

Anthesis: An Agentic Software Development Lifecycle

Anthesis Whitepaper (Draft)

Versioning & Authors

Version	Date	Author	Notes
0.1.0	2025-12-10	Ryan Jennings	Initial draft compiled from Meristem RFCs.
0.1.1	2026-01-18	Ryan Jennings	Expanded details and organizing
0.1.2	2026-02-12	Ryan Jennings	Patch update to whitepaper overview.

1. Executive Summary

Anthesis is a governed agentic SDLC operating system designed to enable autonomous software delivery without surrendering human authority, auditability, or safety.

It treats governance as a first-class system boundary. Autonomy is permitted only when context is complete, risk is understood, and approvals are explicit.

Anthesis exists to answer:

How can AI assist real software development without undermining safety, governance, or trust?

Primary sources:

- CHARTER.md
- meristem/rfc_0001_agentic_sdlc_platform.md

2. System Overview

Anthesis centers on the Model Context Protocol (MCP) as the orchestration and governance layer. MCP coordinates events, agent execution, and audit trails while preserving Git as the system of record.

This separation allows automation to operate safely: agents never mutate Git directly, and all actions flow through MCP-controlled proposals, approvals, and state transitions.

Primary sources:

- `meristem/rfc_0001_agentic_sdlc_platform.md`
- `meristem/rfc_0002_mcp_orchestration_api.md`

3. Governance and Safety Model

Anthesis enforces governance via Calyx policies, risk tiers, and a formal state machine. Approvals gate risky actions and ensure human oversight.

Policy evaluation and state transitions are auditable. This makes the system deterministic about when an agent can act, when it must pause, and how it recovers from failure.

Primary sources:

- `meristem/rfc_0003_approval_policy_engine.md`
- `meristem/rfc_0014_risk_classification_safety_tiers.md`
- `meristem/rfc_0020_state_machine_formal_specification.md`

4. Agent Model

Agents are defined by explicit schemas and execution contracts. They produce proposals and never self-approve. MCP enforces lifecycle rules and idempotency.

Agents are capability-scoped and operate under least-privilege constraints. Execution modes and retries are standardized to ensure predictable behavior across the SDLC.

Primary sources:

- `meristem/rfc_0005_agent_schema.md`
- `meristem/rfc_0009_agent_execution_semantics.md`

5. Context and Determinism

Context assembly is deterministic, provenance-aware, and token-budgeted. This ensures reproducibility across agent runs.

Inputs are structured and attributable, preserving provenance from source artifacts, prompts, and event metadata. The result is a reproducible context snapshot for audits and replay.

Primary sources:

- `meristem/rfc_0007_context_assembly_prompt_composition.md`
- `meristem/rfc_0004_provenance_ingestion.md`

6. Model Runtime Abstraction

Anthesis abstracts model runtimes so agents declare requirements rather than specific backends. MCP resolves runtimes safely and audits drift.

This approach supports local-first deployments while allowing controlled model evolution. Drift budgets and benchmarking constrain unexpected behavior changes.

Primary sources:

- `meristem/rfc_0008_model_runtime_abstraction.md`
- `meristem/rfc_0025_model_evaluation_benchmarking_drift_budgets.md`

7. Observability and Evidence

Executions are fully observable with logs, traces, and audit trails. Evidence remains attributable and immutable.

Observability ties together events, executions, approvals, and outputs so incidents can be investigated and compliance evidence can be exported without ambiguity.

Primary sources:

- `meristem/rfc_0011_observability_telemetry_drift.md`
- `meristem/rfc_0023_audit_compliance_evidence_export.md`

8. Orchestration and Events

Repository events and explicit triggers initiate agent workflows. MCP schedules, retries, and coordinates execution.

Event ingestion provides a consistent entry point for automation, while execution controls prevent duplicate runs and enforce idempotency.

Primary sources:

- `meristem/rfc_0006_repository_event_trigger_model.md`
- `meristem/rfc_0009_agent_execution_semantics.md`

9. Interfaces and Control Surfaces

CLI, UI, and chat interfaces provide operator control. All interfaces map to MCP actions and enforce the same governance rules.

Human control surfaces are designed for transparency and recovery, including approvals, overrides, and kill switches, without granting hidden capabilities.

Primary sources:

- `meristem/rfc_0015_human_interfaces_control_surfaces.md`
- `meristem/rfc_0021_cli_ux_command_semantics.md`

10. Extensibility and Plugins

Plugins extend capability without expanding authority. Governance, auditability, and isolation remain mandatory.

Extensions are constrained by explicit contracts and policy enforcement so they cannot bypass MCP or elevate privileges.

Primary sources:

- `meristem/rfc_0026_plugin_extension_model.md`

11. Security and Identity

Secrets, credentials, and integration boundaries are explicitly managed. Trust boundaries are documented and enforced.

Agents operate without direct access to sensitive material unless explicitly authorized, and integration tokens are scoped and auditable.

Primary sources:

- `meristem/rfc_0012_secrets_credential_management.md`
- `meristem/rfc_0022_external_integrations_mcp_federation.md`

12. Lifecycle and Governance

RFCs, policies, agents, and models follow an explicit lifecycle and ownership model. Amendments are governed and auditable.

This lifecycle prevents silent drift, preserves intent, and ensures future agents can reason about historical decisions.

Glossary

- **Anthesis:** The governed execution state where agents are permitted to act.
- **Phloem (MCP):** The Model Context Protocol server that orchestrates events, context, policies, and execution.
- **Calyx:** Policy engine that evaluates approvals, risk tiers, and governance rules.

- **Bloom Class:** Risk tier classification used to gate approvals and execution.
- **Xylem:** Relay/worker execution plane and scheduling fabric.
- **Meristem:** Constitutional layer containing normative RFCs.
- **Inflorescence:** Multi-agent coordination graphs and workflow orchestration. Primary sources:
 - `meristem/rfc_0017_governance_ownership_rfc_lifecycle.md`
 - `meristem/rfc_0099_v1_unfreeze_amendment_rules.md`

Anthesis — Executive Summary

Anthesis enables governed, asynchronous background agents to automate every stage of the software development lifecycle, while governance ensures humans remain accountable, decisions auditable, and risk explicitly controlled.

The Problem Anthesis Addresses

AI tools can now:

- Generate code
- Modify repositories
- Execute tools
- Propose architectural changes

But they lack:

- Accountability
- Provenance
- Clear authority boundaries
- Auditable decision trails

In most environments today, AI is either:

- **Over-trusted** (silent changes, auto-execution), or
- **Under-used** (manual copy/paste, no integration)

Anthesis exists to answer:

How can AI assist real software development without undermining safety, governance, or trust?

What Anthesis Is

Anthesis is:

- A **governance system** for AI-assisted SDLC
- A **policy-driven control plane** for agents and tools
- A **Git-native source of truth** for intent, decisions, and state
- A **human-in-the-loop framework** by design

It treats AI agents as **contributors**, not authorities.

What Anthesis Is Not

Anthesis is **not**:

- A code generator
- A replacement for IDE tools like Copilot
- A single AI model or vendor platform
- An autonomous “auto-ship” system

Anthesis governs *how* those tools may be used — not *which* tools you use.

Core Principles

- 1. Git Is the Source of Truth** All plans, decisions, approvals, and outcomes are recorded in version control.
 - 2. Agents Propose — Humans Approve** Agents may analyze, draft, and recommend. Final authority remains human.
 - 3. Everything Is Auditable** Every action has provenance: who or what proposed it, under which policy, and why.
 - 4. Risk Determines Oversight** Higher-risk changes require stronger review and approval guarantees.
 - 5. No Silent Execution** No tool execution or state mutation occurs without an explicit, recorded transition.
-

How Anthesis Works (Conceptual Flow)

- 1. Intent is declared** A task, change, or investigation is proposed.
- 2. Agents assist** AI agents analyze context, generate plans, draft artifacts, or surface risks.
- 3. Policies evaluate risk** Governance rules determine required approvals and constraints.

4. **Humans review and approve** Decisions are explicit and recorded.
 5. **Execution is controlled** Tools run only within approved boundaries.
 6. **State is recorded** Outcomes and rationale are committed to Git.
-

Why This Matters (Especially for Regulated Environments)

Anthesis enables:

- Safe adoption of AI in production SDLCs
- Clear audit trails for compliance and incident review
- Controlled use of external tools and models
- Incremental rollout with increasing governance

It is designed for environments where:

- Security matters
 - Compliance matters
 - Mistakes are costly
 - “The agent did it” is not an acceptable explanation
-

The Role of RFCs

Anthesis is governed by a set of **living RFCs**.

- RFCs define what is allowed, forbidden, and required
- RFCs are precise, versioned, and auditable
- This document explains *why* Anthesis exists
- The RFCs define *how Anthesis works*

If there is a conflict, RFCs always win.

Adoption Model

Anthesis is designed to be adopted:

- Gradually
- Incrementally
- Alongside existing tools and workflows

Organizations can start with:

- Planning and review governance
- Then expand to controlled execution
- Then introduce more advanced agent participation

Anthesis Platform PRD

Anthesis is a governed agentic SDLC orchestration platform purpose-built to coordinate AI agents against code repositories while enforcing strict human-in-the-loop controls, policy gates, and comprehensive audit trails. The platform enables automation with deterministic, safe, and fully traceable execution: teams can scale agent workflows without sacrificing control, governance, or risk tolerance.

Goals

Business Goals

- Achieve 90% reduction in ungoverned code changes by enforcing mandatory policy gates and human approvals.
- Enable safe deployment of AI agent workflows in at least 5 enterprise SDLC environments within the first year.
- Demonstrate regulatory and audit compliance for AI-powered SDLC pipelines, opening up new enterprise sales opportunities.
- Drive 50% adoption among engineering leads and platform engineers within pilot client organizations.
- Decrease mean-time-to-remediate risky agent behaviors/events below 2 hours via streamlined oversight and rollback tools.

User Goals

- Ensure all agent-driven code changes are subject to explicit, auditable human approval or rejection.
- Empower platform engineers and ICs to safely compose, propose, and manage complex agent workflows without risk of policy violation.
- Give engineering leads full, real-time visibility and control over agentic SDLC processes, preventing accidental or unauthorized changes.
- Streamline the agent workflow proposal/approval lifecycle to avoid friction while maintaining oversight.
- Provide clear, machine-and-human readable records of all agent actions and governance events.

Non-Goals

- Anthesis does not aim to replace CI/CD pipeline tools; it orchestrates agentic workflows and integrates with existing pipelines.

- The platform will not operate in unsupervised or autonomous mode; human-in-the-loop is always required for sensitive or production-impacting actions.
 - Anthesis is not an AI agent development platform; it serves as the orchestrator and policy/governance layer for existing agent workflows.
-

User Stories

Personas 1. Engineering Lead

- As an Engineering Lead, I want to review and approve AI agent proposals before execution, so that I can ensure all actions comply with company policy and best practices.
- As an Engineering Lead, I want a complete audit trail of agent activity and approvals, so that I can quickly address compliance requests.

2. Platform Engineer

- As a Platform Engineer, I want to define, classify, and compose agentic workflows using policy templates, so that deterministic and compliant automation is guaranteed.
- As a Platform Engineer, I want to trigger proposals via CLI and assign reviewers based on required clearance levels, so that appropriate oversight is enforced.

