Legacy Code Re-engineering: Agentic Pipeline Design Document

Executive Summary

This document outlines the design of an autonomous, multi-agent system for re-engineering legacy codebases into modern architectures. The system employs a three-stage pipeline: (1) Knowledge Assembly via ETL, (2) Intelligent Development with embedded validation, and (3) Automated CI/CD with continuous monitoring. The architecture emphasizes robustness through validation gates at each step, human oversight for critical decisions, and comprehensive testing against production data.

Primary Objective: Transform legacy systems (e.g., COBOL on Oracle DB) into modern implementations (e.g., Python/C++ combined with TypeScript/react) while preserving business logic, ensuring functional parity, and maintaining auditability.

System Architecture Overview

The system consists of three primary stages operating asynchronously:

Stage 0-1: Discovery & ETL (Knowledge Assembly)
Stage 2: Development Team (Architecture, Development, QA, Integration)
Stage 3: CI/CD & Operations (Build, Deploy, Monitor)

A central **Orchestrator Agent** serves as the event router and state store, coordinating work across all agents while preventing deadlocks and maintaining system-wide visibility.

Stage 0: Pre-Ingestion & Discovery

Discovery Agent

Purpose: Comprehensive asset inventory of the legacy system beyond just source code.

Input: Repository paths, URLs, file system access
Output: Complete asset inventory including source code, configuration files, database schemas, infrastructure-as-code (IaC), API contracts, and deployment scripts

Key Responsibilities:

- Scan all legacy system components
- Identify technology stack and dependencies
- Create initial metadata for downstream processing
- Detect non-code assets that contain business logic

Rationale: Legacy systems often embed critical business logic in configuration files, database stored procedures, and infrastructure definitions. Missing these assets would result in incomplete modernization.

Domain Expert Agent (SME Simulator)

Purpose: Address the common problem of sparse or missing documentation in legacy codebases.

Input: Code with minimal documentation, business domain context from available sources
Output: Inferred business logic, domain terminology mappings, contextual annotations

Key Responsibilities:

- Pattern recognition across the codebase to infer intent
- Business domain terminology extraction
- Gap-filling where documentation is absent
- Annotation of code with inferred business context

Rationale: Many legacy systems lack comprehensive documentation. This agent uses ML-based pattern recognition and domain inference to reconstruct the "why" behind the code, enabling better architectural decisions downstream.

Stage 1: ETL & Knowledge Assembly

Code Ingestion Agent

Purpose: Structured intake of all discovered assets with version tracking.

Input: Asset inventory from Discovery Agent
Output: File inventory, dependency graphs, entry point mappings, version tracking metadata

Key Responsibilities:

- Organize files by type, language, and purpose
- Build initial dependency graphs
- Track version information for delta processing
- Handle multi-language codebases

Enhancement: Supports **incremental updates** via delta detection, allowing the system to process new commits to the legacy repository without full re-ingestion.

Static Analysis Agent

Purpose: Deep structural and semantic analysis of all assets, including database schemas.

Input: File inventory, non-code assets (configs, schemas, IaC)
Output: Abstract Syntax Trees (AST), code structure graphs, Oracle DB schema mappings, API contracts, infrastructure topology, complexity metrics, coupling analysis, identified anti-patterns

Key Responsibilities:

- Parse source code into ASTs for structural understanding
- Extract database schema definitions and stored procedures
- Analyze configuration-driven logic
- Compute complexity metrics (cyclomatic, coupling, cohesion)
- Identify technical debt and anti-patterns

Rationale: Database schemas and stored procedures often contain significant business logic in legacy systems. Extracting and analyzing these alongside source code provides complete context.

Documentation Mining Agent

Purpose: Extract human-readable context and business intent from available documentation.

Input: Codebase files, configuration files, README documents, inline comments
Output: Business logic context, function intent descriptions, domain knowledge

Key Responsibilities:

- Parse inline comments and docstrings
- Extract business rules from documentation
- Map functions to business capabilities
- **Trigger**: Invokes Domain Expert Agent when documentation is sparse

Fallback Strategy: When documentation quality is insufficient, this agent escalates to the Domain Expert Agent for inference-based context generation.

Knowledge Synthesis Agent

Purpose: Consolidate all analysis outputs into a queryable, semantic knowledge base.

Input: AST data, business context, Oracle DB schema context, non-functional requirements (NFRs)
Output: Code embeddings, knowledge graph, semantic tags, component functionality map, NFR metadata, cross-cutting concern mappings (logging, security, compliance)

Key Responsibilities:

- Generate vector embeddings for semantic search
- Build knowledge graphs connecting components, dependencies, and business logic
- Tag components by functionality, complexity, and business domain
- Map cross-cutting concerns (observability, security models, compliance requirements)
- Store NFR metadata (latency requirements, throughput constraints)

Rationale: The knowledge graph enables intelligent querying by downstream agents, ensuring they have full context when making architectural or implementation decisions.

