AI SDLC — Implementation Requirements

Version: 3.2.0 **Date**: 2026-02-20 **Derived From**: AI_SDLC_ASSET_GRAPH_MODEL.md (v2.1.0) **Parent Theory**: Constraint-Emergence Ontology

---

Purpose

Platform-agnostic implementation requirements for tooling that delivers the AI SDLC Asset Graph Model. Defines WHAT, not HOW. Concept numbers (#N) reference the ontology.

Audience: Implementers building AI SDLC tooling for any platform.

Requirement Format

`REQ-{CATEGORY}-{SEQ}[.MAJOR.MINOR.PATCH]`

Categories: `INTENT`, `GRAPH`, `ITER`, `EVAL`, `CTX`, `FEAT`, `LIFE`, `EDGE`, `TOOL`

Version semantics: PATCH (clarification), MINOR (criteria change), MAJOR (breaking). No version suffix = current.

Document Structure

1. Intent & Spec — Entry point: what to build
2. Asset Graph — Topology: admissible transitions
3. Iteration Engine — The only operation
4. Evaluators — Convergence criteria
5. Context — Constraint surface
6. Feature Vectors & Traceability — Trajectories through the graph
7. Full Lifecycle — CI/CD, telemetry, homeostasis
8. Edge Parameterisations — TDD, BDD, ADRs per edge type
9. Tooling — Plugins, workspace, commands

---

1. Intent & Spec

REQ-INTENT-001: Intent Capture

Priority: Critical | **Phase**: 1

The system shall capture intents (desires for change) in a structured format that enters the asset graph.

Acceptance Criteria: - Intents from: human input, runtime feedback, ecosystem changes - Unique identifier (INT-*) - Fields: description, source, timestamp, priority - Persisted, version-controlled

Traces To: Asset Graph Model §6.2 (Intent Lineage) | Ontology #36 (delta/intent)

REQ-INTENT-002: Intent as Spec

Priority: High | **Phase**: 1

Intents shall compose with Context[] to form the Spec — the encoded representation (#40) that drives construction.

Acceptance Criteria: - Intent + Context[] = Spec for a given graph traversal - Spec is the fitness landscape against which evaluators measure convergence - Spec evolves as intent lineage accumulates through traversal

Traces To: Asset Graph Model §3.1 (Iteration Function signature) | Ontology #40 (encoded representation)

REQ-INTENT-003: Eco-Intent Generation

Priority: Medium | **Phase**: 2

The system shall generate intents automatically when ecosystem changes are detected.

Acceptance Criteria: - Monitor for: security vulnerabilities, deprecations, API changes, compliance updates - Generate INT-ECO-* intents with ecosystem context - Feed into graph as new feature vectors

Traces To: Asset Graph Model §7.3 (Complete Cycle) | Ontology #44 (deviation signal)

REQ-INTENT-004: Spec Reproducibility

Priority: High | **Phase**: 1

The Spec (Intent + Context[]) shall be canonically serialisable, versioned, and hashable for audit reproducibility.

Acceptance Criteria: - Spec has a deterministic canonical serialisation (same inputs → same byte sequence) - Each Spec instance is content-addressable (hash of canonical form) - Spec hash recorded at each iteration and at convergence - Independent tools given the same Intent + Context[] shall compute the same Spec hash - Spec versions are immutable — evolution produces new versions, not mutations

Traces To: Asset Graph Model §3.1 (Iteration Function — Spec as input), §5.1 (Context[] contents) | Ontology #40 (encoded representation), #7 (Markov object — stable boundary)

---

2. Asset Graph

REQ-GRAPH-001: Asset Type Registry

Priority: Critical | **Phase**: 1

The system shall maintain a registry of asset types with typed interfaces.

Acceptance Criteria: - Default types: Intent, Requirements, Design, Code, Unit Tests, Test Cases, UAT Tests, CI/CD, Running System, Telemetry - Each type has: schema/interface definition, Markov criteria (#7) - Registry extensible — new types addable without engine changes

Traces To: Asset Graph Model §2.1 (Asset Types) | Ontology #7 (Markov object), #8 (Markov blanket)

REQ-GRAPH-002: Admissible Transitions

Priority: Critical | **Phase**: 1

The system shall define and enforce admissible transitions between asset types.