3. Individual Contributor (IC)

- As an IC, I want to submit code or infrastructure change proposals through Anthesis, so that my work benefits from automated agent support while staying within governance.
 - As an IC, I want machine-readable feedback on rejected proposals, so that I can rapidly iterate and comply with required revisions.
 - As an IC, I want to see which agent actions require human approval, so that I can track and accelerate the review process.
-

Functional Requirements

- **Orchestration Core (Phloem) — Priority: Critical**
 - *Agent Workflow Orchestration*: Execute, monitor, and manage AI agent-driven SDLC workflows.
 - *State Machine Enforcement*: Enforce allowable transitions, preventing agents from acting outside approved workflow states.

- *Output Standardization*: Normalize agent outputs for deterministic audit and review.

- **Governance & Controls — Priority: Critical**

- *Policy Definition & Enforcement*: Allow admins to create/tune policy gates, clearance classifications, and risk thresholds.
- *Approval Workflow*: Route agent proposals through multi-stage human approval with granular, role-based access controls.
- *Audit Logging & Traceability*: Record every agent and human interaction/decision in an immutable, queryable audit log.

- **User Interface & CLI — Priority: High**

- *CLI Submission*: Allow users to compose, test, and submit proposals directly from the command line.
- *Feedback & Notification*: Deliver immediate, clear CLI or webhook-based feedback for approvals, rejections, and policy violations.
- *Machine/Human-Readable Outputs*: Output workflow status, proposal metadata, and audit records in formats suitable for both human and machine consumption.

- **Policy/Classification Library — Priority: Medium**

- *Reusable Policy Templates*: Catalog of company/vetting policies, risk classifications, and action boundaries for workflow reuse.
- *Classification Levels*: Allow for automatic classification tags (e.g., LOW, MEDIUM, HIGH RISK) and clearance requirements on agent actions.

- **Oversight & Rollback Tools — Priority: Medium**

- *Rollbacks*: Manual or automated reversal of agent actions with provenance management.
- *Approval/Denial Comments*: Require rationale for all human overrides/denials to strengthen audit and learning.

User Experience

Entry Point & First-Time User Experience

- Users discover Anthesis either through internal onboarding documentation or via direct CLI/console prompts in their development environments.

- On first use, the CLI provides a guided onboarding flow: user authentication, selection of clearance level, introduction to policy gates, and a sample agentic workflow proposal.
- A tutorial (CLI/console-based) walks users through proposal composition, policy selection, and the review/approval cycle with stepwise feedback.

Core Experience

- **Step 1:** User composes an agentic workflow (e.g., refactor codebase using Agent X) and submits for proposal via Anthesis CLI.
 - Intuitive CLI command structure; prompts for required metadata and classification.
 - Validates proposal structure, flags missing fields, and prevents submission if out of policy.
- **Step 2:** Proposal appears in the governance dashboard (or mirrored CLI feed) for required human review.
 - Role-based access: Only reviewers with sufficient clearance see/rule on the item.
 - Real-time notification to reviewers (email, Slack, webhook).
- **Step 3:** Reviewer opens the proposal, sees policy diff, agent action summary, and risk classification.
 - Provides approve/deny buttons and optional comment box.
 - Record is written to audit log, including rationale if denied.
- **Step 4:** Upon approval, Anthesis executes the agent workflow; output is streamed back to the CLI in real time.
 - Execution is paused if new policy violations are detected post-approval.
- **Step 5:** Every state transition—proposal, approval, execution, post-execution classification—is both human- and machine-readable, (e.g., JSON, RFC-style outline).
 - Rollback option available immediately on post-execution events.

Advanced Features & Edge Cases

- Power users can save and share reusable workflow/policy templates within their organization.
- Supports bulk-approval or multi-proposal batch review for high-velocity pipelines.
- If an agent attempts an unauthorized transition, Anthesis halts execution and files an incident for escalation review.

- Handles disconnected/failed review scenarios by auto-expiring stale proposals and alerting the submitter.

UI/UX Highlights

- CLI output is color-coded for warnings/errors and uses ANSI-safe formatting for accessibility.
 - Responsive web dashboard (for audit/review) supports keyboard navigation and screen readers.
 - All outputs and records can be exported in compliance-friendly, immutable formats (YAML, JSON, RFC text).
 - Minimal multi-step flows; every action requires explicit confirmation and provides context for next steps.
 - High-contrast, scalable elements for both terminal and web UIs to meet AA accessibility standards.
-

Narrative

Taylor, an engineering lead at a large fintech firm, worries about the risk and opacity of letting AI agents modify their codebase. Her team wishes to automate routine maintenance using modern agent tools, but CTO mandates require auditable, policy-governed pipelines—no exceptions.

With Anthesis, Taylor’s platform engineers hook into their code repositories and orchestrate workflows using Phloem. An individual contributor submits a refactoring task via CLI; the workflow is automatically classified as “Medium Risk” due to the codebase size, requiring explicit review. Taylor receives a detailed summary—policy diff, proposed changes, required clearance—right in her Anthesis dashboard and can approve or reject with rationale.

Upon approval, Anthesis executes the workflow under strict state machine supervision, streaming every result, decision, and agent output into a structured audit log. Weeks later, a client regulator requests a full trace: Taylor instantly exports an immutable, human- and machine-readable record of the entire event.

By shifting from opaque automation to governed, agentic SDLC, Taylor not only accelerates the team’s work but dramatically reduces compliance overhead and mitigates operational risk—turning the AI revolution into a business advantage.

Success Metrics

User-Centric Metrics

- Number/proportion of agent proposals submitted through Anthesis vs. “shadow” automation workflows (target: >90% via Anthesis)
- User satisfaction score for core CLI/review UX (target: 4.5/5)
- Average review/approval cycle time (target: <30 minutes)

Business Metrics

- New enterprise customers acquired citing audit/governance as key driver (target: 5 in year one)
- Decrease in compliance incidents or risk events from agentic SDLC (target: 0 critical/major events in pilot phase)
- Percentage reduction in manual policy review workload for leads/managers (target: 60%)

Technical Metrics

- System uptime for orchestration engine and CLI (target: >99.9%)
- Median latency for workflow proposal to execution (<2 minutes under normal load)
- Audit log integrity (zero detected inconsistencies, tampering, or gaps)

Tracking Plan

- Agent workflow submission events
- Approval/denial actions and timings
- Policy violation detections and escalations
- Audit log access/export requests
- Rollback and incident/alert triggers

Technical Considerations

Technical Needs

- Orchestration engine (Phloem): Manages agent workflow lifecycles, execution, and state transitions.
- Policy engine: Evaluates workflows against active policy templates/classifications at proposal and execution time.
- CLI Client: Cross-platform, secure, integrates with user auth and proposal APIs.

- Data models: Immutable audit log objects; proposal, workflow, approval, policy schema.
- Dashboard (optional in v1): For review/approval traceability, policy management.
- APIs: Internal and (optionally) public endpoints for workflow submission, review, policy access.

Integration Points

- SCM/code repository hooks (e.g., GitHub, GitLab)
- Notification/email/Slack/workflow integration (for real-time approvals/reviews)
- SIEM/Compliance tool export endpoints

Data Storage & Privacy

- Immutable audit/event log stored in append-only, encrypted persistence (cloud or on-prem)
- Separation of user data/proposal content from minimal PII (all access/exports logged)
- Meets SOC2, ISO, and regulator requirements for SDLC traceability (for US/EU clients)

Scalability & Performance

- Supports hundreds of concurrent workflow proposals/executions per org (enterprise use case)
- Designed for low-latency review/approval, resilient to reviewer delays
- Horizontal scaling for orchestration and policy enforcement microservices

Potential Challenges

- Guaranteeing out-of-band agent activity is captured/prevented (requires deep SCM hooks and alerting)
- Handling workflows with overlapping/conflicting policy classifications
- User education: ensuring leads/engineers understand the limits and powers of the agentic model
- Ensuring audit log/tamper protection in on-premise deployments

Anthesis SDLC Usage Example

This example shows how a team uses Anthesis inside a repository to automate the SDLC while keeping governance intact.

Phloem is the MCP server; references to MCP below refer to Phloem.

Scenario: Add a Billing Endpoint

- 1) An engineer opens a feature branch and updates `requirements/` with a short PRD note describing the billing endpoint.
- 2) A human runs:

```
anthesis agent run requirements "draft API requirements for billing endpoint"
```

MCP processes the run:

- Pre-Bloom: event recorded.
 - Context Assembly: RFCs and requirements pulled in.
 - Calyx: Bloom Class evaluated; approval required for schema changes.
- 1) MCP creates a proposal and halts at `awaiting_approval`. An approver reviews and approves via CLI or UI.
 - 2) MCP executes the requirements agent and writes updated artifacts under `requirements/` or `meristem/`.
 - 3) The engineer runs:

```
anthesis agent run implementation "implement billing endpoint"
```

- 1) MCP enforces policy, generates a proposal with code diffs, and waits for approval if required.
- 2) The approver approves; MCP applies changes and records execution meta-data.
- 3) The engineer runs:

```
anthesis agent run testing "add tests for billing endpoint"
```

- 1) MCP executes the testing agent and logs outcomes; CI runs as usual and is not bypassed.
- 2) Audit evidence is available via MCP (`/v1/audit/logs`) for compliance and review.

Outcome:

Every agent action is gated by Calyx, recorded via MCP, and replayable or auditable. Humans remain in control; automation is used only where allowed.

Scenario: Repository Modification Triggers SDLC Workflow

This shows a repo-change-driven flow where the human does not call the CLI directly.

- 1) A developer updates `requirements/payment_api.md` and pushes a commit.
- 2) MCP receives the repo event (e.g., `git.commit`) via webhook or a workflow tool.
- 3) Calyx classifies changes under `requirements/**` as Bloom Class 2 and requires approval.
- 4) MCP assembles context and runs a requirements-review agent to identify gaps or draft updates.
- 5) MCP creates a proposal and halts at `awaiting_approval`.
- 6) An approver reviews and approves; MCP applies the agent's updates and logs audit metadata.

Outcome:

The commit itself triggers SDLC automation, but all changes are still gated by policy and human approval.

Scenario: Deferred Execution with Manual Dispatch

This shows a queued execution that is explicitly dispatched later.

- 1) A developer runs:

```
anthesis agent run implementation "refactor payment adapter" --mode deferred
```

- 1) MCP creates the execution in `queued` lifecycle state.
- 2) An operator dispatches it when ready:

```
curl -X POST "$ANTHESIS_MCP_URL/v1/executions/<id>/dispatch" \
-H "X-Anthesis-Key: $ANTHESIS_API_KEY" \
-H "Content-Type: application/json" \
-d '{"reason":"manual dispatch","actor":"operator"}'
```

- 1) MCP advances through context, Calyx, and either awaits approval or completes.

Scenario: Clarification via Approval Payload

This shows how to capture clarifying answers without mid-run input.

- 1) Trigger the agent run:

```
anthesis agent run requirements "clarify billing rules"
```

- 1) MCP creates a proposal that includes the question in `rationale`.
- 2) The approver supplies the answer in the approval reason (or attached payload via the UI).
- 3) MCP records the decision and continues execution under the approved context.

Documentation: Types and Conventions

- **RFCs** (Request for Comments) are proposals for changes or additions to the Anthesis system that may impact requirements and specifications. They provide a structured process for discussing and approving modifications to ensure alignment with the overall goals and design principles of Anthesis.
- **Architecture** documents the technical design and structure of the Anthesis system, including decision records, design notes, and diagrams.
- **Requirements** define the “what” - the necessary conditions and capabilities for the Anthesis system.
- **Specifications** define the “how” - the expected structure, behavior, and interfaces for components that fulfill the requirements.