Delta Monitoring Agent

Purpose: Handle version drift in the legacy system during modernization.

Input: Continuous monitoring of legacy repository commits
Output: Change events, updated embeddings for Vector DB

Key Responsibilities:

- Detect new commits to legacy codebase
- Trigger incremental ETL updates
- Update Vector DB with delta changes
- Notify Orchestrator of significant changes

Rationale: Legacy systems rarely freeze during modernization efforts. This agent ensures the knowledge base remains current, preventing the modernized system from diverging from the source of truth.

Enhanced Vector DB

Purpose: Central knowledge store for all legacy system information.

Contents:

- Code embeddings for semantic search
- Dependency graphs and component relationships
- Oracle DB schema context and stored procedure logic
- Business logic mappings and domain knowledge
- NFR metadata (performance, scalability requirements)
- Cross-cutting concern mappings (security, compliance, observability)
- Version history and change tracking

Query Interface: All downstream agents query this database for context-aware decision-making.

Stage 2: Development Team Layer

Orchestrator Agent (Central Event Router)

Purpose: Coordinate all agent activities, manage state, prevent deadlocks, and route work intelligently.

Input: Vector DB query results, component dependency maps, complexity metrics, feedback events, escalation requests
Output: Prioritized task tickets, routing events, state updates, escalation decisions, telemetry events

Key Responsibilities:

- **Hierarchical Task Assignment**: Implements infrastructure → backend → integration → UI sequencing to build features incrementally
- **Deadlock Detection**: Monitors validator rejection loops and triggers escalation after 3 consecutive rejections
- **Dependency Management**: Ensures tasks are assigned in correct order based on component dependencies
- **State Management**: Tracks progress of all tickets across the pipeline
- **Event Routing**: Directs completed work to appropriate next agents

Logic Enhancements:

- Prioritization avoids bias toward "easy" fixes by using hierarchical feature building
- Incremental feature construction ensures foundational components are modernized first
- Telemetry integration enables human oversight and audit trail

Additional orchestration functions include:

- **Telemetry & Audit Agent**
  Collects action logs, validator outcomes, and agent decisions into a compliance-grade audit trail.
  Provides full traceability for governance and regulatory requirements.
- **Escalation Agent**
  Monitors rejection loops across validators.
    - If rejections exceed 3 iterations, escalation is triggered.
    - Provides automated resolution strategies or routes issues to Human Oversight.
- **Human Oversight Interface**
  Provides a channel for policy overrides, approvals, and high-stakes interventions.
  Acts as a safeguard for compliance, security, and accountability.

Rationale: A single orchestrator prevents coordination failures in asynchronous execution while maintaining system-wide visibility and control.

Telemetry & Audit Agent

Purpose: Ensure traceability, auditability, and enable human oversight.

Input: All agent actions, decision logs, state changes
Output: Audit trail, traceability reports, human oversight alerts

Key Responsibilities:

- Log every agent decision and action
- Maintain traceability from legacy code to new implementation
- Generate compliance reports
- Alert humans for critical decisions requiring approval
- Support post-mortem analysis

Rationale: Addresses the "ethics & alignment" concern by ensuring humans can review and override AI decisions, and provides audit trails for compliance.

Escalation & Resolution Agent

Purpose: Break deadlocks and handle persistent validation failures.

Input: Validator rejection loops, deadlock detection signals
Output: Resolution strategies, human intervention requests, priority overrides

Key Responsibilities:

- Detect when validators repeatedly reject work (>3 iterations)
- Analyze root causes of rejection loops
- Determine if issue requires human intervention
- Propose alternative approaches or architectural pivots
- Override priorities when necessary

Rationale: Prevents infinite loops and ensures forward progress even when agents disagree.

Technical Architect Agent

Purpose: Design modern system architecture that satisfies functional and non-functional requirements.

Input: Task ticket, legacy component context, Oracle DB schema, NFR requirements, cross-cutting concerns, security & compliance models
Output: Architecture document, API specifications, data models, security architecture, observability design, compliance checklist

Key Responsibilities:

- Design system architecture for assigned components
- Define technology stack and design patterns
- Specify API contracts and data models

- Address **cross-cutting concerns**: logging, observability, security, compliance
- Ensure **NFR coverage**: latency, throughput, maintainability, scalability
- Create security architecture and authentication/authorization models

Enhancement: Explicitly addresses non-functional requirements and cross-cutting concerns, which are often overlooked in legacy modernization.

