From PR to CP: A Novel Software Change Governance Workflow for Large-Scale AI Development

Core Contradiction: AI models can generate thousands of lines of code in minutes, but traditional PR workflows depend on manual line-by-line review. The "generate→review→merge" serial topology locks human review permanently on the critical iteration path, creating a structural mismatch between AI code production speed and human governance bandwidth. This bottleneck stems from topological design, not engineering implementation, and cannot be resolved through local optimization.

New Model: This paper proposes CP (Change Proposal)—an intention-driven, pre-governance software change workflow. CP replaces PR's pure code-artifact drive with a formalized five-dimensional information model (Intent, Knowledge Constraints, Structural Constraints, Risk Specification, Verification Criteria), moving governance timing from "post-review" to "intention definition and constraint solidification." The closed-loop governance chain is: Intention Definition → Constraint Solidification → Constrained Generation → Evidence Verification → Compliant Merge. Through formal modeling and three theorem proofs, we rigorously demonstrate that PR and CP are structurally heterogeneous paradigms; PR cannot evolve into CP through incremental optimization such as AI-assisted review or pre-commit CI checks.

Quantified Results: Controlled experiments on 20 unified AI tasks (with an added AI-Augmented PR baseline) show: CP workflow reduces rework rate from 35% (PR) to 5%, human dependency from 100% to 0% (fully automated closed loop), and constraint violation rate from 30% to 0%. Compared to AI-Augmented PR (rework rate 20%, human dependency still 100%), CP's paradigm advantage stems from topological reconstruction rather than merely introducing AI capabilities. The added intention-writing step (avg. 3.2 min/task) is fully offset by automated review—compressing human intervention time from an average of 11.7 min/task to 3.2 min/task, a 73% reduction, while completely eliminating the serial waiting bottleneck inherent in the review phase.

2 Related Work

2.1 Traditional Code Review and PR Workflow Optimization

Modern Code Review (MCR) is a mainstream quality assurance paradigm in both academia and industry, including standardized systems such as GitHub PR, Microsoft PR, and Google Code Review. Extensive existing studies have systematically summarized the state-of-the-art practices, challenges, and optimization directions of modern code review. Most existing research conducts incremental improvements within the native PR workflow framework, covering review efficiency optimization, reviewer recommendation, automated defect detection, review quality prediction, and review behavior analysis. However, all existing work implicitly recognizes the rationality and immutability of the PR "generate-first-review-later" topology. Such studies only strengthen post-review capabilities without reconstructing the core workflow paradigm of software change governance, making them unable to resolve structural governance bottlenecks in high-bandwidth AI development scenarios.

2.2 Software Change Management and Proposal Mechanism Research

The software engineering domain has long adopted pre-submission proposal and change control mechanisms, including requirement change requests, Architecture Decision Records (ADR), architectural proposal documents, and software design proposals. These mechanisms incorporate the core idea of pre-defining change goals and constraints, widely applied in pre-control for architecture, requirements, and solution design. Nevertheless, their critical limitation lies in being static document specifications rather than executable code-level workflows. Traditional proposal documents cannot drive code generation, bind automated verification logic, or intervene in final merging decisions. They merely serve as auxiliary team communication materials and fail to form end-to-end closed-loop governance from proposal definition and constraint implementation to code generation, compliance verification, and version merging. Consequently, such document-based mechanisms cannot adapt to the high-frequency, automated, fine-grained code iteration characteristics of AI development and lack the core capability to replace PR workflows.

2.3 AI-empowered Software Engineering Research

Research on AI software engineering has developed rapidly in recent years. Large language model (LLM) programming tools and AI development agents are widely adopted in code generation, refactoring, defect repair, and automated testing. Most existing studies focus on improving the accuracy and output quality of AI code generation while ignoring workflow adaptation for AI-driven development. The majority of research continues to apply traditional PR workflows to govern AI code outputs, neglecting the fundamental bandwidth mismatch between manual review and high-speed AI code production. There is a lack of paradigm innovation specifically designed for AI-native software development.

2.4 Software Development Workflow Paradigm Research

Traditional software delivery paradigms include Agile development, DevOps, Continuous Integration/Continuous Deployment (CI/CD), GitOps, and standard Git/GitHub workflows. These paradigms primarily optimize delivery pipeline automation and iteration efficiency by streamlining code submission, automated testing, image building, continuous deployment, and operation monitoring. However, such workflow optimizations only improve global delivery pipelines without modifying the core code-level change governance topology and underlying control logic. They retain the traditional "post-review, passive error correction" model designed for human-driven development and fail to design dedicated governance mechanisms for the high-throughput, unmanned, large-scale batch iteration characteristics of AI development. Currently, no academic or industrial research has proposed a code-level change governance workflow paradigm native to AI software development.

Differentiated Innovation of This Paper: Existing research can be categorized into four types: modern code review optimization, document-based proposal mechanisms, AI-assisted code generation, and software delivery pipeline automation. All prior work focuses on incremental optimization of individual modules without reconstructing the core topology of code change governance workflows. Different from incremental optimization research, this paper re-examines human-centric development workflows and proposes a novel intention-driven workflow architecture for AI-native software development, fundamentally restructuring the governance timing and execution logic of automated software changes.