Meristem: The Living RFCs of Anthesis

This directory is the **public summary view** of Anthesis Meristem RFCs.

- Each `rfc_*.md` file here is a concise, non-normative summary.
- The canonical, normative RFC source is `meristem/`.
- If any discrepancy exists between this directory and `meristem/`, `meristem/` is authoritative.

Meristem is Anthesis’s constitutional layer: the RFC set that defines system behavior, governance, and constraints.

RFC-0000: Anthesis Terminology & Naming Convention

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2025-12-18
- **Last Updated:** 2025-12-18
- **Related RFCs:** None

This RFC canonizes Project Anthesis as the official project name and defines the shared terminology used across all existing and future RFCs.

Summary

- This RFC canonizes Project Anthesis as the official project name and defines the shared terminology used across all existing and future RFCs.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: Canonical Project Name, Canonical Anthesis Vocabulary, RFC Naming Alignment, Backward Compatibility.

RFC-0001: Agentic SDLC Platform

- **Status:** Living
- **Owner:** RJ
- **Created:** 2025-01-13
- **Last Updated:** 2025-01-13
- **Related RFCs:** RFC-0002, RFC-0003, RFC-0004, RFC-0005

This RFC defines an agentic AI platform to support the full Software Development Lifecycle (SDLC) using specialized AI agents coordinated via a Model Context Protocol (MCP) server, with Git as the system of record.

Summary

- This RFC defines an agentic AI platform to support the full Software Development Lifecycle (SDLC) using specialized AI agents coordinated via a Model Context Protocol (MCP) server, with Git as the system of record.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 7. Human Interaction Model, 8. Security Considerations, 2. Context & Motivation, 3. Goals.

RFC-0002: MCP Orchestration API

- **Status:** Living
- **Owner:** RJ
- **Created:** 2025-01-13
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0001, RFC-0003, RFC-0004, RFC-0005, RFC-0033

Defines the Model Context Protocol (MCP) server as the central orchestration and governance layer for agentic workflows interacting with Git repositories.

Summary

- Defines the Model Context Protocol (MCP) server as the central orchestration and governance layer for agentic workflows interacting with Git repositories.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 4.5 Approval-Artifact Mapping (Normative Reference), 4.4 Proposal Taxonomy & Approval Scoping (Normative Reference), 9.1.4 Cross-Repo Query Semantics: "Current" vs "Audit", 9.2 Cross-Org Trust Model (Artifact/Output Exchange Only).

RFC-0003: Approval Policy Engine

- **Status:** Living
 - **Owner:** RJ
 - **Created:** 2025-01-13
 - **Last Updated:** 2026-02-11
 - **Related RFCs:** RFC-0001, RFC-0002, RFC-0027, RFC-0029
-

Defines a declarative policy engine governing when and how agentic actions are permitted to affect the Git repository.

Summary

- Defines a declarative policy engine governing when and how agentic actions are permitted to affect the Git repository.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 2. Policy Model, 2.1 Policy Sources (Normative), 2.3 Policy Composition (Normative), 2.4 Explicit Disable Semantics (Normative).

RFC-0004: Provenance & Ingestion Model

- **Status:** Living
 - **Owner:** RJ
 - **Created:** 2025-01-13
 - **Last Updated:** 2026-02-10
 - **Related RFCs:** RFC-0001, RFC-0002, RFC-0006, RFC-0009, RFC-0023, RFC-0035
-

Defines how human and agent-generated artifacts are ingested, indexed, and attributed.

Summary

- Defines how human and agent-generated artifacts are ingested, indexed, and attributed.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 2. Provenance Model (Normative), 2.2 Linkage Requirements (Normative), 6.1 Integrity and Append-Only Requirements (Normative), 6.2 Boundaries (Normative).

RFC-0005: Agent Schema

- **Status:** Living
- **Owner:** RJ
- **Created:** 2025-01-13
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0001, RFC-0002, RFC-0013, RFC-0031

Defines the standard schema for AI agents participating in the agentic SDLC platform.

Summary

- Defines the standard schema for AI agents participating in the agentic SDLC platform.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 4. Execution Contract (Normative), 6. Lifecycle (Normative), 2. Agent Identity (Normative), 3. Capabilities and Declarations (Normative).

RFC-0006: Repository Event & Trigger Model

- **Status:** Living
- **Owner:** RJ
- **Created:** 2025-01-13
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0001, RFC-0002, RFC-0003, RFC-0004

Defines how changes to the Git repository—whether made by humans or agents—emit events that trigger agentic workflows within the MCP platform.

Summary

- Defines how changes to the Git repository—whether made by humans or agents—emit events that trigger agentic workflows within the MCP platform.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 4. Event Schema (Normative), 5.2 Trigger Evaluation Semantics (Normative), Future Extension: Cross-Repo Triggered Executions (Non-Normative), 3.1 Tracked Refs (Normative).

RFC-0007: Context Assembly & Prompt Composition

- **Status:** Living
 - **Owner:** RJ
 - **Created:** 2025-01-13
 - **Last Updated:** 2026-02-10
 - **Related RFCs:** RFC-0001, RFC-0002, RFC-0004, RFC-0006, RFC-0009, RFC-0030, RFC-0034, RFC-0035
-

Defines how the MCP server assembles contextual information and composes final prompts before invoking LLMs.

Summary

- Defines how the MCP server assembles contextual information and composes final prompts before invoking LLMs.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 7. Provenance Annotation (Normative), 9. Security Considerations (Normative), 4. Retrieval & Ranking (Normative), 5. Context Budgeting (Normative).

RFC-0008: Model & Runtime Abstraction

- **Status:** Living
 - **Owner:** RJ
 - **Created:** 2025-01-13
 - **Last Updated:** 2026-02-10
 - **Related RFCs:** RFC-0001, RFC-0002, RFC-0005, RFC-0007, RFC-0009, RFC-0027, RFC-0035, RFC-0025
-

Defines an abstraction layer between agents and model runtimes, allowing MCP to select, configure, and invoke models (generation and embeddings) without coupling agents to a specific backend.

Summary

- Defines an abstraction layer between agents and model runtimes, allowing MCP to select, configure, and invoke models (generation and embeddings) without coupling agents to a specific backend.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 6.1 Minimal Runtime Contract (Normative), 4. Model Registry, 5. Agent-to-Model Binding, 6.2 Resolution Evidence (Normative).

RFC-0009: Agent Execution Semantics

- **Status:** Living
 - **Owner:** RJ
 - **Created:** 2025-12-22
 - **Last Updated:** 2026-02-14
 - **Related RFCs:** RFC-0001, RFC-0002, RFC-0003, RFC-0005, RFC-0007, RFC-0008
-

Defines how agents are executed, scheduled, retried, and terminated within the MCP platform.

Summary

- Defines how agents are executed, scheduled, retried, and terminated within the MCP platform.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 10.3 Execution Semantics Schema (Normative), 3. Lifecycle States, 3.1 Lifecycle vs Bloom-Phase Mapping, 5.3 Diagnostic Fallback Semantics (Normative).

RFC-0010: CI/CD Integration Model

- **Status:** Living
 - **Owner:** RJ
 - **Created:** 2025-01-13
 - **Last Updated:** 2026-02-10
 - **Related RFCs:** RFC-0001, RFC-0002, RFC-0003, RFC-0006, RFC-0009
-

Defines how agentic workflows integrate with Continuous Integration and Continuous Delivery (CI/CD) systems to validate, gate, promote, and release artifacts.

Summary

- Defines how agentic workflows integrate with Continuous Integration and Continuous Delivery (CI/CD) systems to validate, gate, promote, and release artifacts.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 2.1 Governance Posture Alignment (Normative), 3. Integration Points, 6.1 CI Callback Events (Normative), 7.1 Multi-Environment Promotion (Normative).

RFC-0011: Observability, Telemetry & Drift

- **Status:** Living
- **Owner:** RJ
- **Created:** 2025-01-13
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0002, RFC-0004, RFC-0006, RFC-0007, RFC-0008, RFC-0009, RFC-0010, RFC-0023

Defines the observability framework for the agentic SDLC platform, covering logging, metrics, tracing, and drift detection across agents, models, prompts, and outcomes.

Summary

- Defines the observability framework for the agentic SDLC platform, covering logging, metrics, tracing, and drift detection across agents, models, prompts, and outcomes.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 3.1 Telemetry Record Envelope (Normative), 6.1 Trace Propagation (Normative), 7.1 Drift Inputs (Normative), 8. Replay & Debugging.

RFC-0012: Secrets & Credential Management

- **Status:** Living
- **Owner:** RJ
- **Created:** 2025-01-13
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0002, RFC-0003, RFC-0008, RFC-0009, RFC-0010, RFC-0011

Defines how secrets, credentials, and identities are managed across MCP, agents, CI/CD systems, and integrations (e.g., Git, Slack, email).

Summary

- Defines how secrets, credentials, and identities are managed across MCP, agents, CI/CD systems, and integrations (e.g., Git, Slack, email).
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 6.1 Per-Execution Secret Access (Normative), 8.1 Secret Audit Envelope (Normative), 3. Identity Model, 4.1 Secret Descriptor (Normative).

RFC-0013: Multi-Agent Coordination & Chaining

- **Status:** Living
- **Owner:** RJ
- **Created:** 2025-01-13
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0002, RFC-0005, RFC-0006, RFC-0007, RFC-0009, RFC-0010, RFC-0011

Defines how multiple agents collaborate within a single workflow, including sequencing, fan-out/fan-in patterns, dependency management, and deadlock prevention.

Summary

- Defines how multiple agents collaborate within a single workflow, including sequencing, fan-out/fan-in patterns, dependency management, and deadlock prevention.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 4.2 Workflow Graph Artifact (Normative), 5.1 Execution Linkage (Normative), 6. Approval Boundaries, 6.1 Workflow-Level Bounds (Normative).

RFC-0014: Risk Classification & Bloom Classes

- **Status:** Living
- **Owner:** RJ
- **Created:** 2025-01-13
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0003, RFC-0006, RFC-0007, RFC-0009, RFC-0010, RFC-0011, RFC-0013, RFC-0020

Defines a unified risk model used across the agentic SDLC platform to govern approvals, execution semantics, context limits, CI/CD gates, and human oversight.

Summary

- Defines a unified risk model used across the agentic SDLC platform to govern approvals, execution semantics, context limits, CI/CD gates, and human oversight.
- Highlights the sections that carry the main normative or architectural weight.

- Key sections: 5.2 Authority and Timing (Normative), 5.3 Override Semantics (Normative), 5.1 Risk Computation Inputs (Normative), 9. Security Considerations.

RFC-0015: Human Interfaces & Control Surfaces

- **Status:** Living
- **Owner:** RJ
- **Created:** 2026-01-18
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0002, RFC-0003, RFC-0009, RFC-0014, RFC-0021

Defines required human control surfaces for Anthesis, including approval interactions, visibility, and safe intervention boundaries.

Summary

- Defines required human control surfaces for Anthesis, including approval interactions, visibility, and safe intervention boundaries.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: Approval & Pruning Model, Repo Scoping (Normative), Approval Actions, Visualization & Canonical Projections (Normative, v1).

RFC-0016: Dormancy & Disaster Recovery

- **Status:** Living
- **Owner:** Anthesis Core Team
- **Created:** 2025-12-18
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0003, RFC-0004, RFC-0009, RFC-0010, RFC-0011, RFC-0012, RFC-0014, RFC-0015, RFC-0021, RFC-0023

This RFC defines how Project Anthesis behaves under failure, interruption, corruption, or emergency conditions. It introduces Dormancy as a first-class system state and establishes standards for backup, recovery, replay, and graceful degradation.

