

AI-Powered Code Review System

Multi-Dimensional Automated Quality Analysis for GitHub Pull Requests

Team: Project 1 | Software Engineering

Team Structure & Roles



Product Owner

Chunjin ZHU

Req. Engineering, MOSCOW
Prioritization, Scope
Management.



Scrum Master

Yi ZHUANG

Process Facilitation, Velocity
Tracking, CI/CD Pipeline.



Dev Team

Wenrui, Xian, Han, Yiran

Full-Stack Implementation
(Layers 1-5), Testing.

The Problem: Manual Review Bottlenecks



1. High Latency

Manual reviews take **1-2 hours** per PR. Waiting for feedback blocks the entire deployment pipeline.



2. Inconsistency

Humans miss hidden errors. Senior devs focus on logic, while juniors might miss style violations.



3. Limited Context

Standard Linters catch syntax errors but lack understanding of **Business Rules** (e.g., ``requirement.txt``).

Review of Related Works

Current solutions are fragmented. We aim to bridge the gap.

Category	Examples	Limitations (Our Motivation)
Static Analysis	SonarQube, ESLint	Rigid rules. No semantic understanding of intent. High maintenance overhead.
AI Tools	GitHub Copilot, DeepCode	Focus on generation/security. Lack project context (docs) and strict compliance checks.
Workflow Tools	GitHub Actions	Execution runners only. Produces fragmented outputs rather than a unified report.

Our Solution: 5-Layer Architecture

Unified Pipeline

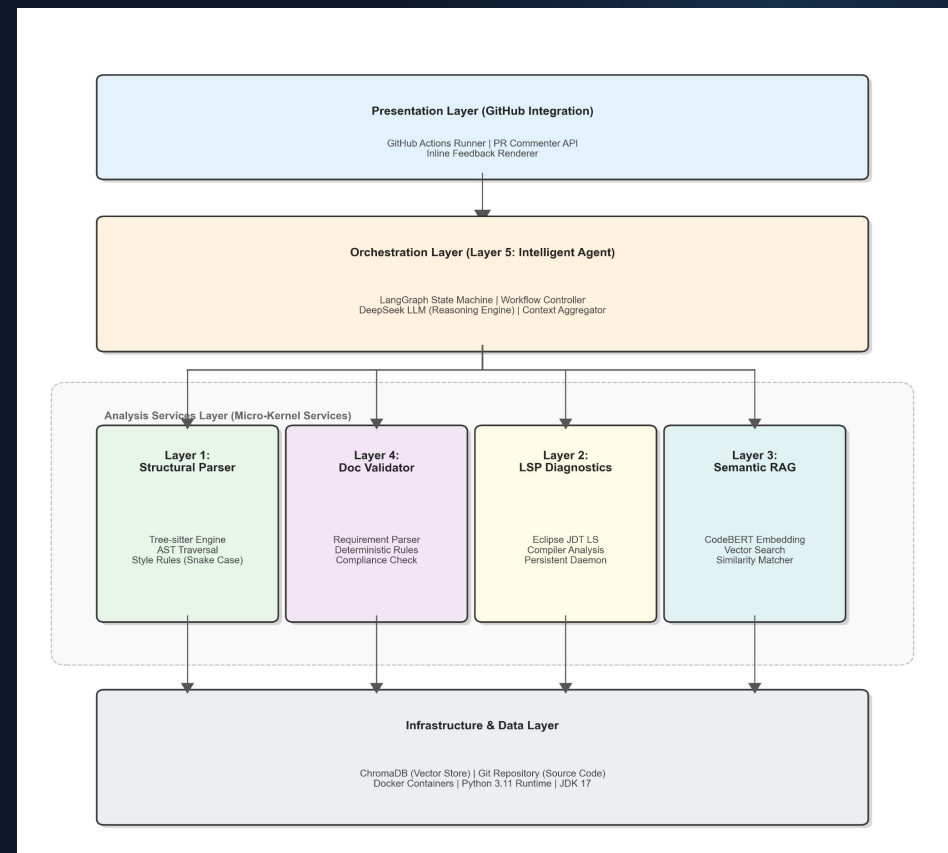
Layer 1: **Tree-sitter** (Fast Style Check)

Layer 2: **LSP (JDT)** (Compiler Diagnostics)

Layer 3: **CodeBERT** (Semantic RAG)

Layer 4: **Doc Validator** (Compliance)

Layer 5: **LLM Agent** (Orchestration)



Engineering Decision: ADD (Layer 1)

Using **Attribute-Driven Design** to select the parsing engine.