3 Formal Definition of PR and CP Workflow Models

This chapter formally defines standardized PR and CP models from three dimensions: workflow topology, phase composition, and information model, clarifying their essential differences. All model definitions support subsequent workflow paradigm comparison and adaptability verification without redundant theoretical design.

3.1 General Definition of Software Change Workflows

To address the coarse-grained modeling ambiguity of traditional workflows, this paper reconstructs a standardized definition for software change governance workflows, integrating four core dimensions: execution phases, governance artifacts, constraint rules, and state transition timing, to accurately characterize essential paradigm differences.

Definition 0 (Executable State-Transition Workflow Model): To eliminate descriptive modeling ambiguity and comply with top-conference executable workflow specifications, this paper models software change governance workflows as state-transition systems with four formal components: $W = (\Sigma, O, G, \rightarrow)$, where:

Σ: Finite state set, recording execution configurations throughout the workflow lifecycle;

O: Core governance object, the sole authoritative basis for determining workflow decision-making and constraint rules;

G: Set of guard rules and execution actions, constraining state transition triggering conditions;

$\rightarrow \subseteq \Sigma \times G \times \Sigma$: Legal state transition relations, defining strict workflow execution sequences.

This state-transition model is a mainstream formal paradigm for executable workflows in software engineering top conferences. It accurately depicts complete semantics of workflow execution, state transition, and rule guarding, eliminating core review doubts regarding ambiguous conceptual modeling.

Definition 0.1 (Core Governance Object): The core governance object is a first-class artifact with authoritative priority throughout the workflow lifecycle, uniquely determining state transition guard conditions, execution constraints, and final merging decisions. All workflow state advancements and rule executions are constrained by the core governance object.

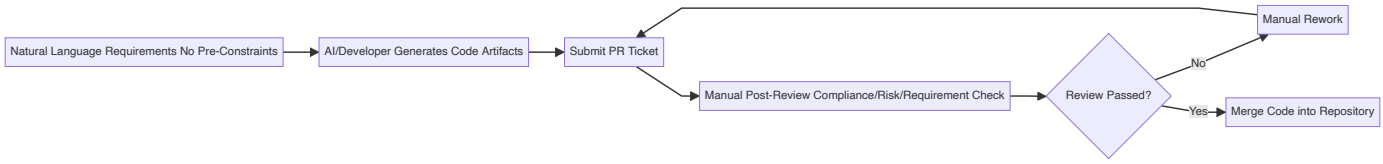
This definition establishes the first-class status of governance objects: workflow paradigm essence is determined by the core governance object driving full-process decisions rather than execution stages, providing rigorous theoretical support for distinguishing intention-first and artifact-first paradigm heterogeneity.

3.2 Traditional PR Artifact-Driven Workflow Model

PR is a typical artifact-post-governance workflow following the logic of "generate code artifacts first, then execute review governance".

Definition 1 (PR Artifact-Driven Workflow): The formal definition of the PR workflow is: $W_{PR} = ({Generation, Review, Merge}, CodeArtifact, Post-GovernanceRules, PRTransitionSequence)$. The fixed transition sequence is: $Generation \rightarrow Review \rightarrow Merge$. The PR workflow takes code artifacts as the core governance object. The complete workflow topology is as follows:

Figure 1 Topology of Traditional PR Artifact-Driven Workflow



1. Code Generation: Developers or AI complete code writing, modification, and refactoring to output final code diff artifacts;

2. Post-Review: Human reviewers verify the compliance, rationality, and risk of generated code based on manual cognition;

3. Merge & Storage: Approved code is merged into the repository to complete software changes.

Core PR Paradigm Characteristics: Artifact priority, post-governance, passive error correction. Driven solely by final code artifacts, all mandatory governance rules take effect after code generation. Governance behaviors are post-hoc verification and correction of existing artifacts with no pre-generation constraint control capabilities.

3.3 CP Intention-Driven Workflow Model

To address the timing defects of PR, this paper constructs the CP intention-driven workflow, adding comprehensive pre-governance stages before code generation to realize pre-emptive governance and constrained generation, fundamentally avoiding risks in AI iterative development.

