# 140509\_48.md — Legacy System Modernization Assistant

**Theme:** AI in Software Engineering Lifecycle, AI for Reengineering  
**Mission:** Analyze legacy systems to generate modernization strategies, refactor/transform code safely, and validate functional equivalence with automated tests and phased migration plans.

## README (Problem Statement)

**Summary:** Build an AI assistant that analyzes legacy codebases and provides recommendations for modernization, refactoring, and technology migration.  
**Problem Statement:** Organizations struggle to modernize legacy systems due to complexity, risk, and scarce knowledge. Create an assistant that understands legacy code (COBOL/PL-SQL/C/Java, etc.), maps dependencies, identifies modernization options, estimates risk/effort, generates transformed code, and validates behavior through automated testing.

**Steps:**  
- Legacy code analysis & dependency mapping  
- Strategy generation w.r.t. target stacks  
- Risk assessment for planning  
- Code transformation with business logic preservation  
- Test strategy generation  
- Project planning & resource estimation

**Suggested Data:** Legacy repos; migration case studies; modernization patterns (Strangler, microservices, event-driven); risk criteria; testing artifacts; DB schemas.

## 1) Vision, Scope, KPIs

**Vision:** Compress modernization timelines while de-risking rewrites through AI-guided analysis, refactoring, and verification.  
**Scope:**  
- v1: static analysis, call/data-flow graphs, modernization strategy, risk heatmaps.  
- v2: partial code transformation (module-level), test synthesis, DB migration scripts.  
- v3: end-to-end pipelines with canary releases, runtime shims, continuous equivalence testing.

**KPIs:**  
- Manual discovery effort ↓ 50%  
- Functional equivalence ≥ 95% on golden test suites  
- Auto-generated test coverage ≥ 80% of critical paths  
- PROD incident rate during migration < baseline by 30%

## 2) Personas & User Stories

* **Modernization Architect:** needs impact/risk analysis and target-state blueprint.
* **Legacy Engineer:** wants accurate dependency maps and side-effect awareness.
* **Platform Engineer:** needs deployment patterns and infra IaC.
* **QA Lead:** needs equivalence tests and regression nets.
* **Product Owner:** needs phased plan with cost/benefit and timelines.

**Stories:**  
- US‑01: Generate a system map (modules ↔ DB ↔ batch jobs ↔ external).  
- US‑07: Recommend refactor vs rewrite with rationale.  
- US‑12: Produce COBOL→Java/Spring sample with passing tests.  
- US‑15: Create phased migration Gantt with risk mitigation.

## 3) PRD (Capabilities)

1. **Code Intelligence:** parsers for COBOL, PL/SQL, C, Java, .NET; build AST, symbol table, call/dep graphs; detect patterns (batch, screen, file I/O).
2. **System Discovery:** runtime tracing option; map interfaces, data lineage, critical paths, SLAs.
3. **Strategy Engine:** target-state options (cloud-native, microservices, serverless, DDD); pros/cons & feasibility.
4. **Risk Assessment:** complexity, churn, coupling, business criticality, test gaps → risk score.
5. **Transformation:** rule-based + ML transpilation; idiomatic templates for target language/framework.
6. **DB & Schema Migration:** DDL diff, data type mapping, ETL/CDC pipelines.
7. **Test Synthesis:** unit/property/integration tests; golden master recording; contract tests for external deps.
8. **Planner:** effort estimation (COCOMO-like + historical priors), roadmap, staffing, canary plans.
9. **Safety Nets:** runtime adapter/shim, shadow traffic, kill switches.

## 4) FRD (Functional Requirements)

* **Parsers & Indexers:** ANTLR-based parsers; build AST; symbol resolution; type inference where needed.
* **Graph Builder:** call graph, dataflow, dependency graph (modules↔tables↔files↔jobs); export to Neo4j.
* **Pattern Detectors:** mainframe file I/O, COBOL COPYBOOK usage, PL/SQL cursors, transactional boundaries, transaction scripts.
* **Strategy Generator:** matches detected patterns to modernization patterns (Strangler Fig, Saga, CQRS, Event-sourcing).
* **Risk Model:** Risk = f(complexity, churn, coupling, criticality, test deficit, defect density).
* **Transformer:** AST→AST rules (e.g., cursor loop → ORM stream); LLM-assisted idiomatic code; human review gates.
* **Schema Migrator:** DDL translator, index strategy, CDC for cutover; data quality checks.
* **TestGen:** glean requirements from comments/specs; mine logs for realistic inputs; golden master snapshots.
* **Planner:** dependency-based slicing; milestone generator; cost/benefit and ROI.