Summary

- This RFC defines how Project Anthesis behaves under failure, interruption, corruption, or emergency conditions. It introduces Dormancy as a first-class system state and establishes standards for backup, recovery, replay, and graceful degradation.

- Highlights the sections that carry the main normative or architectural weight.
- Key sections: `Boundaries (Normative)`, `In-Flight Execution Handling (Normative, v1)`, `Replay Retention Window (Normative, v1)`, `Scope (Normative)`.

RFC-0017: Governance, Ownership & RFC Lifecycle

- **Status:** Living
- **Owner:** Anthesis Core Team
- **Created:** 2025-12-18
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0003, RFC-0004, RFC-0014, RFC-0015, RFC-0021, RFC-0023, CHARTER.md

This RFC defines how Project Anthesis governs itself. It establishes ownership models, decision authority, and the lifecycle for RFCs, agents, policies, and models.

Summary

- This RFC defines how Project Anthesis governs itself. It establishes ownership models, decision authority, and the lifecycle for RFCs, agents, policies, and models.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: `Ownership Model`, `Ownership Recording (Normative)`, `Roles & Authority`, `RFC Lifecycle`.

RFC-0018 — Anthesis Tooling & Project Bootstrap

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2025-12-18
- **Last Updated:** 2026-02-12
- **Related RFCs:** RFC-0000, RFC-0001, RFC-0002, RFC-0003, RFC-0006, RFC-0010, RFC-0014, RFC-0015, RFC-0016, RFC-0017, RFC-0021, RFC-0023

This RFC defines the tooling layer that applies Anthesis to new and existing projects. It introduces a CLI-driven bootstrap and adoption model that standardizes governance while preserving project autonomy.

Summary

- This RFC defines the tooling layer that applies Anthesis to new and existing projects. It introduces a CLI-driven bootstrap and adoption model that standardizes governance while preserving project autonomy.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: `Runtime and State (Normative, v1)`, `Plugin System (Normative, v1)`, `Version Pinning (Normative, v1)`, `Integration with n8n and MCP`.

RFC-0019: Anthesis Core Distribution & Update Strategy

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2025-12-18
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0000, RFC-0003, RFC-0014, RFC-0017, RFC-0018, RFC-0021, RFC-0023

This RFC defines how Anthesis Core (shared agents, policies, profiles, and templates) is distributed, versioned, and updated across multiple projects while preserving local autonomy and auditability.

Summary

- This RFC defines how Anthesis Core (shared agents, policies, profiles, and templates) is distributed, versioned, and updated across multiple projects while preserving local autonomy and auditability.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: `Boundaries (Normative)`, `Provenance & Audit, Provenance Recording Location (Normative, v1)`, `Distribution Models`.

RFC-0020: Anthesis State Machine (Formal Specification)

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2025-12-18
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0000, RFC-0003, RFC-0007, RFC-0009, RFC-0014, RFC-0016

This RFC formally specifies the Anthesis State Machine, defining the canonical workflow phases, transitions, invariants, and failure behavior for agentic workflows.

Summary

- This RFC formally specifies the Anthesis State Machine, defining the canonical workflow phases, transitions, invariants, and failure behavior for agentic workflows.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: `Boundaries (Normative)`, `State Transitions`, `Replay Semantics`, `Canonical Workflow Phases`.

RFC-0021: Anthesis CLI UX & Command Semantics

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2025-01-05
- **Last Updated:** 2026-02-12
- **Related RFCs:** RFC-0000, RFC-0009, RFC-0015, RFC-0018, RFC-0020, RFC-0027, RFC-0035

This RFC specifies the command-line interface (CLI) for Project Anthesis. The CLI is a first-class human control surface providing deterministic, scriptable access to Anthesis workflows while preserving governance, approvals, and safety invariants.

Summary

- This RFC specifies the command-line interface (CLI) for Project Anthesis. The CLI is a first-class human control surface providing deterministic, scriptable access to Anthesis workflows while preserving governance, approvals, and safety invariants.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: `Workflow Phase vs Execution State (Normative)`, `3) Approval Flow (If Required)`, `5) Audit & Execution Reads (Optional CLI Surface)`, `anthesis model benchmark`.

RFC-0022: External Integrations & MCP Federation

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2025-12-18

- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0000, RFC-0002, RFC-0006, RFC-0015, RFC-0017, RFC-0018, RFC-0020, RFC-0021, RFC-0023

This RFC defines how Project Anthesis integrates with external systems and how multiple MCP servers may federate securely.

Summary

- This RFC defines how Project Anthesis integrates with external systems and how multiple MCP servers may federate securely.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: `Boundaries (Normative)`, `Integration Principles`, `Integration Classes`, `Inbound Integrations`.

RFC-0023: Audit, Compliance & Evidence Export

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2025-12-18
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0000, RFC-0004, RFC-0011, RFC-0014, RFC-0016, RFC-0020, RFC-0022

This RFC defines how Project Anthesis produces, preserves, and exports audit evidence suitable for internal review, incident forensics, and external compliance frameworks (e.g., SOC 2, ISO 27001).

Summary

- This RFC defines how Project Anthesis produces, preserves, and exports audit evidence suitable for internal review, incident forensics, and external compliance frameworks (e.g., SOC 2, ISO 27001).
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: `Boundaries (Normative)`, `External Auditor Access (Normative, v1)`, `Audit Views`, `Evidence Bundle Manifest (Normative, v1)`.

RFC-0024: Anthesis UI & Visualization Standards

- **Status:** Living
- **Owner:** Anthesis Core

- **Created:** 2025-12-18
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0000, RFC-0009, RFC-0011, RFC-0013, RFC-0015, RFC-0020, RFC-0021, RFC-0023

This RFC defines user interface (UI) and visualization standards for Project Anthesis. It specifies how system state, agent behavior, policies, risk, and evidence are rendered for humans in a way that is accurate, explainable, and safe.

Summary

- This RFC defines user interface (UI) and visualization standards for Project Anthesis. It specifies how system state, agent behavior, policies, risk, and evidence are rendered for humans in a way that is accurate, explainable, and safe.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: Workflow Phase vs Execution State (Normative), 2. Execution Timeline View, 4. Approval Queue, 5. Audit & Evidence Explorer.

RFC-0025: Model Evaluation, Benchmarking & Drift Budgets

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2025-12-18
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0000, RFC-0007, RFC-0008, RFC-0011, RFC-0014, RFC-0020, RFC-0021, RFC-0023

This RFC defines how Project Anthesis evaluates language models, benchmarks their performance, and enforces drift budgets to ensure reliability, safety, and cost control over time.

Summary

- This RFC defines how Project Anthesis evaluates language models, benchmarks their performance, and enforces drift budgets to ensure reliability, safety, and cost control over time.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: Boundaries (Normative), Evaluation Evidence Requirements (Normative, v1), Drift Evaluation Rules (Normative,

v1), Drift Breach Actions (Normative, v1).

RFC-0026: Anthesis Plugin & Extension Model

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2025-12-18
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0000, RFC-0002, RFC-0003, RFC-0014, RFC-0015, RFC-0017, RFC-0018, RFC-0021, RFC-0022

This RFC defines a plugin and extension model for Project Anthesis, enabling third-party and internal extensions while preserving governance, safety, and auditability.

Summary

- This RFC defines a plugin and extension model for Project Anthesis, enabling third-party and internal extensions while preserving governance, safety, and auditability.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: **Boundaries (Normative)**, **Manifest Minimum Contract (Normative, v1)**, **Security Model**, **Sandboxing Requirements (Normative, v1)**.

RFC-0027: Configuration Model

- **Status:** Living
- **Owner:** RJ
- **Created:** 2025-12-28
- **Last Updated:** 2026-02-14
- **Related RFCs:** RFC-0002, RFC-0003, RFC-0012, RFC-0018, RFC-0021

Defines how Anthesis services load configuration across multiple processes and environments, with explicit precedence and security guarantees.

Summary

- Defines how Anthesis services load configuration across multiple processes and environments, with explicit precedence and security guarantees.
- Highlights the sections that carry the main normative or architectural weight.

- Key sections: 12. Transitional Runtime Mode/Topology Contract (Normative, v1), 12.3 Migration-Phase Configuration Contract (Normative, v1), 9. Validation and Auditability, 11. Override Lock Policy (Normative, v1).

RFC-0028: Repo Registry & Identity Resolution

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2026-02-11
- **Last Updated:** 2026-02-19
- **Related RFCs:** RFC-0002, RFC-0006, RFC-0010, RFC-0029, RFC-0035

Defines how Anthesis identifies repositories (`repo_id`) and resolves multiple aliases (`https://`, `ssh://`, `file://`, local paths) to one canonical identity.

Summary

- Defines how Anthesis identifies repositories (`repo_id`) and resolves multiple aliases (`https://`, `ssh://`, `file://`, local paths) to one canonical identity.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 4) Alias Lifecycle Controls (Normative, v1), 5) Evidence Requirements for Alias Changes (Normative, v1), 6) Reachability/Deletion Semantics (Normative, v1), v1 Decisions (Normative).

RFC-0029: Policy Composition & Override Resolution

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2026-02-11
- **Last Updated:** 2026-02-19
- **Related RFCs:** RFC-0002, RFC-0003, RFC-0027, RFC-0028, RFC-0033, RFC-0035

Defines how Anthesis composes policy from organization baseline controls and repo-level overrides, including fail-closed conflict handling.

Summary

- Defines how Anthesis composes policy from organization baseline controls and repo-level overrides, including fail-closed conflict handling.
- Highlights the sections that carry the main normative or architectural weight.

- **Key sections:** 7) Effective Policy Export Contract (Normative, v1), 1) Composition Model (Normative, v1), 2) Configuration Source Precedence (Normative, v1), v1 Decisions (Normative).

RFC-0030: Approval-Artifact Mapping

- **Status:** Living
- **Owner:** RJ
- **Created:** 2025-01-13
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0002, RFC-0003, RFC-0004, RFC-0007, RFC-0023, RFC-0033

Defines how approval decisions map to specific artifacts (path + commit) so that context blocks can report deterministic, auditable approval_status metadata.

Summary

- Maps approvals to exact artifacts using path plus commit identity.
- Prevents vague or overbroad approval claims from leaking across changes.
- Supports deterministic approval-status reporting in context assembly and audit surfaces.
- **Key sections:** 8. Provenance & Audit Alignment (RFC-0004 / RFC-0023), 5. Data Model (Logical), 5.2 Approval-Artifact Link, 6. Mapping Rules.

RFC-0031: Agent Classes & Trust Model

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2026-01-17
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0003, RFC-0005, RFC-0013, RFC-0014, RFC-0023

Defines canonical Anthesis agent classes and trust boundaries, separating authority from execution with explicit non-escalation constraints.

Summary

- Defines canonical Anthesis agent classes and trust boundaries, separating authority from execution with explicit non-escalation constraints.
- Highlights the sections that carry the main normative or architectural weight.

- Key sections: Action Authority Matrix (Normative, v1), Authority Flow (Hard Rule), Canonical Class IDs (Normative, v1), Calyx - Authority & Accountability.

Why It Matters

- This RFC gives Anthesis an authoritative reference for canonical Anthesis agent classes and trust boundaries, separating authority from execution with explicit non-escalation constraints.
- Without that reference, the same concern is likely to drift across services, tools, or workflows.