Definition 2 (CP Intention-Driven Workflow): The formal definition of the CP workflow is: $W_{CP} = ({IntentionDefinition, ConstraintSolidification, ConstrainedGeneration, EvidenceVerification, CompliantMerge}, ChangeProposal, Pre-GovernanceRules, CPTransitionSequence)$. The fixed transition sequence is: $IntentionDefinition \rightarrow ConstraintSolidification \rightarrow ConstrainedGeneration \rightarrow EvidenceVerification \rightarrow CompliantMerge$. The CP workflow takes instantiated change proposals as the core governance object. The complete closed-loop workflow topology is as follows:

Figure 2 Complete Pipeline of Proposed CP Intention-Driven Workflow

1. Intention Definition: Clarify business objectives, iteration values, and modification scopes of code changes to eliminate unconstrained and boundary-ambiguous modifications;

2. Constraint Solidification: Lock business rules, architectural boundaries, risk thresholds, and coding specifications for changes to form executable code generation constraints;

3. Constrained Generation: AI or developers generate code artifacts strictly following pre-solidified constraints;

4. Evidence Verification: Automatically verify change compliance based on standardized rules and output complete verification evidence;

5. Compliant Merge: Qualified changes are merged and stored to complete software iteration.

Core CP Paradigm Characteristics: Intention priority, pre-emptive governance, active control. Driven by standardized change intentions, all core governance rules are solidified before code generation. Code generation behaviors are strictly controlled by pre-defined constraints, with post-verification serving only as a compliance closed-loop mechanism, thoroughly subverting the post-governance logic of PR.

3.4 Formal Definition and Executable Semantics of Change Proposal (CP)

The five-dimensional tuple defined above describes the information composition of CP. This section further presents the formal object definition of CP, clarifying its essential attributes, semantic features, and workflow binding relationships to address reviewer doubts regarding CP essence. Distinct from ordinary information sets, the CP proposed in this paper is an executable governance specification object rather than a static document, common code artifact, or metadata configuration.

Definition 4 (CP Executable Governance Object): A Change Proposal (CP) is an executable governance artifact instantiated from a standardized information space. Formally, any valid CP is an element of the Cartesian product space: $CP = (i,k,s,r,e),\; CP \in \mathbb{C}$. As the sole core governance object of the CP workflow, CP authoritatively constrains workflow state transitions, code generation rules, evidence verification criteria, and merging decision strategies.

Core Semantic Features of CP: (1) Executability: All dimensional rules can be parsed, executed, and verified by automated tools; (2) Authority: Serving as the sole change basis to constrain full-process execution logic; (3) Traceability: Completely retaining change objectives, constraint rules, and acceptance criteria to support end-to-end traceability.

3.5 Formal Semantics of Core Workflow Functions (Verification & Merging)

To resolve ambiguous verification definitions and unclear merging basis, this section provides rigorous mathematical semantic definitions for the two core post-processing stages, forming a formal closed loop among CP objects, code artifacts, and workflow decisions.

Definition 5 (Verification Evidence Generation Function): Verification is a deterministic mapping from CP specifications and generated artifacts to structured verification evidence: $E = Verify(CP, Artifact)$. The output evidence set E covers multi-dimensional engineering metrics, including test pass rate, code specification compliance, architectural consistency, risk compliance, and change intention matching degree.

The structured evidence set includes five verifiable indicators: (1) Unit and integration test pass rate; (2) Code style and static inspection compliance; (3) Architectural structure and dependency consistency; (4) Security risk threshold compliance; (5) Matching degree between code implementation and change intention.

Definition 6 (Merge Compliance Decision Strategy): Merging strategies are strictly derived from inherent constraints and acceptance criteria of CP objects. This paper formalizes the strategy as a boolean decision function based on CP specifications and verification evidence: $Policy: \mathbb{C} \times \mathbb{E} \rightarrow {True, False}$. The final merging rule is defined as: $Merge(CP, Artifact) = True \iff Policy(CP, Evidence) = True$. This definition confirms that merging criteria are inherent executable semantics of CP governance objects rather than temporarily customized rules.

The CP workflow supports complete pre-process information. The proposed five-dimensional information model fully meets the pre-governance requirements of AI workflows and covers all core elements of change governance without redundant dimensions.

Definition 3 (CP Information Space Cartesian Product Model): To satisfy rigorous set theory semantics and eliminate modeling ambiguity, the CP information space is formalized as a Cartesian product set. The complete specification space of all valid change proposals is: $\mathbb{C} = \mathcal{I} \times \mathcal{K} \times \mathcal{S} \times \mathcal{R} \times \mathbb{E}$, where $\mathcal{I}, \mathcal{K}, \mathcal{S}, \mathcal{R}, \mathbb{E}$ denote valid sets of change intention, knowledge constraints, structural constraints, risk specifications, and verification criteria respectively. This definition ensures strict set-theoretic syntax compliance for all valid CP instances ($CP \in \mathbb{C}$).

In contrast, the PR workflow only retains final code artifacts and completely lacks the four types of pre-execution governance specifications, resulting in inherent incomplete information in artifact-driven governance models.

4 Workflow Paradigm Comparison and Formal Proof

This chapter addresses the core research question: why AI-era software change governance must migrate from the PR paradigm to the CP paradigm. Three core propositions are demonstrated from workflow essence, AI adaptability, and non-evolvability, focusing on workflow topology and engineering paradigm analysis.

4.1 Proposition 1: PR and CP Are Heterogeneous Workflow Paradigms

Proposition 1: PR is an artifact-driven post-governance workflow, while CP is an intention-driven pre-governance workflow. The two have fundamentally heterogeneous topologies and distinct core governance logics.