Acceptance Criteria: - Transitions are directed edges with source type and target type - Graph is cyclic (feedback edges are first-class) - Only admissible transitions can be traversed - Transition registry extensible — new edges addable - Transitions logged for audit

Traces To: Asset Graph Model §2.2 (Graph Properties) | Ontology #5 (admissible transformation), #9 (constraint manifold)

REQ-GRAPH-003: Asset as Markov Object

Priority: High | **Phase**: 1

An asset shall achieve Markov object status only when it satisfies all evaluators for its type.

Acceptance Criteria: - Boundary: typed interface/schema present and valid - Conditional independence: usable without construction history - Stability: all evaluators report convergence - Non-converged assets are candidates, not Markov objects

Traces To: Asset Graph Model §2.3 (Asset as Markov Object) | Ontology #7 (Markov object)

---

3. Iteration Engine

REQ-ITER-001: Universal Iteration Function

Priority: Critical | **Phase**: 1

The system shall implement a single iteration function for all graph edge traversals.

Acceptance Criteria: - Signature: `iterate(Asset<Tn>, Context[], Evaluators(edge_type)) → Asset<Tn.k+1>` - The asset carries intent, lineage, and full history — these are not separate parameters - Same function for all edges; behaviour parameterised by evaluators and context - Constructor (#41) implementations are edge-specific (LLM agent, human, compiler, test runner) - Iteration repeats until `stable()` reports convergence

Traces To: Asset Graph Model §3.1 (Signature) | Ontology #15 (local preorder traversal), #41 (constructor)

REQ-ITER-002: Convergence and Promotion

Priority: Critical | **Phase**: 1

The system shall promote candidates to the next asset type only upon evaluator convergence.

Acceptance Criteria: - `stable(candidate, edge_type) = ∀ evaluator ∈ evaluators(edge_type): evaluator.delta(candidate, Spec) < ε` - Convergence threshold (ε) configurable per evaluator per edge - Promotion: `ATn.m` becomes `ATn+1.0` - Non-convergence after max iterations raises to human evaluator

Traces To: Asset Graph Model §3.3 (Convergence) | Ontology #7 (stability condition)

---

4. Evaluators

REQ-EVAL-001: Three Evaluator Types

Priority: Critical | **Phase**: 1

The system shall support three evaluator types, composable per graph edge.

Acceptance Criteria: - **Human**: judgment, approval/rejection, domain evaluation - **Agent(intent, context)**: LLM-based gap analysis, coherence checking, refinement — probabilistic compute (#45) - **Deterministic Tests**: pass/fail — type checks, schema validation, test suites, SLA monitors — deterministic compute (#45) - All three compute delta (#36) between current state and target state

Traces To: Asset Graph Model §4.1 (Three Evaluator Types) | Ontology #35 (evaluator-as-prompter), #45 (two compute regimes)

REQ-EVAL-002: Evaluator Composition Per Edge

Priority: High | **Phase**: 1

Each graph edge shall declare its evaluator composition.

Acceptance Criteria: - Edge type → set of evaluators constituting `stable()` - Composition configurable (not hardcoded) - Default compositions provided (per Asset Graph Model §4.2 table) - Override at project, team, or organisation level

Traces To: Asset Graph Model §4.2 (Evaluator Composition Per Edge)

REQ-EVAL-003: Human Accountability

Priority: Critical | **Phase**: 1

Humans shall remain accountable for all decisions. AI evaluators assist; humans decide.

Acceptance Criteria: - AI (Agent evaluator) suggestions require human review before acceptance at edges where Human evaluator is configured - Decisions attributed to humans, not AI - Override capability always available

Traces To: Asset Graph Model §4.1 (Human evaluator type)

---

5. Context

REQ-CTX-001: Context as Constraint Surface

Priority: High | **Phase**: 1

The system shall manage Context[] as the standing constraint surface bounding construction.

Acceptance Criteria: - Context types include (open-ended): User Disambiguations, ADRs, Data Models, Templates, Prior Implementations, Policy - Each context element narrows the space of admissible constructions (constraint density #16) - Context shared across graph edges; subset relevance per edge configurable - Context version-controlled

Traces To: Asset Graph Model §5.1 (What Context Contains), §5.2 (Context as Constraint Density) | Ontology #16 (constraint density), #9 (constraint manifold)

REQ-CTX-002: Context Hierarchy

Priority: Medium | **Phase**: 2

The system shall support hierarchical context composition.