## 5) NFRD

* **Scale:** 10M+ LOC; parallel parse ≥100k LOC/min/node.
* **Accuracy:** transformation pass rate ≥ 85% first-pass on selected patterns.
* **Reliability:** 99.9% service uptime.
* **Security:** on-prem option; code never leaves VPC; SBOMs of tools.
* **Compliance:** audit of transforms and approvals.

## 6) Architecture (Logical)

[Legacy SCM/Artifacts] -> [Parsers] -> [AST/IR Store] -> [Graph Builder]  
 | |  
 v v  
 [Pattern/Risk Engine] [System Map (Neo4j)]  
 |  
 v  
 [Strategy Generator]  
 |  
 v  
 [Transform Engine] -> [Modern Repo]  
 |  
 v  
 [Test Synthesis & Runner]  
 |  
 v  
 [Planner/Dashboards]

## 7) HLD (Key Components)

* **IR Store:** persisted AST/IR shards (columnar for queries).
* **System Mapper:** visual explorer (React + Cytoscape), overlays critical paths & risks.
* **Transform Engine:** hybrid (rules + LLM); compiles safety diffs, runs unit tests; emits PRs.
* **Golden Master Runner:** record/ replay against legacy to compare outputs (tolerances).
* **Planner:** critical path analysis; dependency-aware slicing for phased rollout; Gantt + RACI.

## 8) LLD (Selected)

**Risk Scoring:**  
- risk = sigmoid(a\*complexity + b\*coupling + c\*churn + d\*criticality + e\*test\_gap); tiers: Low <0.33, Med 0.33–0.66, High >0.66.

**Transformation Rule (COBOL READ loop → Java):**  
- Detect COPYBOOK record; map to POJO; READ ... AT END → while(reader.hasNext()); WRITE → repository .save().

**DB Migration:**  
- Type map (NUMBER(10) → BIGINT); date handling; seq→identity; triggers→app events.

**Golden Master:**  
- Capture I/O pairs for critical modules; assert equivalence with tolerance configs.

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

analyze(repo):  
 ast = parse\_all(repo)  
 graph = build\_graph(ast)  
 risks = score\_risks(graph)  
 strategy = recommend(graph, risks, targets)  
 plan = plan\_migration(strategy)  
 return {graph, risks, strategy, plan}  
  
transform(module):  
 rules = select\_rules(module)  
 code\_new = apply\_rules(module, rules)  
 tests = synthesize\_tests(module, code\_new)  
 assert equivalence(module, code\_new, tests)  
 create\_pr(code\_new, tests)

## 10) Data & Evaluation

* **Corpora:** open-source legacy code (Gov COBOL samples, OSS PL/SQL/C), internal anonymized systems.
* **Metrics:** transformation success %, test coverage uplift, defect escape rate, time-to-modernize.
* **Benchmarks:** run pilot on 3 representative subsystems; track rollback rate.

## 11) Security & Governance

* On-prem execution; no internet; signed toolchain; immutable logs of transforms; approvals required for merge.
* RBAC for architects, engineers, QA, and approvers.

## 12) Observability & Cost

* Metrics: LOC analyzed/day, % high-risk modules, PR acceptance, test pass rates.
* FinOps: parallelize analysis on spot nodes; cache IR; incremental re-analysis.

## 13) Roadmap

* **M1 (4w):** Parsers + graphs + risk heatmaps.
* **M2 (8w):** Strategy generator + pilot transforms.
* **M3 (12w):** Test synthesis + golden master + DB migration.
* **M4 (16w):** Full pipeline + phased rollouts + runtime shims.

## 14) Risks & Mitigations

* **Semantic drift:** strict golden masters; manual checkpoints.
* **Partial parser coverage:** incremental grammar expansion; fallbacks.
* **Operational risk:** strangler pattern; canary releases with kill switches.
* **Stakeholder resistance:** show ROI and phased wins.