RFC-0032: External Tool Integration & Admission

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2026-01-17
- **Last Updated:** 2026-02-10
- **Related RFCs:** RFC-0003, RFC-0014, RFC-0022, RFC-0023, RFC-0026, RFC-0031

Defines admission controls and governance requirements for external tools and agent integrations, including policy, auditability, and trust-boundary constraints.

Summary

- Defines how external tools and agent integrations may be admitted into Anthesis.
- Requires explicit policy, scope, provenance, and auditability constraints.
- Treats integrations as trust-boundary crossings rather than neutral helpers.
- Key sections: Admission Contract (Normative, v1), Admission Lifecycle (Normative, v1), Expiration Policy (Normative, v1), Provenance Requirements.

Why It Matters

- Anthesis is agentic and tool-capable, so unmanaged integrations can silently expand authority and risk.
- This RFC keeps external tool use governed, attributable, and revocable.

RFC-0033: Proposal Taxonomy & Approval Scoping Strategy

- **Status:** Living

- **Owner:** Anthesis Core
- **Created:** 2026-02-05
- **Last Updated:** 2026-02-19
- **Related RFCs:** RFC-0002, RFC-0003, RFC-0007, RFC-0020, RFC-0023, RFC-0030

This RFC defines a minimal, normative proposal taxonomy and the rules that govern approval scoping in Anthesis.

Summary

- Defines the proposal kinds Anthesis recognizes.
- Limits which proposal types may confer artifact-level approval scope.
- Prevents execution or operational approvals from being misread as artifact authorization.
- Key sections: 0. Boundaries (Normative), 6. Approval Scoping Rules (Normative), 6.2 Execution Gate Semantics, 5. Proposal Taxonomy (Normative).

Why It Matters

- Approval scope is a governance boundary, not a convenience label.
- This RFC prevents Anthesis from inferring artifact approval from the wrong proposal type and keeps approval semantics reviewable.

RFC-0034: Sessions & Prompt Context Governance

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2026-01-26
- **Last Updated:** 2026-02-11
- **Related RFCs:** RFC-0000, RFC-0004, RFC-0007, RFC-0009, RFC-0021, RFC-0023, RFC-0027, RFC-0030

This RFC defines sessions and the governance rules for prompt context in Anthesis. It introduces a machine-readable session declaration, the Core Invariant vs Session Overlay model, and provenance expectations required to reconstruct the Effective Prompt after execution.

Summary

- Defines sessions as governed prompt-context declarations.
- Separates always-on core invariants from narrower session overlays.
- Requires machine-readable session declarations and provenance sufficient to reconstruct effective prompt context.

- Key sections: 4.1 Boundaries (Normative), 7.3.1 Minimum Policy Snapshot (Normative, v1), 8. Audit & Provenance, 9. CLI Integration (Normative Reference).

RFC-0035: Operating Modes (Offline / Local / Remote)

- **Status:** Living
- **Owner:** RJ
- **Created:** 2026-01-29
- **Last Updated:** 2026-02-14
- **Related RFCs:** None

This RFC defines Anthesis operating modes—offline, local, and remote—and the constraints, service dependencies, and governance expectations that apply to each. The goal is to make runtime posture explicit, predictable, and auditable while preserving RFC-level invariants (policy enforcement, auditability, and safe defaults).

Summary

- Defines offline, local, and remote operating modes.
- Makes runtime posture explicit in terms of available services and governance constraints.
- Preserves Anthesis invariants even when parts of the system are unavailable.
- Key sections: Transitions (Normative), Transition Safety and Recoverability (Normative), Governance Requirements by Mode (Normative), Runtime Axes (Normative, v1).

RFC-0036: QART Engineering

- **Status:** Living
- **Owner:** Governance Layer
- **Created:** 2026-02-21
- **Last Updated:** 2026-02-23
- **Related RFCs:** RFC-0037, RFC-0038, RFC-0099

This RFC defines QART Engineering as the required pre-decision governance discipline for significant architectural intent.

Summary

- This RFC defines QART Engineering as the required pre-decision governance discipline for significant architectural intent.

- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 5. The Four Phases (Normative Specification), 8. Escalation Model, 9. Compliance Requirements, 12. AI-Agent Participation Policy.

RFC-0037: Drift Loop Engineering

- **Status:** Living
- **Owner:** Governance Layer
- **Created:** 2026-02-21
- **Last Updated:** 2026-02-23
- **Related RFCs:** RFC-0036, RFC-0038, RFC-0099

This RFC defines Drift Loop Engineering as the governed control loop for detecting, classifying, reconciling, and auditing drift between canonical intent and implementation reality.

Summary

- This RFC defines Drift Loop Engineering as the governed control loop for detecting, classifying, reconciling, and auditing drift between canonical intent and implementation reality.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 5. Normative Requirements, 14. Drift Lifecycle State Machine, 4. System Model, 4.2 Loop Lifecycle.

RFC-0038: Unified Governance Control Doctrine

- **Status:** Living
- **Owner:** Governance Layer
- **Created:** 2026-02-21
- **Last Updated:** 2026-02-23
- **Related RFCs:** RFC-0036, RFC-0037, RFC-0099

This RFC defines Unified Governance Control as the meta-doctrine that binds QART pre-decision governance with Drift Loop post-decision governance.

Summary

- Binds QART and Drift Loop into one governance doctrine.
- Connects pre-decision review with post-decision drift handling.
- Defines governance as a continuous control system rather than isolated checks.

- Key sections: 11.1 State Definitions, 11.2 Transitions and Guards, SECURITY_SENSITIVE, 12.2 Mode Transition Rules.

RFC-0039: MCP Transport Profiles (`http`, `stdio`, `sse`)

- **Status:** Living
- **Owner:** TBD
- **Created:** 2026-02-22
- **Last Updated:** 2026-02-22
- **Related RFCs:** RFC-0002, RFC-0003, RFC-0017, RFC-0021, RFC-0027, RFC-0035, RFC-0099

This RFC defines governed MCP transport profile support in Phloem across `http`, `stdio`, and `sse` with explicit mode and security constraints.

Summary

- This RFC defines governed MCP transport profile support in Phloem across `http`, `stdio`, and `sse` with explicit mode and security constraints.
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: 6. Security Requirements (Proposed Normative), 8. Configuration Contract (Proposed Normative), 7.2 Startup Validation Matrix (Normative, v1), 10. Audit and Evidence Requirements.

RFC-0040: Install Roots & Search Paths

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2026-03-10
- **Last Updated:** 2026-03-10
- **Related RFCs:** RFC-0017, RFC-0018, RFC-0021, RFC-0027, RFC-0028, RFC-0034

This RFC defines governed install roots and deterministic search path resolution for Anthesis prompt and session assets.

Summary

- This RFC defines governed install roots and deterministic search path resolution for Anthesis prompt and session assets.
- Highlights the sections that carry the main normative or architectural weight.

- Key sections: 11. Security Model (Normative), 4.3 Auditability and Provenance Requirement, 10. Contract (Normative), 10.6 Provenance and Output Requirements.

RFC-0041: Mode/Topology Transition Configuration Contract

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2026-03-10
- **Last Updated:** 2026-03-10
- **Related RFCs:** RFC-0017, RFC-0021, RFC-0027, RFC-0035, RFC-0039

This RFC defines the configuration contract for operating mode and execution topology transitions, extracting the transition-specific posture rules from RFC-0027 into a dedicated control surface.

Summary

- Defines the config contract for mode and topology transitions.
- Separates transition posture rules from general configuration concerns.
- Requires reproducible, auditable, fail-closed transition behavior.
- Key sections: 4. Normative Contract (Extracted from RFC-0027 §12), 4.3 Runtime Validation Requirements, 4.4 Worker Topology Queue Configuration, 4.6 Migration-Phase Contract.

RFC-0042: Loop Contract and UGC Bridge Semantics

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2026-03-17
- **Last Updated:** 2026-03-17
- **Related RFCs:** RFC-0009, RFC-0013, RFC-0017, RFC-0020, RFC-0021, RFC-0022, RFC-0036, RFC-0037, RFC-0038

This RFC defines a loop contract for Inflorescence-managed loop types and a governed bridge model for relating loop instances under Unified Governance Control, without changing Anthesis workflow-phase law or execution lifecycle semantics.

Summary

- Defines governed loop and bridge semantics for Inflorescence-managed lifecycle artifacts.

- Relates loop instances without changing Anthesis execution law.
- Keeps loop coordination subordinate to existing workflow and governance primitives.
- Key sections: 16. Authority Boundaries, 6. Loop Contract, 7. Bridge Contract, 8. Loop Types And Stage Semantics.

RFC-0043: RFC Governance Model

- **Status:** Review
- **Owner:** RJ
- **Created:** 2026-03-22
- **Mutability:** Amendable

This RFC defines the lifecycle, mutability model, and governance rules for RFCs within Anthesis, treating RFCs as governed, versioned, machine-readable system artifacts rather than static documents.

Summary

- Defines the lifecycle and mutability model for Anthesis RFCs.
- Treats RFCs as governed system artifacts rather than static prose.
- Sets expectations for amendment, transition, and enforcement behavior.
- Key sections: Lifecycle Model, Core Mental Model, States, Mutability Model.

RFC-0044: gittuf as Git-Native Policy and Verification Substrate

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2026-03-26
- **Last Updated:** 2026-03-26
- **Related RFCs:** RFC-0003, RFC-0004, RFC-0010, RFC-0016, RFC-0017, RFC-0021, RFC-0023, RFC-0028, RFC-0030, RFC-0032, RFC-0033, RFC-0035

This RFC defines how Anthesis integrates gittuf as the Git-native policy enforcement and verification substrate for protected refs, protected paths, and approval-backed repository changes without replacing Anthesis governance.

Summary

- Uses gittuf as the repository-native verification and policy layer.

- Keeps Anthesis governance, approvals, and evidence semantics authoritative.
- Protects refs, paths, and release workflows with portable verification.
- Bridges Anthesis approvals into Git-native attestations without making gittuf the governance system.
- Key sections: 6.1 Trust Model, 6.2 Policy Model, anthesis trust attest-approval, 4. Mental Model.

RFC-0045: Tracks as Governed Coordination Artifacts

- **Status:** Living
- **Owner:** RJ
- **Created:** 2026-03-19
- **Last Updated:** 2026-03-26
- **Related RFCs:** RFC-0002, RFC-0009, RFC-0020, RFC-0023, RFC-0033, RFC-0034, RFC-0038

This RFC defines tracks as governed, repo-scoped coordination artifacts that group references to authoritative Anthesis artifacts for bounded streams of work, without introducing new execution authority, approval scope, or runtime state.

Summary

- Defines tracks as reference-oriented coordination artifacts only.
- Lets Anthesis group tasks, sessions, approvals, executions, and evidence for bounded workstreams.
- Prohibits tracks from becoming execution containers, approval grants, or runtime state objects.
- Key sections: 4. Authority Boundaries, 9. Lifecycle and Removal Semantics, 14. Audit Model, 3. Normative Basis.

RFC-0046: SSDF Integration and Enforcement Model

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2026-03-22
- **Last Updated:** 2026-03-26
- **Related RFCs:** RFC-0002, RFC-0003, RFC-0009, RFC-0010, RFC-0020, RFC-0023, RFC-0032, RFC-0038, RFC-0044, RFC-0045

This RFC defines how NIST SSDF is integrated into Anthesis as a governed security baseline and enforcement model across policy, evidence, scanning, artifact validation, and vulnerability response, without redefining runtime authority.