Rationale: A well-defined architecture prevents downstream rework and ensures the modernized system meets operational requirements.

Architecture Validator Agent

Purpose: Ensure architectural designs are complete, feasible, and compliant.

Input: Architecture document, NFR requirements, legacy component requirements
Output: Validation result with feedback, approval/rejection event

Validation Criteria:

- Completeness: All requirements addressed
- NFR coverage: Latency, throughput, maintainability specified
- Cross-cutting concerns: Logging, security, compliance included
- Feasibility: Implementation is realistic given constraints
- Consistency: Aligns with overall system design

Escalation Policy: After 3 rejection loops, escalates to Orchestrator for resolution.

Rationale: Prevents incomplete or infeasible designs from reaching implementation, reducing costly downstream corrections.

Developer Agent

Purpose: Implement code according to architectural specifications.

Input: Architecture specification, task ticket (new code OR refactor), legacy implementation

details, Oracle DB schema access, business logic context
Output: New code or refactored code, unit tests, migration scripts (if needed)

Key Responsibilities:

- Write modern, clean code following architectural specifications
- Query Vector DB for legacy implementation context
- Handle both **new code creation** and **refactoring tasks**
- Generate unit tests with high coverage
- Create data migration scripts for schema changes
- Support multiple output languages (Python, C++, etc.)

Enhancement: Explicitly handles refactoring in addition to new code, recognizing that legacy modernization involves both.

Rationale: The agent has full context from Vector DB to ensure business logic preservation while writing idiomatic modern code.

Code Validator Agent

Purpose: Verify code correctness and specification adherence.

Input: Code, unit tests, architecture specification, legacy code patterns
Output: Validation result with issues list, approval/rejection event

Validation Criteria:

- Specification adherence
- Functional correctness
- Unit test quality and coverage
- Error handling and edge cases
- Security vulnerabilities (basic checks)

Rationale: Catches functional errors before code reaches QA, accelerating feedback loops.

Quality Attribute Agent

Purpose: Enforce code quality standards, style, and maintainability.

Input: Approved code from Code Validator, style guides, coding conventions
Output: Quality report, approval/rejection event

Validation Criteria:

- Code style and formatting consistency
- Naming conventions
- Maintainability metrics (complexity, duplication)
- Code smells detection
- Documentation quality

Enhancement: Addresses the gap in style and convention validation missing from the original design.

Rationale: Ensures the modernized codebase is maintainable and consistent, reducing long-term technical debt.

Build Agent

Purpose: Compile, build, and package the code with automated checks.

Input: Merge event, source code
Output: Build artifacts, test results, security scan reports, code quality metrics

Key Responsibilities:

- Compile/build the codebase
- Run full automated test suite
- Execute security scans (SAST, dependency checks)
- Generate code quality metrics
- Package deployable artifacts

Rationale: Automated build validation ensures code is deployable before reaching QA.

Build Validator Agent

Purpose: Verify build integrity and quality gates.

Input: **Input**: Build artifacts, test reports, security scan results, **SBOM, supply chain security report**
Output: Validation result, approval/rejection event

Validation Criteria:

- Build success and artifact integrity
- All automated tests passing
- No critical security vulnerabilities
- Code quality metrics meet thresholds
- Dependency resolution successful
- Supply chain security requirements met (no critical vulnerabilities in dependencies)

Rationale: Prevents broken or insecure builds from advancing to QA or deployment.

QA Agent

Purpose: Comprehensive testing to ensure functional parity with legacy system.

Input: Validated code, architecture specifications, legacy behavior patterns, edge cases, anonymized production Oracle DB snapshots
Output: Test cases, test results, regression report, coverage metrics

Key Responsibilities:

- **Functional testing**: Validate new code behavior
- **Regression testing**: Compare new vs. legacy system outputs
- **Data-driven testing**: Test against anonymized production data snapshots from Oracle DB
- **Fuzz/chaos testing**: Resilience and edge case validation
- **Security testing**: Authentication, authorization, penetration test basics

- Generate comprehensive test suites from legacy behavior

Enhancement: Includes fuzz testing, chaos engineering, security testing, and data-driven testing against production datasets.

Rationale: Production data testing catches edge cases that synthetic data misses, ensuring true functional parity.

QA Validator Agent

Purpose: Verify testing completeness and quality.

Input: Test results, expected legacy behavior, coverage thresholds
Output: Validation result, approval/rejection event

Validation Criteria:

- Test coverage meets thresholds
- Edge cases adequately covered
- Security testing performed
- Resilience testing included
- Regression tests compare correctly with legacy

Rationale: Ensures QA testing is thorough and doesn't overlook critical scenarios.

