# 140509\_46.md — AI-Enhanced Code Generation and Review Platform

**Theme:** AI in Software Engineering Lifecycle  
**Mission:** Boost developer productivity and code quality via context-aware NL→Code generation, automated tests, intelligent code review, and rich IDE/CI integrations.

## README (Problem Statement)

**Summary:** Build a comprehensive platform that assists developers with code generation from natural language, automated testing, and intelligent code review across the development lifecycle.  
**Problem Statement:** Deliver a platform that understands project context, integrates with IDEs and CI/CD, and suggests code aligned with standards while generating tests and performing high-signal reviews.

**Steps:**  
- Natural language to code with context awareness  
- Automated test generation & coverage analysis  
- Intelligent review (bug/risk detection, optimizations)  
- IDE plugins & workflow integrations  
- Documentation generation/maintenance  
- Quality metrics & technical debt assessment

**Suggested Data:** Large multi-language repos with tests and review comments; style guides; bug/patch histories.

## 1) Vision, Scope, KPIs

**Vision:** Make high-quality software the default by embedding AI across design-build-test-review stages.  
**Scope:**  
- v1: NL→Code, IDE plugin, static checks, unit test stubs, CI comments.  
- v2: ML bug detector, integration tests, doc generation, refactoring suggestions.  
- v3: Multi-repo context, architectural reviews, technical debt analytics.

**KPIs:**  
- Suggestion acceptance rate ≥ 50%  
- Auto-tests raise coverage ≥ 70% lines/branches on new code  
- Review engine catches ≥ 80% of seeded bug patterns  
- Dev cycle time ↓ 30% for target teams

## 2) Personas & User Stories

* **Developer:** Inline NL prompts → code; quick fix & refactor suggestions.
* **QA Engineer:** Auto-generated tests with reports & coverage gates.
* **Tech Lead:** Review dashboards, policy gates, debt trendlines.
* **Security Engineer:** SAST/secret scans with autofixes and PR annotations.

**Stories:**  
- US-01: Generate a typed API client from an OpenAPI spec.  
- US-06: Propose tests to cover edge cases identified by symbolic execution.  
- US-12: PR review auto-flags SQL injection and suggests a parameterized fix.

## 3) PRD (Capabilities)

1. **Context-Aware NL→Code:**
   * Retrieves relevant files/snippets, types, API usage, and project style; supports Python/TS/Go/Java/C# in v1.
2. **Test Generation & Coverage:**
   * Unit/integration test synthesis (property-based where applicable); coverage & mutation testing reports.
3. **Intelligent Review:**
   * Static + ML: bug risk scores, concurrency/safety checks, performance tips, security rules (CWE/OWASP).
4. **IDE/CI Integration:**
   * VS Code/JetBrains plugins; inline diffs; CI bots with PR annotations and auto-fix PRs.
5. **Documentation:**
   * Docstrings, READMEs, architecture digests, change logs.
6. **Quality Metrics:**
   * Lint/complexity/duplication, coverage, hotspot detection, debt scoring.

## 4) FRD (Functional Requirements)

* **RAG Context Server:** AST & symbol index, vector index over code/comments, dependency graph.
* **Prompt Builder:** Templates inject context (API signatures, tests failing, style rules).
* **Generators:** CodeGen, TestGen, DocGen with safety rails (no destructive shell, no secrets).
* **Review Pipeline:** SAST (Bandit/ESLint/semgrep), license checks, secret detection, ML classifiers for bug patterns.
* **CI Gates:** enforce min coverage, max complexity, security severities; staged approvals.
* **Policy Engine:** org/team rules (e.g., forbid eval/exec).
* **Telemetry:** acceptance, regressions, IDE latency; opt-in privacy.

## 5) NFRD

* **Latency:** P95 suggestion ≤ 300 ms (cached context); ≤ 1.5 s cold.
* **Scale:** Repos up to 10M LOC; multi-repo context.
* **Security:** On-prem isolation; no code leaves tenant; SBOM for components.
* **Reliability:** 99.9% plugin service uptime.
* **Compliance:** SOC2/ISO27001; code retention policies.