Summary

- Maps SSDF into Anthesis as governed security requirements.
- Enforces security through policy, validation, evidence, and response loops.
- Keeps SSDF subordinate to Anthesis governance rather than creating a parallel compliance system.
- Uses existing Anthesis primitives instead of inventing new execution states or pipelines.
- Key sections: 9. Audit Model, 5. Normative Basis, 6. SSDF Mapping, 7. Enforcement Model.

RFC-0047: Session Declaration Persistence and Replay Contract

Summary

- Stores Anthesis session declarations as immutable Phloem records.
- Requires append-only linkage from sessions to executions.
- Makes server-side normalized hashing authoritative.
- Supports replay as attributable context-lineage reconstruction.
- Leaves evidence packaging and retention under RFC-0023.
- Key sections: 7. Replay Semantics, 4. Session Record Model, 6. Session-to-Execution Association, 1. Purpose.

RFC-0048: Phloem Policy Evaluation and Decision Contract

Summary

- Defines Phloem's normalized evaluation context and policy result vocabulary.
- Requires strictest-wins aggregation and machine-readable explanations.
- Defines the PDR as the authoritative Phloem decision record.
- Allows PDR-derived attestations without turning them into approval artifacts.
- Keeps approval, evidence, and Git-native verification semantics in their existing canonical RFCs.
- Key sections: 5. Policy Results and Aggregation, 6. Evaluator Explanation Contract, 7.2 Subject Model, 7.3 Decision Model.

RFC-0049: Security Agent Input/Output Contract

Summary

- Defines the bounded producer contract for Anthesis security agents.
- Standardizes review inputs, verdicts, findings, and evidence summaries.
- Keeps agent output advisory to Phloem rather than gate-authoritative.

- Prevents agent findings from replacing human approval or deterministic controls.
- Key sections: 6. Output Contract, 1. Purpose, 2. Scope, In Scope.

RFC-0050: Anthesis-Generated Commit Trailer Policy

Summary

- Defines an optional advisory trailer for Anthesis-generated commits.
- Keeps trailer metadata subordinate to governed evidence and RFC-0044 verification.
- Prohibits inferring approval or authorization from trailer presence alone.
- Defines corroboration, precedence, and history-rewrite handling rules.
- Key sections: 5. Consumer Semantics, 6. Security and Compliance, 1. Purpose, 2. Scope.

RFC-0051: Memory Model and Authority Rules

Summary

RFC 0051 establishes a governance-driven memory model for Anthesis. It clarifies what memory is authoritative, how memory sources are ranked, and how agents/tools should read/write state in a multi-agent MCP environment.

What this RFC defines

- Memory classes and sources (e.g. current goal, constraints, decisions, outcomes)
- Authority precedence
- Read policy (priority loading of normative + structured state)
- Write policy (explicit paths, no silent authority changes)
- Session lifecycle behavior (startup and shutdown state sync)
- Conflict/drift handling rules
- Provenance minimums
- Security and token-efficiency principles

Why it matters

- Creates a single coherent memory model across Anthesis agents and tools
- Prevents chat history from becoming implicit long-term truth
- Enables predictable governance enforcement and auditability
- Supports safer prompt construction and state mutation with checkable provenance
- Lays groundwork for future features like memory snapshots, rollbacks, and cross-agent state sharing

RFC-0099: Anthesis v1 Unfreeze & Amendment Rules

- **Status:** Living
- **Owner:** Anthesis Core
- **Created:** 2025-12-19
- **Last Updated:** 2026-02-11
- **Related RFCs:** RFC-0000, RFC-0003, RFC-0020, RFC-0023, RFC-0030, RFC-0033

This RFC transitions Anthesis v1 from Frozen to Active (Amendments Allowed).

Summary

- This RFC transitions Anthesis v1 from Frozen to Active (Amendments Allowed).
- Highlights the sections that carry the main normative or architectural weight.
- Key sections: Emergency Security Amendments (Normative, v1), Lifecycle Controls (Normative), v1 Corpus Boundary (Normative), Precedence and Conflict Handling (Normative).

Calyx: Policy and Approval Governance

Calyx is Anthesis' **policy and approval governance system**. It defines how changes are classified by risk, which actions require human approval, and how the system maintains accountability over all agentic decisions.

Purpose

Calyx exists to answer:

Is this action allowed? Who must approve it? Can we prove they did?

Calyx is the **enforcement mechanism** that ensures agents operate only within explicitly defined boundaries, with transparent audit trails.

Core Concepts

Bloom Classes Bloom Classes are **risk tiers** assigned to artifacts and changes:

Class	Label	Definition	Approval Required
0	Read-Only	Documentation, analysis	No
1	Low Risk	Prompts, tests, minor refactors	No
2	Medium Risk	Application code, workflows	Yes (reviewer)


```

+-----+-----+
                        ↓
+-----+-----+
| 4. Execute |
|   Agent commits changes to Git |
|   Audit trail: Action ID → Calyx decision → Commit |
+-----+-----+

```

Policy Evaluation Outcomes

Outcome	Description	Next Step
Auto-Approve	No approvals required	Execute immediately
Await Approval	Human approval needed	Halt; wait for approver
Reject	Policy forbids action	Fail; log reason
Escalate	Route to higher authority	Forward to admin channel

Approval Channels

Approvals can be collected from:

1. **GitHub PR Reviews** - Reviewer approves PR
2. **CLI Confirmation** - Human confirms in terminal
3. **n8n Workflows** - Approval action in workflow
4. **MCP API** - Programmatic approval submission

All approvals are **logged immutably** with:

- Approver identity
- Timestamp
- Action ID
- Decision rationale (if provided)

Example Scenarios

Scenario 1: Low-Risk Change (Auto-Approve)

```

Agent: "Update prompts/system-message.md"
File: prompts/system-message.md
Bloom Class: 1 (Low Risk)
Policy Match: default_low_risk
Approvals Required: []
Result: Auto-approved → Execute

```

Scenario 2: Medium-Risk Change (Require Review)

```

Agent: "Refactor src/lib/domain/agent.ts"
File: src/lib/domain/agent.ts

```

Bloom Class: 2 (Medium Risk)
Policy Match: default_medium_plus
Approvals Required: [reviewer]
Result: Awaiting Approval → Notify reviewers

Scenario 3: Critical Change (Require Admin)

Agent: "Update .env with new API key"
File: secrets/.env.production
Bloom Class: 4 (Critical)
Policy Match: critical_secrets
Approvals Required: [admin]
Result: Admin approval only → Escalate to admin channel

Audit Logs Every Calyx decision is logged to:

- **MCP Event Log** - Structured JSON
- **Git Commit** - Approval metadata in commit message
- **External SIEM** - Via audit sink (if configured)

This ensures a **verifiable audit trail** for compliance and review.

Inflorescence: Multi-Agent Coordination

Inflorescence is Anthesis' **multi-agent coordination system**. It defines how multiple agents collaborate within workflow graphs, enabling complex, observable, and safe agent-to-agent coordination.

Purpose

Inflorescence answers:

*How can multiple agents work together on a single problem safely?
Who depends on whom? What happens when something fails?*

Inflorescence is the **orchestration layer** that turns individual agents into a coordinated team.

Core Concepts

Coordination Graphs Coordination graphs are **directed acyclic graphs (DAGs)** where:

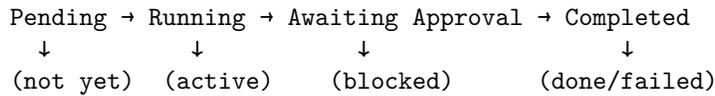
- **Nodes** represent agent executions
- **Edges** represent data flow or approval dependencies
- **Workflows** describe task sequences, fan-out/fan-in, and failure strategies

Graphs are declarative, versioned in Git, and enforced by Anthesis at runtime.

Coordination Patterns Inflorescence supports standard workflow patterns:

Pattern	Description	Use Case
Sequential	A → B → C	Linear workflows
Fan-out	One agent spawns many	Parallel analysis
Fan-in	Multiple agents → aggregator	Consolidate results
Map/Reduce	Partition → analyze → synthesize	Large-scale tasks
Supervisor/Worker	Controller delegates scoped tasks	Distributed execution
Conditional	Branch on intermediate results	Decision trees

Execution States Each node in a graph transitions through states:



Workflow Format

Inflorescence workflows are **YAML files** that define nodes, edges, and execution parameters.

Example 1: Simple Sequential Three agents run in sequence:

1. Agent A: Gather Requirements
2. Agent B: Validate Requirements
3. Agent C: Finalize Documentation

Example 2: Fan-Out / Fan-In One trigger spawns multiple parallel agents, then aggregate.

1. Read Specification file
2. Fanout: Three parallel agents analyze different sections
 - Agent A: Analyze Performance
 - Agent B: Analyze Security
 - Agent C: Analyze Usability
3. Fan-in: Aggregator agent synthesizes findings into a report

Example 3: With Approval Gate Workflow with human approval check-point:

1. Agent A: Deployment Planner
2. Agent B: Deployment Review Requires Approval
3. Agent C: Execute Deployment (only if approved)

Example 4: Failure Handling Workflow that continues if some nodes fail

1. Agent A: File Analysis (must succeed)
2. Agent B: File Analysis (must succeed)
3. Agent C: File Analysis with leniency (if it fails, log and continue)

Validation & Safety

Inflorescence enforces:

- **DAG constraint:** No cycles allowed
- **Acyclic graphs:** Prevent infinite loops
- **Execution limits:** Max depth, max nodes per execution
- **Timeouts:** Per-node and global workflow timeouts
- **No shared state:** Agents communicate only via edges
- **Immutable history:** All executions logged and replayed

Agents Agents are nodes in workflows. They:

- Receive inputs via templated substitution
- Produce JSON output
- Can specify approval requirements
- Respect Calyx policies

Inflorescence + Calyx Workflows are subject to Calyx policies:

Workflow → Node Execution → Calyx Classification → Approval Gate

Monitoring & Observability

Every state transition emits an event that is queryable via MCP.

Metrics

- Workflow execution time
- Node success/failure rates
- Approval wait time
- Data flow volume
- Approval decision distribution

Best Practices

1. **Keep workflows simple** - Fewer than 10 nodes per workflow
2. **Name nodes clearly** - Use descriptive, actionable IDs
3. **Document inputs/outputs** - Specify data contracts
4. **Set appropriate timeouts** - Prevent hung executions
5. **Plan failure strategies** - Decide at design time
6. **Version workflows** - Increment version on changes
7. **Test locally** - Use workflow simulator before deploying

8. **Monitor execution** - Watch approval and failure metrics
9. **Iterate** - Refine workflows based on real-world usage

Reference

Related RFCs

- RFC-0013: Multi-Agent Coordination - Coordination patterns
- RFC-0009: Execution Semantics - Execution states
- RFC-0005: Agent Schema - Agent definitions

Services: The Execution Layer of Anthesis

Microservices that power Anthesis: Phloem (MCP server & orchestration) and Xylem (worker execution service).