Acceptance Criteria: - Levels: global → organisation → team → project - Later contexts override earlier contexts - Deep merge for objects - Customisation without forking

Traces To: Asset Graph Model §5.3 (Context Stability) | Ontology #23 (scale-dependent time)

---

6. Feature Vectors & Traceability

REQ-FEAT-001: Feature Vector Trajectories

Priority: Critical | **Phase**: 1

Features shall be tracked as trajectories through the asset graph, identified by REQ keys.

Acceptance Criteria: - REQ key format: `REQ-{TYPE}-{DOMAIN}-{SEQ}` (the trajectory identifier — stable and immutable) - Requirement version: `REQ-{TYPE}-{DOMAIN}-{SEQ}.MAJOR.MINOR.PATCH` (the requirement statement — versioned per §Requirement Format) - The key (without version suffix) is the immutable identifier; the versioned form tracks how the requirement statement evolves - Types: F (functional), NFR (non-functional), DATA (data quality), BR (business rule) - Keys propagate through all assets on the trajectory - Bidirectional navigation: forward (intent → runtime) and backward (runtime → intent)

Traces To: Asset Graph Model §6.1 (Feature as Vector), §6.2 (Intent Lineage) | Ontology #2 (constraint propagation)

REQ-FEAT-002: Feature Dependencies

Priority: High | **Phase**: 1

The system shall track cross-vector dependencies between features.

Acceptance Criteria: - Feature B can depend on Feature A's asset at a given graph node - Dependencies are between trajectories, not between individual assets - Circular dependencies detected and flagged - Dependency graph visualisable

Traces To: Asset Graph Model §6.3 (Feature Dependencies)

REQ-FEAT-003: Task Planning as Trajectory Optimisation

Priority: High | **Phase**: 1

Tasks shall emerge from feature vector decomposition, not top-down planning.

Acceptance Criteria: - Decompose intent into feature vectors - Map each vector's path through the asset graph - Identify inter-vector dependencies at each graph node - Compress: batch parallel edges, sequence dependent ones - Result: task graph with dependency order, parallelisation, batching

Traces To: Asset Graph Model §6.4 (Task Planning as Trajectory Optimisation) | Ontology #3 (generative principle)

---

7. Full Lifecycle

REQ-LIFE-001: CI/CD as Graph Edge

Priority: High | **Phase**: 2

Deployment shall be a first-class iterative graph transition, not an external afterthought.

Acceptance Criteria: - Code → CI/CD → Running System are admissible graph edges - Evaluators: Deterministic Tests (build, package, deploy checks) - Iteration: failed deployment retries with evaluator feedback - Release manifests list feature vector IDs (REQ keys)

Traces To: Asset Graph Model §7.1 (Beyond Testing)

REQ-LIFE-002: Telemetry and Homeostasis

Priority: High | **Phase**: 2

The running system shall be monitored as a Markov object maintaining its boundary conditions.

Acceptance Criteria: - Telemetry tagged with feature vector IDs (REQ keys) - Homeostasis check: is the running system within constraint bounds? - Evaluators at runtime: Deterministic Tests (alerting, SLA checks, health probes), Agent (anomaly detection), Human (incident response) - Drift detection generates deviation signals (#44)

Traces To: Asset Graph Model §7.2 (Running System as Markov Object), §7.3 (Complete Cycle) | Ontology #49 (teleodynamic), #44 (deviation signal)

REQ-LIFE-003: Feedback Loop Closure

Priority: Critical | **Phase**: 2

Runtime deviations shall generate new intents that re-enter the asset graph as new feature vectors.

Acceptance Criteria: - Deviations generate INT-* intents (automatic or via human review) - Intents include: source (runtime), deviation details, impacted feature vectors - New intents enter graph as new feature vectors - The specification becomes a living encoding (#46) — updated from runtime observation

Traces To: Asset Graph Model §7.4 (Self-Maintaining Specification) | Ontology #46 (living encoding), #49 (teleodynamic)

REQ-LIFE-004: Feature Lineage in Telemetry

Priority: High | **Phase**: 2

Runtime telemetry shall carry REQ keys as structured fields, enabling per-feature observability from spec through production.