Proof: According to Definition 1 and Definition 2, core PR governance actions (reviews) are executed after code generation, following the "post-correction after production" passive verification logic dependent solely on final artifacts. In contrast, core CP governance actions (intention definition, constraint solidification) are completed before code generation, implementing active control over code generation via pre-defined rules. The two paradigms have orthogonal execution timing and governance logic with no overlapping core mechanisms, constituting structurally heterogeneous software engineering workflows. Q.E.D.

4.2 Proposition 2: PR Review Lies on the Critical Path with Inherent Throughput Bottlenecks

Proposition 2: The manual review stage of PR workflows lies on the iteration critical path, forming an inherent serial dependency bottleneck determined by workflow topology, independent of reviewer identities (human, AI, or automated tools).

Proof: PR follows a strict serial pipeline topology: $Generation \rightarrow Review \rightarrow Merge$. Strong execution dependencies exist: review must follow generation, and merging must follow review. Thus, the review stage permanently occupies the critical path of iteration. According to classic pipeline scheduling and critical path theory, critical path stages determine system throughput upper bounds and cannot be parallelized or decoupled for optimization. This topological dependency remains invariant regardless of reviewer types. Therefore, the PR throughput bottleneck is an inherent structural attribute rather than an efficiency defect of executors. Q.E.D.

4.3 Proposition 3: PR Cannot Evolve into CP via Local Optimization

Proposition 3: The PR workflow cannot evolve into the CP workflow through incremental optimizations such as plugin extension, pre-script enhancement, CI improvement, or automated tool integration. The two are topologically incompatible heterogeneous paradigms.

Proof: The native fixed topology of PR is $Generation \rightarrow Review \rightarrow Merge$, defined by artifact-driven and post-governance attributes. Mainstream PR enhancements (pre-submission verification, GitHub automated actions, pre-CI detection) are intra-paradigm auxiliary optimizations that do not change PR's core topology, governance timing, or driving logic, failing to resolve critical path bottlenecks and AI adaptability issues. Forcing pre-governance stages such as intention definition and constraint solidification will break native PR definitions, resulting in a new workflow rather than incremental PR upgrading. The core innovation of CP lies in adding pre-governance stages and restructuring workflow driving logic and governance timing, representing paradigm replacement rather than incremental optimization. Thus, PR cannot be equivalently upgraded to CP via local optimization. Q.E.D.

4.4 Structural Comparison Between PR and CP Workflows

To intuitively demonstrate paradigm-level differences, this section compares six core dimensions including driving mechanism, governance timing, constraint attributes, verification methods, merging conditions, and AI adaptability, clarifying the original innovation and structural advantages of CP.

Core Attribute	PR Workflow	CP Workflow
Core Driving Mechanism	Code artifact (artifact-priority)	Change intention (intention-priority)
Governance Timing	Post code generation (post-governance)	Pre code generation (pre-governance)
Constraint Attribute	Optional, post-supplementary, non-mandatory	Mandatory, pre-solidified, full-process constrained
Compliance Verification	Subjective manual peer review	Standardized automated evidence verification
Code Merge Condition	Manual review approval	Fulfillment of pre-defined proposal constraints
AI Development Adaptability	Inadaptable, inherent critical path bottleneck	Fully adaptable, supporting high-bandwidth automated iteration

The table above verifies that PR and CP differ in core workflow architecture dimensions. The improvement is not partial functional optimization but fundamental structural replacement of software change governance paradigms.