Service Architecture

Services (Tier 3: Execution)

```
|
+-- Phloem (Orchestration & API)
|   +-- MCP Server implementation
|   +-- Request routing and validation
|   +-- Policy integration (Calyx)
|   +-- Proposal management
|   +-- Outbox execution queue
|   +-- State management
|   +-- Audit logging
|
+-- Xylem (Worker Execution)
|   +-- Message queue consumer
|   +-- Agent execution
|   +-- Context assembly
|   +-- Output validation
|   +-- Result collection
|   +-- Error handling
|   +-- Queue initialization (scripts/)
|
+-- Xylem (Embedded Execution)
    +-- Local execution mode for CLI
    +-- Same execution logic as worker
    +-- Runs in-process with CLI for low-latency operations
```

Service Communication

Request Flow

1. Client → Phloem: Request
2. Phloem → Calyx: Classification
3. Phloem → Xylem: Dispatch
4. Xylem → Phloem: Claim
5. Xylem → Phloem: Dispatch/Execute
6. Response to Client

Service SLOs

Target service level objectives:

Metric	Target
Availability	99.9% (9h/month downtime)
Latency (p50)	< 500ms
Latency (p99)	< 5s
Error Rate	< 0.1%
Queue Latency	< 1s (p95)

Phloem Service: The Anthesis MCP Server and Orchestration Hub

Phloem is the Anthesis MCP server implementation. It exposes the orchestration API, enforces policy via Calyx, and records execution/audit data for governed workflows.

Purpose: Central orchestration hub, MCP server, policy enforcement

Responsibilities:

- Receive and parse incoming requests
- Classify requests with Calyx
- Route work to executors
- Aggregate and return results
- Record audit trail
- Handle approvals and callbacks

Configuration

Runtime Posture The runtime posture is modeled as four axes:

- `operating_mode`: `offline` | `local` | `remote`
- `execution_topology`: `embedded` | `worker`
- `relay_mode`: `embedded` | `process`
- `deployment_scope`: `single` | `distributed`

Canonical runtime profiles and configuration overlays can be defined as combinations of these axes. See Configuration for details on how to configure these profiles and overlays.

Queue Backend Configuration Phloem uses a vendor-agnostic queue abstraction layer. Configure via environment variables.

Xylem Service: The Anthesis Execution Agent and Queue Consumer

Xylem is the execution agent for Anthesis. It consumes dispatch messages from the Xylem message queue, claims executions from the Phloem MCP server, and relays dispatch notifications back to the API.

Xylem has two operating modes depending on runtime configuration:

1. **Queue Consumer Mode** (default) - Subscribes to configured queue backend, consumes dispatch messages, claims executions, and relays notifications.
2. **Embedded Mode** - Runs as a standalone process without queue consumption, useful for testing or environments without queue backends.

Xylem operates with Phloem (MCP server) to claim and execute tasks/workflows and relay notifications. It also supports health monitoring, reporting, and graceful shutdown.

Responsibilities:

- Subscribe and initialize work queues
- Load execution context
- Execute agents with scoped access
- Validate output
- Handle errors gracefully
- Report results back
- Health monitoring

Worker Types:

- **Agent Worker** - Executes agents
- **Tool Worker** - Executes external tools
- **Approval Worker** - Handles approval callbacks
- **Notification Worker** - Sends notifications

Execution Flow

1. Worker receives dispatch message from queue backend
2. Extracts `execution_id` from payload
3. Claims execution: `POST /v1/executions/{id}/claim`
4. Executes agent or workflow with context from MCP

5. Relays dispatch: `POST /v1/executions/{id}/dispatch`
6. Acknowledges message to queue backend
7. Reports status to provenance queue (if enabled)
8. Handles errors and retries according to policy

Scaling:

- Horizontal scaling (add more workers)
- Different worker pools for different task types
- Prefetching and batching for efficiency
- Auto-restart on failure
- Graceful shutdown on signals

Tooling: Building, Testing, and Deploying Anthesis

Development tools and utilities for Anthesis: CLI, AI client library, templating system, and test infrastructure. Everything needed to build, test, and deploy the platform.

Included Tools

1. CLI (Command-Line Interface) Anthesis CLI is the local control surface for governed workflows, providing scriptable commands that mirror MCP operations while enforcing Anthesis policy and plugin contracts.

2. Embedded Runtime Anthesis Embedded Runtime is a lightweight execution environment for running agents and workflows without a worker deployment. It provides a direct API to the Phloem MCP server and manages context in-memory, making it ideal for testing, development, and environments without queue backends.

3. AI Client Library `ai_client` is a shared Python library providing OpenAI-compatible client helpers for AI runtime integration. It manages model discovery, configuration loading, context, and chat completion requests across Anthesis services and plugins. It supports multiple backends (Ollama, LM Studio, etc.) and provides a consistent interface for AI interactions.

4. Observability & Logging Standardized logging and observability utilities for consistent instrumentation across services and agents. Provides structured logging, OpenTelemetry integration, and centralized configuration for log levels and formats. This ensures that all components of Anthesis emit consistent, queryable logs that can be easily integrated with monitoring and alerting systems.

5. Message Queue Abstraction Vendor-agnostic message queue abstraction layer supporting RabbitMQ, Redis Streams, Apache Kafka, and AWS SQS.

Provides a consistent interface for enqueueing and consuming messages, allowing Phloem and Xylem to operate with different queue backends based on deployment needs. Configuration is handled via environment variables, enabling flexibility without code changes.

6. Templates Shared Jinja2 templates for Anthesis tooling, distribution packaging, and test scaffolding. Templates ensure consistency in generated artifacts, such as agent definitions, workflow graphs, and plugin manifests. They can be extended by plugins to provide domain-specific templates for various use cases.

Template Types:

- **Workflow templates** - Assemble execution context for agents
- **Artifact templates** - Format changes as commits or diffs
- **Context templates** - Build agent input from previous outputs
- **Output templates** - Format agent output for downstream use

7. Test Infrastructure Mainly focused on integration testing between control surfaces, data, and services. Tests are split into **unit** and **integration**. Integration testing happens through services and the CLI in docker compose. Coverage is tracked and CI/CD pipelines enforce minimum thresholds.

Tools Overview

Tool	Purpose	Status
CLI	Command-line interface	Active
Embedded Runtime	Local execution environment	Active
ai_client	AI runtime library	Active
templates	Jinja2 templating	Active
message_queue	Message queue abstraction	Active
observability	Logging and instrumentation	Active
tests	Test infrastructure	Active

Anthesis CLI: The Local Control Surface for Governed Workflows

Anthesis CLI is the local control surface for governed workflows, providing scriptable commands that mirror MCP operations while enforcing Anthesis policy and plugin contracts.

Features

- Sessions for invariant and overlay prompts, context, and configuration
- Plugin command execution with policy enforcement

- Execution of agents and workflows
- Agent management
- Model governance and configuration
- RFC management
- AI client library for model interactions
- Observability utilities for consistent logging and instrumentation
- System runtime profiles for different environments (local, staging, production)
- Tool for integration testing

Plugin registry loading

The CLI loads plugins from a registry configured in user or project scope. The registry defines available plugins, their sources (local or release), and configuration.

Configuration hierarchy

Anthesis CLI configuration is loaded from multiple sources with the following precedence (highest to lowest):

1. CLI arguments
2. Environment variables
3. User config file (`~/.config/anthesis/config.yaml`)
4. Project config file (`anthesis/config.yaml`)

Session Profiles Session profiles allow prompt configurations, context, and artifacts to be defined.

Runtime Profiles Runtime profiles define the configuration to access the various services and resources in different environments (local, staging, production). This includes API endpoints, authentication, model configurations, and plugin settings.

Configuration: Design and Documentation

Anthesis configuration is designed to be hierarchical, environment-aware, and security-conscious. It supports multiple layers of configuration (project, user, environment) with clear precedence rules. Sensitive information is handled securely with support for secrets management and encryption. Configuration can be dynamically reloaded to reflect changes without restarting services. This design allows for flexible and secure management of Anthesis across different environments and use cases.

Configuration Layers

1. **Project Configuration** - `anthesis/config.yaml` (optional, version-controlled)
2. **User Configuration** - `~/.config/anthesis/config.yaml` (optional, user-specific)
3. **Environment Variables** - `ANTHESIS_*` (optional, overrides file configs)
4. **CLI Arguments** - `--config` and other flags (optional, highest precedence)

Security Considerations

- Sensitive information (API keys, credentials) should be stored in environment variables or secrets management systems, not in version-controlled files.
- Configuration files should have appropriate permissions to restrict access.
- Support for encrypted configuration values can be implemented for added security.

Dynamic Reloading

Anthesis services can be designed to watch configuration files for changes and reload them dynamically without requiring a restart. This allows for seamless updates to configuration in production environments.

Profiles & Overlays

Configuration profiles can be defined for different environments (local, staging, production) with specific settings. Overlays can be used to apply environment-specific overrides on top of base configurations. This allows for flexible management of configuration across different deployment targets.

Architecture: Decision Records and Design Notes

Architecture decision records, system design notes, and technical diagrams. This section captures the architectural decisions, design patterns, and system structure of Anthesis. It includes documentation on component interactions, data flows, integration patterns, and non-functional requirements such as performance, reliability, and security. The architecture documentation serves as a reference for developers and stakeholders to understand the technical design and rationale behind key decisions in the Anthesis system.

Scope

- ADRs and architectural trade-offs
- System diagrams and component boundaries
- Integration patterns and data flows

- Non-functional requirements (performance, reliability, security)

Conventions

- Use ADRs for decisions that affect long-term structure.
- Link to related RFCs and specs.
- Keep diagrams source-backed (Mermaid, draw.io, etc.).

Requirements: Defining the "What" of Anthesis

Anthesis requirements define the necessary conditions, capabilities, and constraints that must be met for the successful implementation and operation of the Anthesis system. These requirements encompass functional aspects such as agent behavior, workflow execution, plugin interactions, and session management, as well as non-functional aspects like performance, scalability, security, and maintainability. By establishing clear requirements, stakeholders can ensure that the development process is aligned with the intended goals and use cases of Anthesis, facilitating effective design decisions and prioritization throughout the project lifecycle.

Requirement Categories

1. **Functional Requirements** - Define specific behaviors, features, and interactions that the Anthesis system must support, such as agent capabilities, workflow orchestration, plugin functionality, and session management.
2. **Non-Functional Requirements** - Specify performance, scalability, security, reliability, and maintainability criteria that the Anthesis system must meet to ensure a robust and efficient operation.
3. **Governance Requirements** - Outline policies, rules, and constraints for agent behavior, workflow execution, and plugin interactions to ensure responsible and ethical use of AI capabilities.
4. **Integration Requirements** - Detail the necessary interfaces, protocols, and data formats for seamless integration between Anthesis components (MCP, Phloem, Xylem) and external systems or services.
5. **Testing & Validation Requirements** - Define the criteria and methodologies for testing the functionality, performance, and security of the Anthesis system to ensure it meets the specified requirements.
6. **Documentation Requirements** - Specify the necessary documentation for users, developers, and administrators to effectively utilize, extend, and maintain the Anthesis system.

Specifications: The Technical Blueprints of Anthesis

Anthesis specifications define the expected technical structure, behavior, and interfaces for various components of the Anthesis system. These specifications serve as a blueprint for development, ensuring consistency, interoperability, and adherence to design principles across different modules and plugins. Specifications cover areas such as configuration formats, plugin interfaces, API contracts, data models, and communication protocols. By following these specifications, developers can create compatible components that integrate seamlessly within the Anthesis ecosystem, facilitating extensibility and maintainability.