Integration Validator Agent

Purpose: Final end-to-end validation before deployment, including testing against production data.

Input: QA results, all artifacts, legacy component interactions, Oracle DB schema & production data
Output: Integration report, final approval/rejection, multi-service coordination status

Key Responsibilities:

- **End-to-end functionality verification**
- **Business logic preservation**: Run new code against anonymized Oracle DB with production data
- Compare outputs with legacy system on real datasets
- Validate cross-component integration
- Test data migration correctness
- Performance baseline comparison (new vs. legacy)
- Coordinate with Multi-Service Coordinator for parallel modernization

Critical Enhancement: Executes new code against the anonymized production Oracle DB to ensure complete functional coverage and business logic preservation.

Rationale: This is the final gate before deployment. Running against production data provides highest confidence in correctness.

Multi-Service Coordinaton

Purpose: Synchronize parallel component modernization to prevent bottlenecks.

Input: Parallel component completion events
Output: Synchronized integration plan, dependency resolution, coordinated deployment strategy

Key Responsibilities:

- Track multiple components being modernized simultaneously
- Resolve inter-component dependencies
- Coordinate integration testing across components
- Prevent integration bottlenecks
- Plan synchronized deployments

Enhancement: Addresses the concern about multi-service orchestration and integration validator bottlenecks.

Rationale: Large modernization efforts require parallelization. This coordinator prevents the Integration Validator from becoming a serial bottleneck.

Stage 3: CI/CD & Operations

Deployment Agent

Purpose: Manage deployments with advanced strategies and state management.

Input: Approved artifacts, deployment configuration, rollout strategy
Output: Deployment status, environment state, migration status

Key Responsibilities:

- Execute deployments to staging/production
- Support **multiple rollout strategies**:
  - Blue-green deployments
  - Canary releases
  - Progressive rollouts
- Handle **state migrations**: database schema changes, data transformations
- Execute rollback procedures with **state reversion**
- Coordinate infrastructure changes

Enhancement: Supports gradual rollout strategies and handles complex state migrations, addressing the "binary deployment" concern.

Rationale: Modern deployments require sophisticated strategies to minimize risk. State migration handling is critical for database-dependent applications.

Deployment Validator Agent

Purpose: Verify deployment success and system health.

Input: Deployment status, smoke test results, canary metrics (if applicable)
Output: Health check results, approval/rollback event, canary promotion decision

Validation Criteria:

- Deployment completed successfully
- All services healthy and responsive
- Smoke tests passing
- No critical errors in logs
- Progressive rollout metrics acceptable (if canary)
- State migration integrity verified

Key Decisions:

- Approve full production deployment
- Promote canary to wider audience
- Trigger rollback with state reversion

Rationale: Automated validation prevents failed deployments from impacting users and enables safe progressive rollouts.

Monitoring Agent

Purpose: Continuous production monitoring with legacy baseline comparison.

Input: Production metrics, legacy system baseline data
Output: Anomaly events, performance reports, behavior drift alerts

Key Responsibilities:

- Track performance metrics (latency, throughput, error rates)
- Compare new system behavior against legacy baseline
- Detect anomalies and behavior drift
- Monitor resource utilization
- Generate alerts for regressions

Rationale: Ensures the modernized system maintains or improves upon legacy system performance and reliability.

Root Cause Analysis Agent

Purpose: Determine why anomalies occur, not just that they occurred.

Input: Anomaly events, deployment history, code changes, system logs
Output: Root cause report, remediation suggestions, postmortem documentation

Key Responsibilities:

- Correlate anomalies with recent deployments or code changes
- Analyze logs and metrics to identify root causes
- Suggest remediation strategies
- Generate postmortem documentation for learning
- Feed insights back to Orchestrator for corrective action

Enhancement: Addresses the gap in root cause attribution from monitoring events.

Rationale: Automated root cause analysis accelerates incident resolution and enables continuous improvement through learning.

Supply Chain Security Agent

Purpose: Ensure integrity and security of dependencies, build artifacts, and the CI/CD pipeline itself.