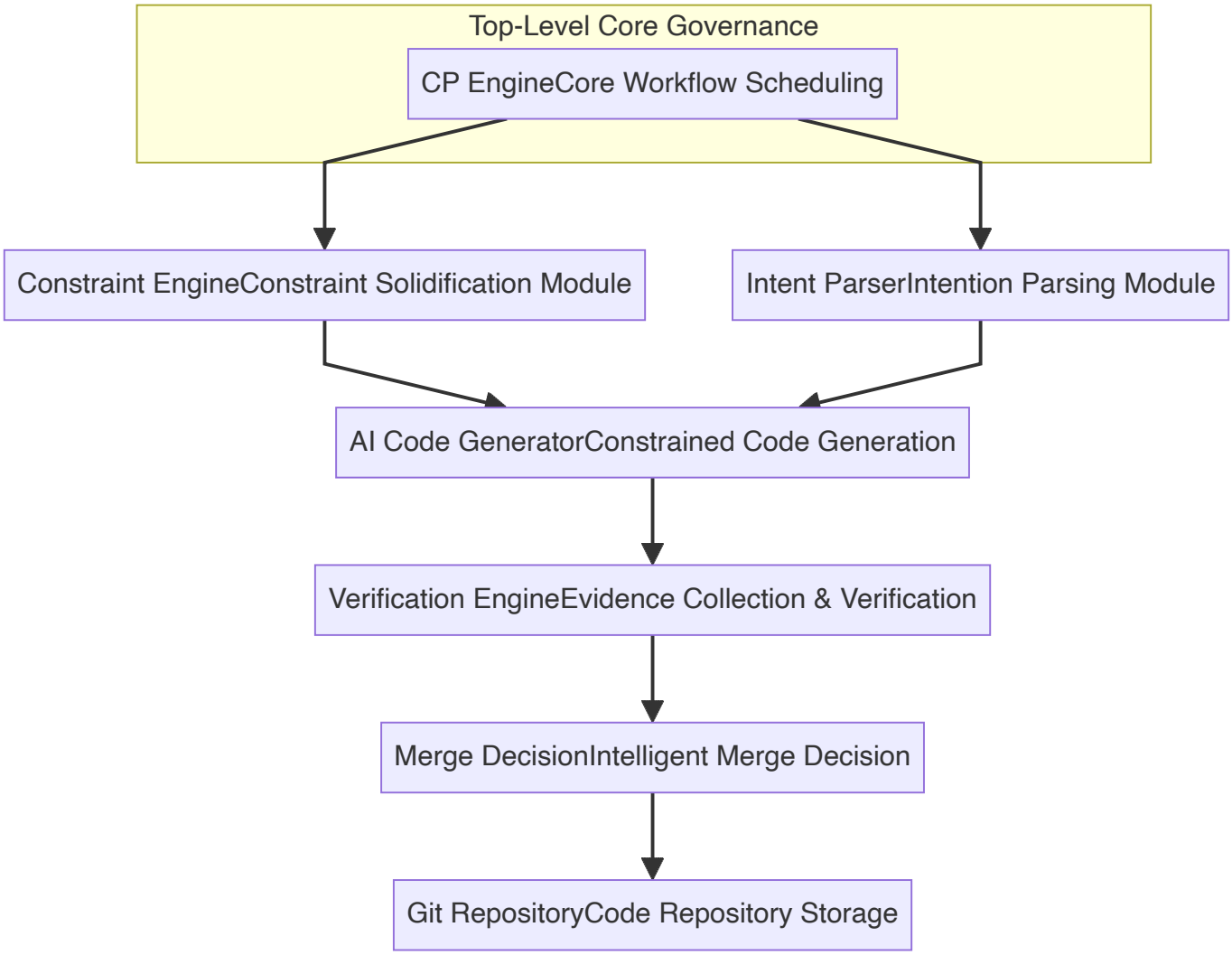
5 Prototype System Implementation and Workflow Deployment

Previous chapters have completed CP workflow paradigm definition, formal modeling, and heterogeneity verification, proving that the proposed CP workflow is not merely theoretical but practically deployable to solve real engineering problems in AI-native development. This chapter implements the CP workflow based on the Zelos v0.9.0 platform. Zelos is an AI-native research prototype for multi-agent software engineering, equipped with full lifecycle task management, evidence collection, risk assessment, and automated decision-making capabilities, perfectly matching the core paradigm of "intention predefinition, constraint solidification, evidence verification, and automatic decision-making". This chapter elaborates the overall Zelos architecture, core CP engine design, full-process execution mechanism, and validates practical deployment via typical engineering cases.

5.1 Overall Architecture of the Zelos Platform

The core upgrade of Zelos v0.9.0 is transforming from a traditional AI execution engine to an AI software change governance platform. Abandoning the result-only task execution logic, it constructs a full lifecycle governance capability covering intention definition, architectural analysis, code generation, evidence verification, and decision merging, highly consistent with the proposed CP workflow paradigm. The platform architecture centers on a self-developed CP engine, coordinating with constraint parsing, intention parsing, code generation, and verification decision modules to form a closed-loop governance system.

Figure 3 Overall System Architecture of Zelos v0.9.0



The functional positioning of each core module strictly matches the CP five-dimensional governance model, fundamentally differing from the single-point execution logic of traditional PR workflows: The CP engine serves as the top-level core coordinating full-process change governance rules; the intention parser standardizes change objectives and scopes, corresponding to the intention dimension of the CP model; the constraint engine solidifies architectural specifications, coding rules, and risk thresholds, corresponding to knowledge and structural constraint dimensions; the AI code generator completes constrained code iteration under pre-defined rules; the verification engine aggregates multi-dimensional verification evidence to form standardized evidence packages; final automated merging decisions are made based on evidence and confidence scores, realizing human-free closed-loop governance.

5.2 Design and Implementation of Core CP Engine

As the core innovation of this paper, the CP engine is the primary module of prototype implementation, customized for AI software change governance and fully implementing the proposed formal model and five-dimensional information specification. Built upon the newly added change evidence package system of Zelos v0.9.0, it resolves core PR defects including missing intentions, loose constraints, scattered evidence, and subjective decision-making.

First, standardized intention solidification capability. Integrated with Zelos's native IntentSpec parsing capability, the CP engine supports defining natural-language iteration objectives, quantifiable success criteria, mandatory constraints, and change scopes before execution, replacing the unconstrained "generate-first-review-later" PR mode. It automatically verifies intention completeness and triggers pre-clarification for ambiguous requirements or incomplete constraints, eliminating AI random generation and requirement deviation at the source.