Goal: Low Latency & High Precision.

Criterion (Weight)	Tree-sitter (Selected)	ANTLR	Babel Parser
Functionality (0.2)	5	4	4
Performance (0.3)	5 (Incremental)	4	4
Ecosystem (0.2)	5	4	3
Scalability (0.3)	5 (Polyglot)	4	3
Weighted Score	5.0	4.0	3.5

Software Process: 12-Week Lifecycle

Sprint 1

Foundation & Requirements

Tree-sitter Setup, Docker Env

Sprint 2

Core & Crisis

LSP Integration, Latency Fix, Scope Cut

Sprint 3

Orchestration

LangGraph, DeepSeek, Testing

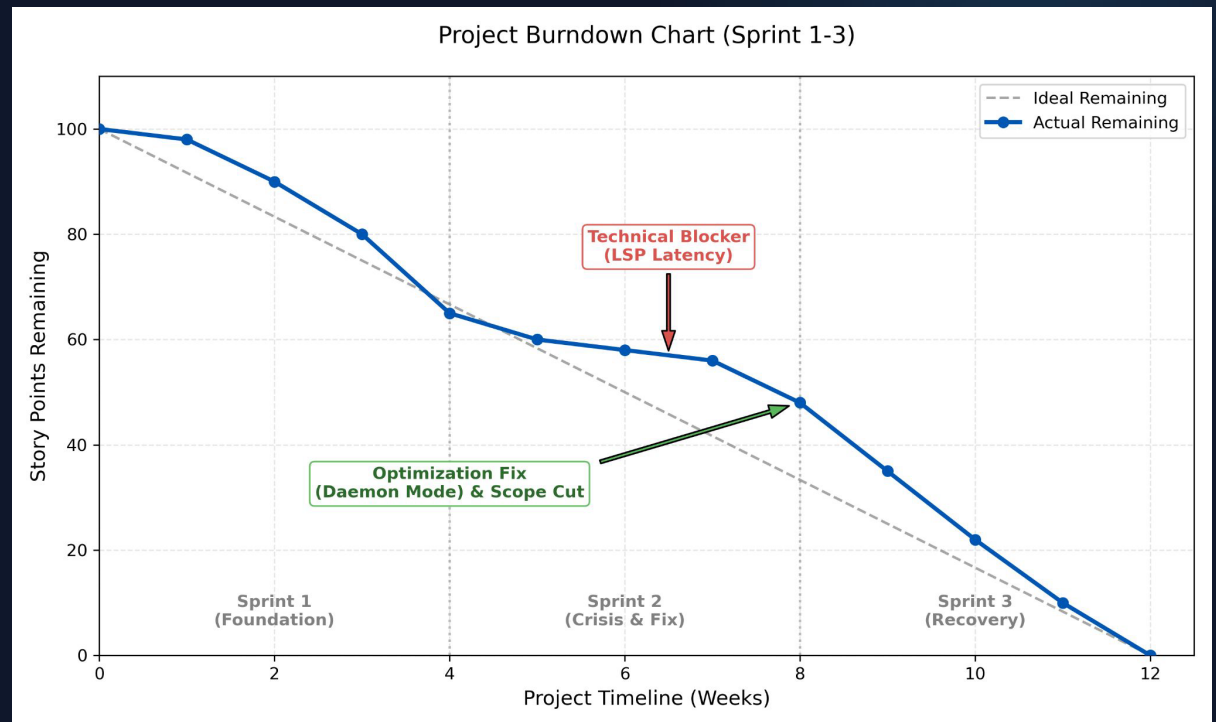
The "LSP Latency Crisis"

In Sprint 2, velocity **plateaued**.

The Issue: Initial JVM startup for LSP took **5s+ per file**, violating our performance NFR.

The Management Decision:

- Halt new features ("Swarm Mode").
- Prioritize Technical Debt resolution.