Specification Categories

1. **Governance Specifications** - Define policies, rules, and constraints for agent behavior, workflow execution, and plugin interactions.
2. **Plugin Specifications** - Outline the expected interfaces, lifecycle, and capabilities for plugins within the Anthesis system.
3. **Session Specifications** - Describe the structure and management of sessions, including context, prompts, and artifacts.
4. **Runtime Specifications** - Detail the expected behavior and interfaces for runtime components such as the MCP server and worker.
5. **API Specifications** - Define the endpoints, request/response formats, and authentication for Anthesis APIs.
6. **Data Model Specifications** - Describe the structure and relationships of data entities used across the system, such as agents, workflows, executions, and artifacts.
7. **Communication Protocol Specifications** - Outline the expected message formats, queues, and protocols for communication between components like Phloem and Xylem.

Plugins: Extending Capabilities with Governance

Plugins extend platform capabilities while preserving governance, policy enforcement, and auditability. They add agents, tools, integrations, and workflows without bypassing Calyx or expanding authority.

Philosophy

Plugins follow a core principle:

Plugins extend capability, never authority.

- Plugins cannot bypass Calyx policies
- Plugins cannot modify approval requirements
- Plugins cannot access privileged operations
- Plugins are **fully auditable** and **replayable**
- Plugins have a manifest and an execution model
- Plugins are signed with certificates auditable

Types

Anthesis supports four plugin categories, each with specific constraints and use cases:

- Agent plugins
- Tool plugins
- Integration plugins
- UI plugins

Document Publisher Plugin: Building Anthesis Documentation from Markdown

This plugin builds Anthesis documentation artifacts from Markdown sources. It converts Markdown files into HTML and PDF formats, integrating with the MCP for artifact storage and retrieval. The plugin is configurable, allowing users to specify source and output paths, as well as custom templates and styling for the generated documentation.

Git Plugin: Local Git Utilities for Anthesis

Local-only git utilities for Anthesis. Provides commit message generation, diff analysis, and integration with the MCP for commit and diff-based workflows. This plugin is designed to assist agents in understanding code changes, generating meaningful commit messages, and analyzing diffs to inform decision-making processes. It can be used in workflows that involve code review, automated commits, or any scenario where understanding code changes is beneficial.

Task Plugin: Local Task Management and Governance Tracking

Task plugin for intended for local task management and governance tracking. This plugin provides capabilities for creating, assigning, and tracking tasks within the Anthesis system, integrating with the MCP for task execution and notifications. It also includes features for governance tracking and reporting on task outcomes, ensuring that all tasks are executed in accordance with defined policies and that their results are properly documented and accessible for review.

CSSLP Analysis Summary

This document is a concise Anthesis-facing summary of the broader CSSLP security assessment. It is intended to orient Anthesis team members to the purpose, scope, and main themes of the CSSLP analysis, and to connect it to relevant RFCs and documentation.

It is explanatory only. Normative behavior remains defined by the Charter and the RFCs in `meristem/`.

Purpose

The CSSLP analysis is used to evaluate Anthesis across the secure software lifecycle, identify control gaps, and prioritize remediation work in a way that fits Anthesis governance rather than bypassing it.

Assessment Position

The current CSSLP assessment indicates:

- a strong baseline in governance, traceability, and auditability
- meaningful gaps in secrets management, dependency management, and validation hardening
- a need to keep security controls tied to existing Anthesis policy and evidence surfaces

How Anthesis Uses CSSLP

Anthesis uses CSSLP as:

- an analysis and review framework
- a source of security improvement priorities
- a way to organize lifecycle-wide security expectations

Anthesis does not use CSSLP as:

- a replacement for RFC governance
- a separate execution authority
- a standalone compliance engine

CSSLP Principles Applied to Anthesis

Anthesis applies CSSLP as a lifecycle-wide security discipline rather than as a single review step. In practical terms, that means:

- security requirements should be explicit, traceable, and tied to governed artifacts
- secure design decisions should be captured in RFCs and architecture records
- implementation controls should combine deterministic validation with governed review
- supply-chain integrity should be treated as part of the SDLC, not a separate concern
- testing and verification should produce attributable evidence rather than informal confidence

- deployment and operational transitions should preserve security posture and auditability
- vulnerability response should feed back into policy, design, and future change control

For Anthesis specifically, these principles map to:

- Charter and RFCs as the canonical design and security intent layer
- Calyx and Phloem as the security gate and policy decision layer
- Xylem, CI, and admitted tools as the validation and evidence-producing layer
- evidence export, provenance, and repository verification as the auditability layer
- human approval and governance review as the final authority layer

Main Themes From the Assessment

Secure Design

- Security should remain part of architectural intent, not a post-build review.
- Trust boundaries, threat models, and security architecture should be documented explicitly.

Secure Requirements

- Security requirements should be traceable to RFCs, implementations, and tests.
- Compliance mappings may be useful, but should remain secondary to concrete Anthesis controls.

Implementation and Validation

- Validation should combine deterministic checks with admitted agent review.
- Security-relevant changes should produce attributable evidence.
- Supply-chain visibility and artifact integrity need to remain first-class concerns.

Response and Improvement

- Vulnerability handling should lead to governed remediation work.
- Repeated security issues should feed back into policy, RFC, or design updates.

Relationship to Current RFCs

The CSSLP analysis aligns most directly with:

- `meristem/rfc_0003_approval_policy_engine.md`
- `meristem/rfc_0010_ci_cd_integration_model.md`
- `meristem/rfc_0023_audit_compliance_evidence_export.md`
- `meristem/rfc_0032_external_agent_tool_admission.md`
- `meristem/rfc_0044_gittuf_git_native_policy_verification_substrate.md`
- `meristem/rfc_0046_ssdf_integration_and_enforcement_model.md`

Summary

The CSSLP analysis is best treated as a structured security review lens over Anthesis, helping convert broad secure-lifecycle concerns into governed, traceable, evidence-backed improvements.

Governed SDLC Security Baseline

This document summarizes the current Anthesis security baseline for governed software delivery. It is explanatory only. The canonical normative sources remain the Charter and the RFCs under `meristem/`.

Purpose

Anthesis treats security as a governed property of the SDLC, not a separate after-the-fact review track. The security baseline is built from four layers:

- governance and approval controls
- execution and validation controls
- Git-native verification controls
- audit and evidence controls

Core Security Posture

- Humans retain final authority over policy, approvals, and trust changes.
- Agents may assist with review and validation, but do not become security authorities.
- Policy gates fail closed when required security evidence is missing.
- Security-relevant changes must remain attributable, reviewable, and exportable as evidence.

Current Control Surfaces

Governance and Approval

- Calyx defines policy and approval requirements.
- Phloem evaluates whether governed actions may proceed.
- Sessions, proposals, approvals, and evidence remain distinct control objects.
- Tracks are coordination-only artifacts and do not create execution permission.

Validation and Enforcement

- Xylem executes or coordinates scans, validation jobs, and supporting security workflows.
- Security agents may review implementation, dependency posture, and SBOM-related outputs.
- Deterministic checks remain authoritative over agent judgment.

Git-Native Verification

- gittuf is used as the Git-native policy and verification substrate for protected refs and protected paths.
- Anthesis governance remains authoritative for semantic decisions; gittuf verifies repository policy compliance.
- Approval bridging links Anthesis-approved decisions to repository verification artifacts.

Evidence and Audit

- Security decisions must be attributable to a human, service, or admitted tool identity.
- Evidence must remain append-only, contextual, and exportable.
- Verification, scan results, approval linkage, and policy decisions together form the reviewable security evidence chain.

Relevant RFCs

- `meristem/rfc_0003_approval_policy_engine.md`
- `meristem/rfc_0009_agent_execution_semantics.md`
- `meristem/rfc_0010_ci_cd_integration_model.md`
- `meristem/rfc_0020_state_machine_formal_specification.md`
- `meristem/rfc_0023_audit_compliance_evidence_export.md`
- `meristem/rfc_0032_external_agent_tool_admission.md`
- `meristem/rfc_0044_gittuf_git_native_policy_verification_substrate.md`
- `meristem/rfc_0045_tracks_as_governed_coordination_artifacts.md`
- `meristem/rfc_0046_ssdf_integration_and_enforcement_model.md`

SAFE-MCP: Structured Security Analysis for MCP-Based Systems in Anthesis

Anthesis adopts SAFE-MCP as a security analysis framework to structure threat modeling, control verification, and detection planning for MCP-based systems. This document describes how SAFE-MCP is used across Anthesis components to maintain a consistent security posture without altering governance or implementation.

Background and Rationale

Anthesis is an agentic SDLC platform with a multi-surface attack profile: MCP server endpoints, CLI clients, multi-agent coordination, plugins, and federated integrations. SAFE-MCP provides a standardized taxonomy of MCP-specific threats and mitigations. Anthesis uses this taxonomy to align security controls with clear evidence and to avoid ad hoc, inconsistent security reviews.

Scope of Application

SAFE-MCP is applied across the full stack:

- MCP server boundary (Phloem): authentication, authorization, tool exposure, and request validation.
- Client and CLI flows: prompt injection risks, unsafe tool invocation, and approval gates.
- Plugin ecosystem: tool poisoning and supply chain style risks in manifests and install paths.
- Policy and risk tiers: approval policy enforcement, least privilege, and bloom class constraints.
- Observability and audit: detection coverage, provenance logging, and audit trails.
- Federation and integrations: trust boundaries and cross-system access constraints.

Method of Use

Anthesis uses SAFE-MCP in three structured modes:

1. Threat Modeling: Map SAFE-MCP techniques to Anthesis surfaces, identify gaps, and document assumptions.
2. Control Verification: Validate that approvals, allowlists, and enforcement points exist and are auditable.
3. Test Design: Convert techniques into adversarial test cases for MCP routes, tool calls, and plugin workflows.

Outputs and Evidence

The primary artifact is the SAFE-MCP coverage matrix in `docs/security/safe-mcp-analysis.md`. It documents:

- Threat class to component mappings
- Expected controls and detections
- Evidence locations in the repository
- Status labels to track coverage and gaps

Governance Alignment

This summary is informational and does not modify RFCs, tasks, or runtime behavior. It is intended to support governance review by grounding security discussions in a common taxonomy and evidence trail.

Security Policies: Governance and Incident Response

Security Policy

- **Scope:** Applies to Anthesis services (Phloem, Xylem, CLI, plugins), all environments, personnel, and third-party integrations.
- **Principles:** Human sovereignty, defense-in-depth, least privilege, traceability, security by design, fail securely.
- **Core Requirements:** Authn/z for all API access, input validation, secrets management, data protection in transit/at rest, monitoring and alerting, secure SDLC practices, incident response adherence, and explicit third-party risk controls.
- **Roles:** Security team owns policy and incident response; engineering and DevOps implement controls; management approves changes.
- **Review Cadence:** Annual review or after P0/P1 incidents.

Incident Response Policy

- **Severity Levels:**
 - P0: Critical, immediate response (<15 minutes), all hands, executive notification.
 - P1: High, response within 1 hour, security lead + DevOps lead + on-call, management notified.
 - P2: Medium, response within 4 hours, on-call + security reviewer.
 - P3: Low, response within 24 hours, standard on-call.
- **Phases:** Detection & triage, containment, investigation, eradication, recovery, and post-incident review.
- **Key Actions:** Credential rotation, system isolation, evidence preservation, patching, staged recovery, and formal postmortem with owners and due dates.
- **Communication:** War room for P0/P1, regular status updates, decisions logged; coordinate external notices with Legal/Compliance as needed.

Related Anthesis Security Summaries

- Governed SDLC Security Baseline - Short orientation to Anthesis's current governed SDLC security posture.
- CSSLP Analysis Summary - Anthesis-facing summary of the broader CSSLP security assessment and how CSSLP principles apply to the platform.