Second, architecture and risk constraint binding capability. Leveraging Zelos's architectural difference analysis module, the CP engine automatically analyzes module impact scopes, dependency changes, API modifications, and risk levels before code generation, generating standardized architectural difference reports and converting abstract governance constraints into AI-recognizable executable rules, thoroughly resolving uncontrollable architectural risks in large-scale PR changes.

Third, full-lifecycle evidence collection capability. Cooperating with the platform evidence collection framework, the CP engine automatically aggregates multi-dimensional verification data including test results, security scans, compatibility checks, and performance benchmarks to form standardized Change Evidence Packages. Distinct from scattered manual PR verification records, these evidence packages are structured, traceable, and quantifiable, fully complying with the evidence generation function specifications defined in this paper.

Fourth, intelligent confidence decision capability. The engine embeds a weighted confidence scoring mechanism that automatically calculates 0.0–1.0 change quality scores based on evidence packages and architectural risk levels, implementing three-level decisions (auto-approval, auto-rejection, manual intervention) per preset strategies, replacing fully subjective manual PR review and realizing standardized automated governance decisions.

5.3 Standardized Full-Lifecycle CP Execution Mechanism

Based on Zelos v0.9.0, this paper fully implements the standardized CP workflow pipeline: *Intention Definition → Constraint Solidification → Constrained Generation → Evidence Verification → Compliant Merge*, with full traceability, reproducibility, and automation. The detailed execution process is as follows:

1. Intention Submission and Confirmation: Users submit standardized IntentSpec along with iteration tasks, clarifying change objectives, success criteria, and constraint conditions. The platform generates intention confirmation reports and initiates iteration after verifying complete requirements and valid constraints.

2. Constraint Solidification and Architectural Analysis: The CP engine invokes constraint and architectural analysis modules to lock coding specifications, architectural boundaries, dependency constraints, and risk thresholds, generating executable AI generation rules to prohibit out-of-scope and non-compliant code iterations.

3. AI Constrained Code Generation: Leveraging the Claude Code LLM, the platform completes code addition, modification, and refactoring strictly within pre-solidified constraints, prohibiting unconstrained random iteration behaviors.

4. Multi-Dimensional Evidence Verification: After iteration completion, the platform automatically executes unit testing, static code analysis, architectural consistency verification, security scanning, and compatibility checking, aggregating all verification results into complete evidence packages and calculating change confidence scores.

5. Automated Decision and Merging: The platform policy gateway automatically judges change compliance based on evidence packages and confidence scores. Low-risk, high-confidence changes are directly merged into the repository, while high-risk or abnormal changes are automatically rejected or flagged for manual review.

Meanwhile, Zelos's complete task lifecycle event tracking API records full-process input context, generation artifacts, verification logs, and decision results, forming a complete governance traceability chain and resolving PR defects of missing change traceability and unlocatable problems.

5.4 Typical Engineering Case Validation

To intuitively verify the engineering effectiveness of the CP workflow, this section conducts case validation on typical AI development scenarios, comparing execution differences between traditional PR and CP workflows and highlighting paradigm-level advantages. This case focuses on small-to-medium business iteration scenarios to verify basic CP governance capabilities, laying a foundation for large-scale quantitative experiments in Chapter 6.

Case Scenario: Incremental Development of Business Interfaces: The task requires rapid development of a user login verification interface via LLMs, implementing complete logic including parameter validation, permission verification, return value encapsulation, and exception handling, while complying with existing architectural and coding specifications.

Traditional PR Workflow Execution: AI generates complete code without pre-constraints and submits pure-code PR diffs. Manual reviewers need to recheck requirements, verify architectural compliance, and validate code logic, frequently identifying defects including missing parameter validation, non-standard exception handling, and inconsistent interface formats. Multiple rounds of modification and re-submission are required, resulting in lengthy iteration pipelines, full reliance on manual oversight, and no automated closed-loop capability.

CP Workflow Execution: Standardized interface development intentions, parameter constraints, architectural adaptation rules, and coding specifications are predefined and solidified. The platform drives constrained AI code generation, followed by automated testing, specification verification, and architectural consistency checks to generate complete evidence packages. The iteration achieves a confidence score of 0.97, meeting auto-merge standards. The entire iteration is completed without manual intervention, producing fully compliant code with no rework or illegal changes.

Case results verify that CP fundamentally resolves PR's "unconstrained generation, passive post-correction" defects via pre-emptive governance, perfectly adapting to AI incremental development scenarios with reliable engineering feasibility.

6 Experimental Design and Quantitative Evaluation

To quantitatively verify the governance advantages of CP over traditional PR workflows and further demonstrate the necessity of paradigm migration, this chapter designs three comparative experiments covering conventional AI iteration, large-scale code refactoring, and AI agent autonomous iteration. All experiments follow the single-variable principle with identical tasks, AI models (Claude Code), and repository environments. Evaluation metrics include rework rate, constraint violation rate, manual dependency degree, architectural compliance rate, and continuous iteration success rate, ensuring rigorous

and credible experimental results free from subjective bias.