Acceptance Criteria: - Telemetry (logs, metrics, traces) includes `req="REQ-*"` as a structured field - REQ keys in telemetry match those in code (`Implements: REQ-*`) - Production queries filterable by REQ key (e.g., latency per feature, error rate per feature) - Incidents traceable to originating requirement via telemetry REQ key

Traces To: Asset Graph Model §6.3 (Feature Lineage in Code and Telemetry) | Ontology #44 (deviation signal)

---

8. Edge Parameterisations

These requirements specify evaluator configuration for common graph edges. They are parameterisations of the universal iteration function, not separate engines.

REQ-EDGE-001: TDD at Code ↔ Tests Edges

Priority: High | **Phase**: 1

The Code ↔ Unit Tests edge shall use TDD workflow as its iteration pattern. TDD is a **co-evolution** — test and code assets iterate together, not a strict directional transition from one to the other.

Acceptance Criteria: - RED: write failing test first (Deterministic Test evaluator reports delta — test exists, code does not yet satisfy it) - GREEN: write minimal code to pass (Deterministic Test evaluator reports convergence) - REFACTOR: improve code quality (all evaluators re-confirm convergence) - COMMIT: save with REQ key in message - The iteration oscillates between test and code assets until both converge — this is a single edge with bidirectional construction - Minimum coverage threshold configurable (default: 80%)

Traces To: Asset Graph Model §9.2 (What Is Preserved — TDD), §2.5 (Graph Scaling — co-evolution as zoomed-in sub-graph) | Context[]: Key Principles

---

REQ-EDGE-002: BDD at Design→Test Edges

Priority: High | **Phase**: 1

Design→Test Cases and Design→UAT Tests edges shall use BDD (Given/When/Then) as their iteration pattern.

Acceptance Criteria: - Scenarios in Gherkin format, tagged with REQ keys - System Test BDD: technical integration scenarios - UAT BDD: pure business language (no technical jargon) - Scenarios are executable, not just documentation - Every REQ key has ≥1 BDD scenario

Traces To: Asset Graph Model §9.2 (What Is Preserved — BDD)

REQ-EDGE-003: ADRs at Requirements→Design Edge

Priority: High | **Phase**: 1

The Requirements→Design edge shall produce Architecture Decision Records as Context[] artefacts.

Acceptance Criteria: - ADRs document: decision, context, consequences, alternatives considered - ADRs acknowledge ecosystem constraints (Context[]) - ADRs reference requirement keys - ADRs are versioned and become part of Context[] for downstream edges

Traces To: Asset Graph Model §5.1 (Context[] contents — ADRs)

REQ-EDGE-004: Code Tagging

Priority: High | **Phase**: 1

Code assets shall include feature vector trajectory tags.

Acceptance Criteria: - Code comments include structured tags: `Implements: REQ-*` (comment syntax is language-specific; the tag format is the contract) - Test comments include structured tags: `Validates: REQ-*` - Commit messages include REQ keys - Tagging validated (not just documentation) — tooling parses the tag format, not the comment syntax

Traces To: Asset Graph Model §6.1 (REQ key as trajectory identifier)

---

9. Tooling

REQ-TOOL-001: Plugin Architecture

Priority: High | **Phase**: 1

Methodology delivered as installable plugins.

Acceptance Criteria: - Plugins include: agent configurations, skills, commands, templates - Discoverable, installable, versioned (semver)

Traces To: Asset Graph Model §5.1 (Context[] — open-ended, not a fixed list) | Ontology #9 (constraint manifold — composable)

REQ-TOOL-002: Developer Workspace

Priority: High | **Phase**: 1

Workspace structure for task and context management.

Acceptance Criteria: - Task tracking: active, completed, archived - Context preservation across sessions - Version-controlled (git)

Traces To: Asset Graph Model §5.3 (Context Stability) | Ontology #40 (encoded representation)

REQ-TOOL-003: Workflow Commands

Priority: Medium | **Phase**: 1

Commands for common workflow operations.

Acceptance Criteria: - Task management: create, update, complete - Context: checkpoint, restore - Status: progress, coverage, gaps

Traces To: Asset Graph Model §3.1 (Iteration Function — operationalising the engine) | Ontology #41 (constructor)

REQ-TOOL-004: Release Management

Priority: High | **Phase**: 1

Versioning and distribution capabilities.