## 6) Architecture (Logical)

[IDE Plugin] → [Gateway] → [Context/RAG Server] → [Generators: Code|Test|Doc]  
 | |  
 v v  
 [Review Pipeline] [Metrics Store]  
 | |  
 v v  
 [CI Bot] [Dashboards]

## 7) HLD (Key Components)

* **Context Server:**
  + Build AST, symbol table, call graph; compute embeddings per symbol/file; elastic code search.
* **CodeGen:**
  + Large code LLM; decoding constrained by types & lints; temperature ≤ 0.2 by default.
* **TestGen:**
  + Path exploration (symbolic execution) + heuristics; property-based tests for pure functions.
* **Review Engine:**
  + Semgrep rules + ML risk model; taint analysis for sinks (SQL, SSRF, command).
* **DocGen:**
  + Generate docstrings from AST; summarize modules; Mermaid UML/sequence diagrams.
* **CI Bot:**
  + PR annotations, auto-fix patch generation, rollback/patch explainers.

## 8) LLD (Selected)

**Context Retrieval:** - Build query with current file, cursor scope, imported types; fetch top-k symbols from vector index; include failing tests and lint findings.

**Prompt Template (Python):**

System: You are a senior Python engineer.  
Context: <snippets+APIs+style+tests>  
Task: Implement function {name} satisfying docstring and tests.  
Constraints: PEP8, type hints, no external calls, raise ValueError on invalid input.

**Review Rule (Semgrep):**

rules:  
- id: py.sql.injection.param  
 pattern: cursor.execute($QUERY)  
 message: Use parameterized queries.  
 severity: ERROR

**CI Gate (Coverage):** - Fail PR if new/changed lines coverage < 70%.

## 9) Pseudocode (End-to-End)

on\_ide\_request(prompt, cursor):  
 ctx = retrieve\_context(repo, cursor)  
 code = codegen(prompt, ctx)  
 tests = testgen(code, ctx)  
 review = review\_engine(code, ctx)  
 docs = docgen(code, ctx)  
 return bundle(code, tests, review, docs)  
  
on\_ci\_pull\_request(pr):  
 metrics = run\_checks(pr)  
 if metrics.coverage < 0.7 or metrics.security.high > 0:  
 annotate(pr, metrics)  
 if can\_autofix(metrics): create\_autofix\_pr(pr)  
 else:  
 approve(pr)

## 10) Data & Evaluation

* **Training/Seeds:** BigCode/The Stack (filtered), CodeSearchNet, internal corpora with consent; review datasets (MSR, Google, GitHub PRs).
* **Metrics:** suggestion acceptance, edit distance to final, test coverage uplift, bug detection precision/recall, time-to-merge.
* **A/B:** team-level rollouts; guarded promotion via gates.

## 11) Security & Governance

* PII/secret scrubbing; local inference option; reproducible builds; signed models; audit logs.
* License compliance checks; third-party component SBOMs.

## 12) Observability & Cost

* Metrics: IDE latency, acceptance %, test gen time, CI queue times, GPU utilization.
* Cost controls: distillation, quantization, shared KV cache, batching; autoscale.

## 13) Roadmap

* **M1 (4w):** IDE plugin + NL→Code + static checks.
* **M2 (8w):** TestGen + CI bot + coverage gates.
* **M3 (12w):** ML bug model + DocGen + auto-fixes.
* **M4 (16w):** Multi-language scale + architectural reviews + debt analytics.

## 14) Risks & Mitigations

* **Hallucinated code:** retrieval augmentation, constrained decoding, unit-test-first mode.
* **False positives in review:** precision-tuned rules, allow suppressions, human-in-loop.
* **Latency spikes:** warm pools, KV cache, local models.
* **IP concerns:** on-prem sealed deployment, data minimization.