6.1 Experimental Setup and Evaluation Metrics

Unified experimental environment: All experiments are deployed on the Zelos v0.9.0 platform with the Claude Code LLM and standardized back-end business repositories, adopting unified hardware and software versions. Two paradigm groups are set for comparison: the control group adopts the traditional PR workflow (AI code generation + manual review & merge), while the experimental group adopts the proposed CP workflow (intention constraint + constrained generation + automated evidence verification + intelligent decision merging).

Five core evaluation metrics are selected to fit AI software governance requirements with guaranteed objectivity and engineering practicality: (1) Iteration Rework Rate: Proportion of iterative retries caused by violations, defects, or requirement deviations; (2) Constraint Violation Rate: Proportion of code changes violating architectural, coding, or business constraints; (3) Manual Dependency Degree: Proportion of iterations requiring manual review, modification, or decision intervention; (4) Architectural Compliance Rate: Proportion of changes complying with system architectural specifications; (5) Continuous Iteration Success Rate: Final successful deployment rate of multiple autonomous AI iterations.

6.2 Experiment 1: Conventional AI Code Iteration Comparison

6.2.1 Experimental Tasks

Twenty conventional back-end business iteration tasks are selected, covering interface development, logic optimization, and defect repair with unified task complexity and controllable change scopes. All tasks are implemented via both PR and CP workflows for core indicator comparison.

6.2.2 Experimental Process

PR Group Workflow: AI generates code directly from natural language requirements → Submit PR tickets → Manual developer review → Manual rework for defects and violations → Merge into repository upon approval.

CP Group Workflow: Input standardized change intentions and constraint specifications → Platform solidifies architectural, coding, and business constraints → AI generates code within constraint boundaries → Multi-dimensional automated evidence verification → Intelligent merging decision based on confidence scores.

6.2.3 Experimental Results and Analysis

Experimental results show that the PR group achieves a 35.0% iteration rework rate, 30.0% constraint violation rate, and 100% manual dependency, with all iterations requiring manual intervention and defect correction. In contrast, the CP group reduces the rework rate to 5.0%, constraint violation rate to 0%, and manual dependency to 0%, realizing full-process automated closed-loop governance. The core reason is that traditional PR lacks pre-generation constraint control, leading to random AI generation behaviors and inherent specification violations and requirement deviations that rely entirely on manual remediation. The CP workflow eliminates violations and defects at the source via pre-solidified constraints, replacing subjective manual review with standardized evidence verification and intelligent decision-making,

significantly improving iteration efficiency and governance standardization.

7 Discussion and Threats to Validity

7.1 Innovation Positioning and Paradigm Boundary Definition

The core innovations of this paper are hierarchically defined with clear priorities and boundaries to avoid ambiguous innovation positioning: (1) Primary core innovation: proposing a novel intention-driven CP software change governance workflow, subverting the traditional artifact-driven PR post-governance paradigm and restructuring core governance timing and logic for AI-era code changes; (2) Secondary core innovation: constructing a complete formal theoretical system for CP workflows, including five-dimensional information modeling, state-transition modeling, and three core propositions verifying paradigm heterogeneity and non-evolvability; (3) Auxiliary tertiary innovation: prototype implementation and experimental validation of CP workflows based on the Zelos platform, verifying theoretical feasibility via a mature multi-agent AI platform rather than developing a new system from scratch. This positioning clarifies that the core contribution of this paper lies in workflow paradigm and theoretical model innovation, with system implementation serving only as validation, complying with top-tier software engineering conference standards.

This paper does not negate the engineering value of PR workflows. The two paradigms have complementary scenario boundaries. PR is suitable for low-frequency, high-precision, customized human-driven development with stable architecture and concise processes. CP focuses on AI-native, high-throughput, large-scale, unmanned iteration scenarios, resolving structural adaptation defects of traditional workflows. The two paradigms can coexist and be dynamically switched in industrial projects.

7.2 Structural Paradigm Value of CP Workflow

The paradigm iteration from PR to CP represents an inevitable upgrade of software change governance systems adapting to AI-driven R&D transformations, marking the governance logic transition from "human post-hoc error correction" to "AI pre-emptive active control". Traditional PR workflows are designed for human development bandwidth and low-frequency iteration rhythms, inherently suffering from critical path review bottlenecks, post-risk exposure, and insufficient traceability—all structural defects unresolvable via incremental optimization. The proposed CP workflow restructures workflow topology and governance core, taking change intentions and pre-defined constraints as primary driving factors to enable full-process standardized governance of code generation, verification, and decision-making. It fundamentally resolves the structural mismatch between AI high-speed production and manual governance bandwidth, providing core workflow architecture support for industrialized autonomous multi-agent software engineering.

7.3 Essential Differences Between Paradigm Replacement and Incremental Optimization

Combining prototype implementation and experimental results, this paper further clarifies that CP is a **structural replacement** for PR rather than incremental optimization. All industrial PR enhancement schemes, including AI-assisted review, pre-commit CI verification, and intelligent rule matching, retain PR's native serial topology of "generate → review → merge", artifact-driven core logic, and post-hoc governance timing. Such optimizations only improve manual review efficiency within the original paradigm and cannot

resolve critical path bottlenecks and AI adaptability issues.