Acceptance Criteria: - Semantic versioning - Changelog generation - Release tagging - Feature vector coverage summary in release notes

Traces To: Asset Graph Model §7.1 (CI/CD as graph asset) | Ontology #7 (Markov object — stable boundary for release)

REQ-TOOL-005: Test Gap Analysis

Priority: High | **Phase**: 1

Identify test coverage gaps against feature vectors.

Acceptance Criteria: - Analyse REQ keys vs existing tests - Identify uncovered trajectories - Suggest test cases for gaps

Traces To: Asset Graph Model §6.1 (Feature as Composite Vector — completeness check) | Ontology #35 (evaluator-as-prompter — gap = delta)

REQ-TOOL-006: Methodology Hooks

Priority: Medium | **Phase**: 1

Lifecycle hooks automating methodology compliance.

Acceptance Criteria: - Hooks trigger on: commit, edge transition, session start - Validate: REQ-* tags present, evaluator convergence recorded - Configurable per project

Traces To: Asset Graph Model §4.1 (Deterministic Tests evaluator type) | Ontology #45 (deterministic compute regime)

REQ-TOOL-007: Project Scaffolding

Priority: High | **Phase**: 1

Installer creates scaffolded project structure.

Acceptance Criteria: - Creates asset graph configuration, context directories, workspace - Templates with placeholder guidance - Design-implementation binding via manifest

Traces To: Asset Graph Model §2.4 (Graph Construction — abiogenesis) | Ontology #39 (abiogenesis — encoding emerges from practice)

REQ-TOOL-008: Context Snapshot

Priority: Medium | **Phase**: 1

Context snapshot for session recovery and continuity.

Acceptance Criteria: - Captures: active tasks, current work context, timestamp - Immutable once created - Integrates with task checkpoint mechanism

Traces To: Asset Graph Model §2.3 (Asset as Markov Object — conditional independence) | Ontology #7 (Markov object — usable without construction history)

REQ-TOOL-009: Feature Views

Priority: High | **Phase**: 1

The system shall generate per-feature cross-artifact status reports by searching for REQ keys across all project artifacts.

Acceptance Criteria: - Given a REQ key, produce a feature view showing which artifacts reference it and at which stage (spec, design, code, tests, telemetry) - Feature views generated by searching tag format (`Implements: REQ-*`, `Validates: REQ-*`, `req="REQ-*"`) across all artifacts - Coverage summary: N of M stages tagged for each feature - Missing stages flagged (e.g., "REQ-F-AUTH-001: no telemetry tagging") - Feature views invocable on demand (e.g., `/aisdlc-trace`)

Traces To: Asset Graph Model §6.4 (Feature Views) | Ontology #15 (trajectory observation)

REQ-TOOL-010: Spec/Design Boundary Enforcement

Priority: Medium | **Phase**: 1

The system shall enforce the Spec/Design boundary — requirements (WHAT) must be technology-agnostic; design (HOW) is technology-bound.

Acceptance Criteria: - Requirements documents must not reference specific technologies, frameworks, or libraries - Design documents must explicitly list technology choices as ADRs - Agent evaluator at the Requirements → Design edge checks for technology leakage in requirements - Multiple design variants per spec are supported (same REQ key, different design trajectories)

Traces To: Asset Graph Model §2.6 (Spec/Design Boundary), §2.7 (Multiple Implementations) | Ontology #40 (encoded representation), #41 (constructor)

Requirement Summary

Category	Count	Critical	High	Medium
Intent & Spec	4	1	2	1
Asset Graph	3	2	1	0
Iteration Engine	2	2	0	0
Evaluators	3	2	1	0
Context	2	0	1	1
Feature Vectors	3	1	2	0
Full Lifecycle	4	1	3	0
Edge Parameterisations	4	0	4	0
Tooling	10	0	6	4
Total	**35**	**9**	**20**	**6**

Phase 1 (Core Graph Engine): 29 requirements

Intent capture + spec, graph topology, iteration engine, evaluators, context, feature vectors, edge parameterisations, tooling (incl. feature views, spec/design boundary).

Phase 2 (Full Lifecycle): 6 requirements

Eco-intent, context hierarchy, CI/CD edges, telemetry/homeostasis, feedback loop closure, feature lineage in telemetry.

Traces To: AI_SDLC_ASSET_GRAPH_MODEL.md — every requirement anchored to a specific section. **Parent Theory**: Constraint-Emergence Ontology — concept numbers (#N) throughout.