Input: Build artifacts, dependency manifests (package.json, requirements.txt, pom.xml, etc.), build logs, pipeline configuration
Output: Software Bill of Materials (SBOM), dependency security report, artifact signatures, provenance attestation, license compliance report

Key Responsibilities:

- **SBOM Generation**: Create comprehensive Software Bill of Materials listing all dependencies and their versions
- **Dependency Provenance**: Verify dependencies come from trusted, authenticated sources
- **Vulnerability Scanning**: Check dependencies against known vulnerability databases

(CVE, GitHub Advisory, etc.)
- **Supply Chain Attack Detection**: Identify typosquatting, dependency confusion, and compromised packages
- **Artifact Signing**: Cryptographically sign build artifacts for chain of custody
- **Pipeline Integrity**: Validate CI/CD pipeline configuration hasn't been tampered with
- **License Compliance**: Ensure all dependencies meet license requirements
- **Anomaly Detection**: Flag unusual build behaviors (unexpected network calls, file access patterns)

Enhancement: Addresses the critical gap in supply chain security, protecting against increasingly common attack vectors like compromised dependencies and malicious packages.

Integration Points:

- Receives build artifacts from Build Agent
- Validates before Build Validator performs quality checks
- Blocks deployment if supply chain risks detected
- Provides SBOM to monitoring systems for runtime correlation

Rationale: Modern supply chain attacks (SolarWinds, Log4Shell, npm package compromises) demonstrate that secure code isn't enough—the entire dependency chain and build process must be trustworthy. This agent provides defense-in-depth by validating that nothing malicious entered the system through dependencies or the build pipeline itself.

Detection Capabilities:

- Known vulnerable dependency versions
- Dependencies from untrusted or anomalous repositories
- Recently published packages with suspicious patterns (supply chain insertion attempts)
- License violations that could create legal risk
- Build process deviations from established baseline
- Unsigned or improperly signed artifacts

Human Oversight Interface

Purpose: Enable human review of critical decisions and prevent over-automation.

Input: Critical decisions, escalation requests, audit trails
Output: Human approvals, manual interventions, policy adjustments

Key Responsibilities:

- Present critical decisions for human review
- Enable manual intervention when agents disagree
- Allow policy adjustments based on human judgment
- Provide visibility into all agent actions
- Support compliance and governance requirements

Enhancement: Addresses ethics, alignment, and over-automation concerns.

Rationale: AI agents should augment, not replace, human judgment on critical architectural and business decisions.

Key Design Principles

1. Validation Gates at Every Step

Every agent output is validated before proceeding, creating multiple quality checkpoints and preventing defects from propagating downstream.

2. Asynchronous Execution with Central Coordination

Agents operate asynchronously for parallelization, but the Orchestrator maintains system-wide coordination and prevents deadlocks.

3. Context-Aware Decision Making

All agents query the Vector DB for legacy system context, ensuring decisions preserve business logic and functional requirements.

4. Human-in-the-Loop

Critical decisions require human approval, and all actions are auditable, preventing over-automation and maintaining alignment.

5. Comprehensive Testing

Testing spans functional, regression, security, performance, and data-driven testing against production datasets, ensuring true functional parity.

6. Incremental Modernization

The hierarchical task assignment (Infrastructure → Backend → Integration → UI) enables incremental, low-risk modernization with continuous validation.

7. Production Data Validation

Integration testing against anonymized production Oracle DB data provides the highest confidence in business logic preservation.

Risk Mitigation Strategies

Risk	Mitigation
Incomplete legacy understanding	Domain Expert Agent + comprehensive ETL
Version drift during modernization	Delta Monitoring Agent with continuous updates
Validation deadlocks	Escalation Agent with >3 rejection limit
Missing non-code assets	Discovery Agent captures configs, schemas, IaC
Inadequate testing	Multi-layered testing with production data validation
Failed deployments	Gradual rollout strategies + automated health checks

Risk	Mitigation
Over-automation	Human Oversight Interface for critical decisions
Lack of auditability	Telemetry & Audit Agent tracking all actions
Multi-service coordination	Multi-Service Coordinator preventing bottlenecks
Behavior regressions	Monitoring Agent with legacy baseline comparison

---

Success Metrics

- **Functional Parity**: 100% of legacy business logic preserved (validated via production data testing)
- **Quality Gates**: <5% rejection rate at each validation stage after initial stabilization
- **Performance**: New system meets or exceeds legacy system baseline
- **Time to Production**: Measured from ticket creation to deployment
- **Human Intervention Rate**: <10% of tickets require human escalation
- **Audit Compliance**: 100% traceability from legacy code to new implementation
- **Deployment Success Rate**: >95% deployments succeed without rollback

---

Conclusion

This agentic pipeline architecture provides a comprehensive, production-ready approach to legacy code modernization. By combining autonomous agents with validation gates, human oversight, and comprehensive testing against production data, the system balances automation efficiency with correctness guarantees. The architecture is designed for large-scale, complex modernization efforts while maintaining auditability, quality, and alignment with business objectives.