang, C. Yang, and B. A. Myers, "A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges," in Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE 2024), 2024, pp. 52:1-52:13, doi: 10.1145/3597503.3608128.
4. K. Jin, C.-Y. Wang, H. V. Pham, and H. Hemmati, "Can ChatGPT Support Developers? An Empirical Evaluation of Large Language Models for Code Generation," in Proceedings of the 21st IEEE/ACM International Conference on Mining Software Repositories (MSR 2024), 2024, pp. 167-171, doi: 10.1145/3643991.3645074.
5. X. Hou et al., "Large Language Models for Software Engineering: A Systematic Literature Review," ACM Transactions on Software Engineering and Methodology, vol. 33, no. 8, Article 220, pp. 1-79, 2024, doi: 10.1145/3695988.
6. M. Khemka and B. Houck, "Toward Effective AI Support for Developers: A Survey of Desires and Concerns," Communications of the ACM, vol. 67, no. 11, pp. 42-49, 2024, doi: 10.1145/3690928.
7. S. Hamer, M. d'Amorim, and L. Williams, "Just another copy and paste? Comparing the security vulnerabilities of ChatGPT generated code and StackOverflow answers," in 2024 IEEE Security and Privacy Workshops (SPW), 2024, pp. 87-94, doi: 10.1109/SPW63631.2024.00014.
8. R. Natella, P. Liguori, C. Improta, B. Cukic, and D. Cotroneo, "AI Code Generators for Security: Friend or Foe?" IEEE Security & Privacy, vol. 22, no. 5, pp. 73-81, 2024, doi: 10.1109/MSEC.2024.3355713.
9. Y. Fu et al., "Security Weaknesses of Copilot-Generated Code in GitHub Projects: An Empirical Study," ACM Transactions on Software Engineering and Methodology, vol. 34, no. 8, Article 218, 2025, doi: 10.1145/3716848.
10. A. Sajadi, B. Le, A. Nguyen, K. Damevski, and P. Chatterjee, "Do LLMs consider security? an empirical study on responses to programming questions," Empirical Software Engineering, vol. 30, Article 101, 2025, doi: 10.1007/s10664-025-10658-6.
11. A. Sergeyuk, I. Zakharov, E. Koshchenko, and M. Izadi, "Human-AI experience in integrated development environments: a systematic literature review," Empirical Software Engineering, vol. 31, Article 55, 2026, doi: 10.1007/s10664-025-10793-0.

12. Y. Qiao, M. I. H. Shihab, and C. D. Hundhausen, "A Systematic Literature Review of the Use of GenAI Assistants for Code Comprehension: Implications for Computing Education Research and Practice," ACM Transactions on Computing Education, vol. 26, no. 2, pp. 1-33, 2026, doi: 10.1145/3785366.
13. M. Constantinides, E. P. Bogucka, D. Quercia, S. Kallio, and M. Tahaei, "RAI Guidelines: Method for Generating Responsible AI Guidelines Grounded in Regulations and Usable by (Non-)Technical Roles," Proceedings of the ACM on Human-Computer Interaction, vol. 8, no. CSCW2, pp. 1-28, 2024, doi: 10.1145/3686927.
14. M. Constantinides et al., "Implications of Regulations on the Use of AI and Generative AI for Human-Centered Responsible Artificial Intelligence," in Extended Abstracts of the CHI Conference on Human Factors in Computing Systems (CHI EA 2024), 2024, pp. 1-4, doi: 10.1145/3613905.3643979.
15. H. Booth, M. Souppaya, A. Vassilev, M. Ogata, and K. Scarfone, "Secure Software Development Practices for Generative AI and Dual-Use Foundation Models: An SSDF Community Profile," NIST Special Publication 800-218A, National Institute of Standards and Technology, 2024, doi: 10.6028/NIST.SP.800-218A.
16. C. Autio, R. Schwartz, J. Dunietz, S. Jain, M. Stanley, E. Tabassi, P. Hall, and K. Roberts, "Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile," NIST AI 600-1, National Institute of Standards and Technology, 2024, doi: 10.6028/NIST.AI.600-1.
17. OWASP Foundation, "OWASP Top 10 for Large Language Model Applications 2025," OWASP, 2025.
18. PCI Security Standards Council, "Payment Card Industry Data Security Standard: Requirements and Testing Procedures, Version 4.0.1," PCI Security Standards Council, 2024.
19. U.S. Department of Health and Human Services, "HIPAA Security Rule Notice of Proposed Rulemaking to Strengthen the Cybersecurity of Electronic Protected Health Information: Fact Sheet," HHS Office for Civil Rights, 2024.
20. U.S. Department of the Treasury, "Artificial Intelligence in Financial Services," U.S. Department of the Treasury, 2024.
21. Basel Committee on Banking Supervision, "Digitalisation of finance," Bank for International Settlements, 2024.
22. Financial Stability Board, "The Financial Stability Implications of Artificial Intelligence," Financial Stability Board, 2024.
23. European Parliament and Council of the European Union, "Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence," Official Journal of the European Union, 2024.
24. Central Bank of Ireland, "Digital Operational Resilience Act (DORA): Frequently Asked Questions," Central Bank of Ireland, 2026.
25. A. Munson, J. Gomez, and A. A. Cardenas, "With a Little Help from My (LLM) Friends: Enhancing Static Analysis with LLMs to Detect Software Vulnerabilities," in 2025 IEEE/ACM International Workshop on Large Language Models for Code (LLM4Code), 2025, pp. 25-32, doi: 10.1109/LLM4Code66737.2025.00008.