Resolving the Crisis & Recovery



Technical Solution

Implemented **Persistent Daemon Mode** using Python's `asyncio`.

Result: Latency dropped to < 500ms.



Managerial Solution

Scope Management: Formally dropped US-05 (GUI Config) to recover velocity.

Result: Delivered core features on time.

| Sprint 3: Orchestration

LangGraph Integration

Implemented the "Brain" of the system (Layer 5).

Created a state machine that orchestrates parallel execution of LSP (Layer 2) and RAG (Layer 3) to reduce latency.

Fail-Fast Mechanism

Implemented **Conditional Edges**.

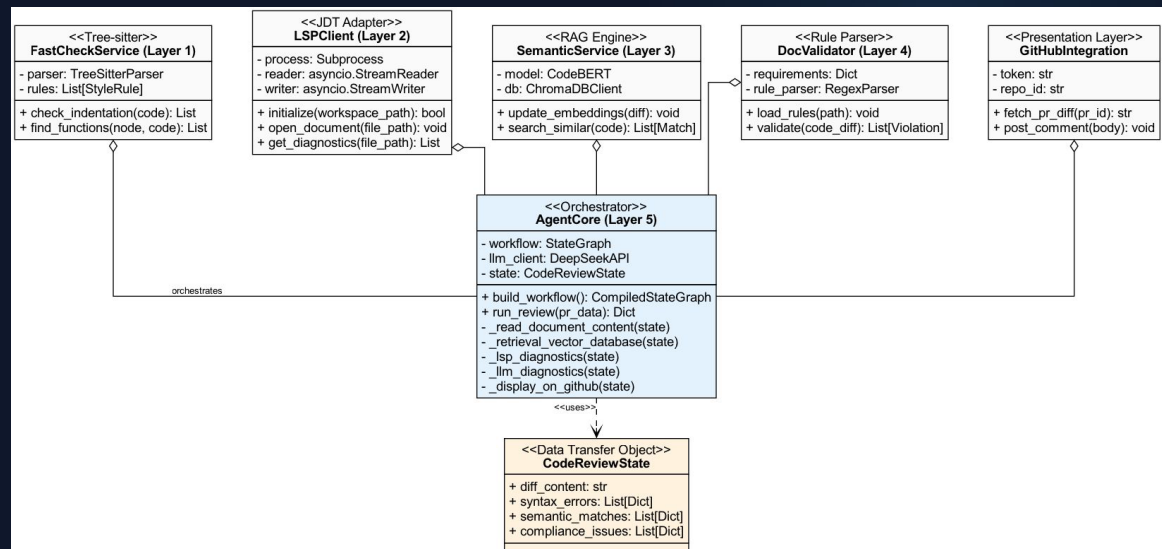
If Layer 1 (Fast Check) fails, the workflow terminates immediately. This saves expensive LLM tokens and computation time.

Object-Oriented Design

Modular Architecture

Strict adherence to SOLID principles.

The **AgentCore** acts as the orchestrator, decoupling specific analysis logic from the execution flow.



Empirical Evaluation: Scenarios

Validated the system using a Target Application with injected defects.

Test Scenario	Detection Layer	Result
Style: Tab indentation & Naming violation	Layer 1 (Tree-sitter)	✓ Blocked
Syntax: Unused variable & Type mismatch	Layer 2 (LSP)	✓ Detected
Business Logic: Age filter (18 vs 20-35)	Layer 4 (Doc Validator)	✓ Detected
Semantic: Duplicate logic suggestion	Layer 5 (DeepSeek)	✓ Suggested

*Data Source: Table XI (Requirement Validation Matrix)

Impact: 90% Efficiency Gain

Manual Review

120 Mins

Our System

12 Mins

90%

Time Reduction

25%

Higher Coverage

Conclusion

- ✓ **Automated Lifecycle:** From pre-commit to PR comments.
- ✓ **Process Driven:** Agile management enabled us to navigate the LSP crisis.
- ✓ **Validated:** Proven effectiveness through empirical scenarios.

Future Work

Expand to Multi-Language support (Python/JS) and add Self-Learning capabilities.

Q & A

Thank you for listening.

Project Repository: github.com/NpcRyanYao/CodeReview