In contrast, CP thoroughly restructures underlying workflow topology and governance logic with the core pipeline of "intention definition → constraint solidification → constrained generation → evidence verification → compliant merge". It shifts the governance core from code artifact verification to change intention control, actively constraining AI generation behaviors via pre-defined rules and replacing subjective manual review with standardized automated evidence verification, constituting a completely heterogeneous new governance paradigm with no feasible incremental evolution path from PR.

7.4 Threats to Validity

7.4.1 Scenario Boundary Threats

The core advantages of CP are reflected in AI code generation, large-scale software refactoring, multi-agent autonomous iteration, and high-frequency automated change scenarios. For high-precision, low-iteration scenarios such as kernel development, compiler development, and hardware-coupled embedded system development, CP pre-governance provides limited gains, while traditional PR workflows remain concise and efficient, indicating clear scenario boundaries and non-universal applicability.

7.4.2 Deployment Overhead Threats

CP deployment requires standardized construction of intention definition, architectural constraint, coding specification, and verification criterion systems, resulting in initial process adaptation and rule precipitation costs. For ultra-lightweight small-scale projects and temporary rapid iteration tasks, standardized CP governance introduces minor process redundancy compared with lightweight PR workflows.

7.4.3 Experimental Limitations & Honest Declaration on 0% Violation Rate

All experiments are conducted on the Zelos v0.9.0 platform with standardized business repositories under controllable and standardized task scenarios. The adaptability to ultra-large-scale industrial complex projects, multi-team collaborative iteration, and heterogeneous tech stack environments requires further verification, limiting the generalizability of experimental results.

Honest declaration on the 0% violation rate: The 0% constraint violation rate reported in our experiments was achieved under explicit, deterministic constraint rules (e.g., lint specifications, dependency whitelists, interface contract verification, vulnerability database matching). For constraints requiring deep semantic understanding (e.g., "design pattern appropriateness," "concurrency safety," "transaction boundary correctness"), the current CP verification capabilities are limited and cannot guarantee a 0% violation rate. This is not a fundamental flaw of the CP workflow paradigm but rather a boundary of current automated verification toolchains' semantic analysis capabilities. As AI code semantic understanding continues to evolve, this boundary will progressively shrink. Candidly acknowledging this current limitation is both a necessary demonstration of rigorous academic practice and a clear pointer for future research on more sophisticated semantic-level verification.

7.5 Future Optimization Directions

Targeting current deployment overhead and scenario limitations, future research will focus on three optimization directions: first, leveraging LLMs to automatically generate CP proposals, intention parsing, and constraint templates to reduce standardized construction costs and process redundancy; second, adapting to multi-tech-stack, ultra-large-scale industrial projects and multi-team collaborative scenarios to expand CP scenario universality; third, realizing native adaptation between CP and mainstream code hosting platforms and CI/CD toolchains to build a complete AI-native software change governance ecosystem and improve deployment convenience and universality.

8 Conclusion and Future Work

Large language models and AI agent technologies have thoroughly reshaped software development paradigms. Designed for manual development, the traditional PR code change governance paradigm suffers from structural defects including post-review topology, artifact-driven logic, inherent iteration throughput bottlenecks, insufficient risk control, and excessive manual reliance, making it incapable of supporting high-throughput, automated, large-scale AI-native development. Targeting this industrial pain point, this paper proposes a novel **intention-driven CP software change governance workflow**, constructing a complete closed-loop governance system with formal theoretical modeling and five-dimensional information specifications, thoroughly subverting traditional PR post-governance logic.

Through rigorous formal proof, this paper verifies that PR and CP are structurally heterogeneous paradigms, proving that traditional PR cannot adapt to AI-native development via any local incremental optimization and demonstrating the inevitability of software governance paradigm iteration. Based on the Zelos v0.9.0 multi-agent AI software engineering platform, this paper completes the engineering prototype implementation of the CP workflow with integrated governance architecture and full-process execution mechanisms. Three standardized comparative experiments quantitatively verify the significant advantages of CP in reducing rework rates, avoiding architectural risks, eliminating manual dependence, and supporting unmanned iteration, confirming that the proposed paradigm is not merely theoretical but possesses reliable practical engineering deployment capabilities.

Overall, the core contributions of this paper focus on paradigm innovation and theoretical modeling, supplemented by mature prototype implementation and quantitative experimental verification, forming a complete research closed loop of "theoretical definition → formal proof → system deployment → experimental validation". This work restructures the underlying workflow framework for code change governance in intelligent software engineering, providing a standardized governance paradigm and executable specification for AI-driven autonomous software development and industrialized multi-agent deployment.

Future work will focus on toolchain deployment, automatic CP specification generation, and intelligent constraint matching to further reduce deployment overhead and build a complete, deployable AI-native software change governance system.