Software Architecture Document (SAD)

Constitutional Build Pipeline for Sovereign AI Infrastructure

IEEE 42010:2011 / IEEE 1471-2000 Compliant

Document Information	
Document Title	Constitutional Build Pipeline SAD
Version	1.0
Date	February 6, 2026
Status	Draft for Review
Classification	Internal
Owner	Solutions Architect / Build Systems Lead

Document Control

Version	**Date**	**Author**	**Description of Changes**
0.1	2026-02-06	Architecture Team	Initial draft from workflow analysis
1.0	2026-02-06	Architecture Team	IEEE-compliant formatting

Reviewers

Role	Name	Signature	Date
Technical Lead			
Engineering Manager			
ML Lead			
DevOps Lead			
Quality Assurance Lead			

Document Dependencies

Document	Version	Relationship
Product Requirements Document (PRD)	1.0	Parent requirement source
System Architecture Document	1.0	System context definition
Routing Logic Specification	1.0	Prolog predicate definitions
Wiggum-Prolog-TDD Triad Protocol	1.0	Enforcement methodology
Model Serving & Orchestration Design	1.0	Model lifecycle management

---

Table of Contents

1. Introduction

1.1 Purpose

This Software Architecture Document (SAD) provides a comprehensive architectural description of the **Constitutional Build Pipeline** for the Sovereign AI Infrastructure system. The document conforms to IEEE 42010:2011 (Systems and Software Engineering — Architecture Description) and its predecessor IEEE 1471-2000.

The Constitutional Build Pipeline defines the methodology by which the Sovereign AI system autonomously constructs software packages through a governed, deterministic workflow that transforms creative ideation into validated, deployable code.

Intended Audience:
- Solutions Architects designing system extensions
- Software Engineers implementing pipeline components
- ML Engineers integrating new models
- Quality Assurance teams validating pipeline outputs
- Operations teams deploying and monitoring the system

1.2 Scope

This document describes the architecture of the Constitutional Build Pipeline, which encompasses:

In Scope:
- The nine-phase workflow for autonomous software construction
- Integration with the Bicameral Validator Ladder architecture
- The Wiggum-Prolog-TDD Triad enforcement mechanism
- Memory ledger interactions during build operations
- Test-first development methodology
- API-from-test derivation process

Out of Scope:
- Core Sovereign AI Infrastructure components (covered in System Architecture Document)
- Model training and fine-tuning workflows
- User interface design
- Deployment infrastructure (covered in Operations documentation)

1.3 Definitions, Acronyms, and Abbreviations

Term	Definition
Bicameral Architecture	Separation of creative generation (GPU) from critical validation (CPU), mimicking biological brain hemispheres
Constitutional Build Pipeline	The governed workflow transforming ideation through validated code assembly
Prolog Router	Declarative routing logic expressed as Prolog predicates that act as “constitutional law”
SAD	Software Architecture Document
SAD-lite	Abbreviated architecture document for individual software packages
Stakes	Risk/criticality assessment level (low, medium, high) determining validation rigor
TDD	Test-Driven Development
Validator	CPU-resident model that checks Worker outputs for correctness
VRAM	Video Random Access Memory (GPU memory)
Warm Pool	Models pre-loaded into system RAM for fast GPU loading
Wiggum Loop	Naive persistence execution loop that retries until tests pass
Worker	GPU-resident specialist model for content generation

1.4 References

ID	Document	Version	Date
REF-1	IEEE 42010:2011 Systems and Software Engineering — Architecture Description	-	2011
REF-2	Product Requirements Document (PRD)	1.0	2026-02-05
REF-3	System Architecture Document	1.0	2026-02-05
REF-4	Wiggum-Prolog-TDD Triad Protocol Specification	1.0	2026-02-05
REF-5	Routing Logic Specification (Prolog Predicates)	1.0	2026-02-05
REF-6	Model Serving & Orchestration Design	1.0	2026-02-05
REF-7	Data Architecture & Memory Design Document	1.0	2026-02-05
REF-8	Security and Governance Design	1.0	2026-02-05

1.5 Document Overview

This document is organized following IEEE 42010:2011 guidance:

- **Section 2** establishes the architectural representation, stakeholders, and viewpoints
- **Section 3** defines architectural goals, constraints, and key decisions
- **Section 4** provides the system context view
- **Section 5** presents the use case view with significant scenarios
- **Section 6** details the logical view of the pipeline phases and components
- **Section 7** describes the process view covering execution flows
- **Section 8** specifies the development view with package structures
- **Section 9** presents the physical/deployment view
- **Section 10** defines the data architecture view
- **Section 11** addresses quality attribute scenarios
- **Section 12** contains appendices with detailed specifications

2. Architectural Representation

2.1 Stakeholders and Concerns

Per IEEE 42010:2011, the following stakeholders and their concerns are identified:

Stakeholder	Concerns	Priority
Solutions Architect	Extensibility, modularity, pattern conformance, integration points	High
Software Engineer	Implementation clarity, debugging capability, API design	High
ML Engineer	Model integration, routing logic, validation policies	High
QA Engineer	Test coverage, traceability, validation accuracy	High
DevOps Engineer	Deployment, monitoring, failure recovery, resource utilization	Medium
Security Architect	Audit trail, governance enforcement, data integrity	High
Technical Lead	Quality governance, build reliability, schedule predictability	High
Compliance Officer	Audit completeness, traceability, policy enforcement	Medium

2.2 Architectural Viewpoints

This document employs the **4+1 View Model** (Kruchten) enhanced with additional viewpoints:

Viewpoint	Notation	Concerns Addressed
Use Case View	UML Use Case Diagrams, Scenarios	Functional requirements, stakeholder interactions
Logical View	UML Component/Class Diagrams, Flowcharts	Functional decomposition, responsibilities
Process View	UML Sequence/Activity Diagrams, State Machines	Concurrency, synchronization, execution flow
Development View	UML Package Diagrams, Directory Trees	Build organization, module dependencies
Physical View	UML Deployment Diagrams	Hardware mapping, process allocation
Data View	Data Flow Diagrams, Entity Schemas	Data structures, persistence, flow

2.3 Architectural Rationale

The Constitutional Build Pipeline architecture embodies three foundational principles:

Principle 1: Constitutional Governance over Agentic Reasoning

> "The Router is the Constitution; models are civil servants."

The system replaces unstructured "agentic reasoning" with deterministic constitutional enforcement. Prolog predicates define immutable laws that govern routing, validation, and execution.

Principle 2: Tests as Invariants, APIs as Derivatives

> "If it is not tested, it does not exist."

Test definitions precede API definitions. APIs are derived to satisfy tests, not the reverse. This prevents premature interface design and ensures all code is behaviorally specified.

Principle 3: Naive Persistence over Clever Optimization

> "Naive persistence beats clever optimization."

The Wiggum Loop employs brute-force retry logic rather than sophisticated planning. Combined with TDD constraints, this produces reliable convergence without complex state management.

---

3. Architectural Goals and Constraints

3.1 Goals

ID	Goal	Rationale	Priority
G1	**Deterministic Execution**	Build pipeline must produce identical outputs for identical inputs under identical conditions	Critical
G2	**Complete Auditability**	Every decision, transformation, and validation must be logged and traceable	Critical
G3	**Drift Prevention**	Software construction must not deviate from specifications through scope creep or alignment drift	Critical
G4	**Resource Efficiency**	Pipeline must minimize GPU swaps and VRAM thrashing	High
G5	**Failure Resilience**	Pipeline must recover gracefully from validation failures and resource constraints	High
G6	**Extensibility**	New phases, validators, or models must integrate without architectural changes	Medium

3.2 Constraints

ID	Constraint	Source	Impact
C1	Single GPU (16GB VRAM)	Hardware specification	One Worker model at a time
C2	Sequential task execution	v1.0 scope	No parallel builds
C3	CPU-resident validation	Bicameral architecture	Validator cannot use GPU
C4	Markdown memory format	Transparency requirement	Human-readable ledger
C5	Prolog-based routing	Governance requirement	Declarative rules only
C6	Maximum 3 retries per block	Resource protection	Escalation after failures

3.3 Key Architectural Decisions

ID	Decision	Rationale	Alternatives Considered	Status
AD-1	Tests before APIs	Tests encode contracts and invariants; APIs become interface layers satisfying tests	APIs-first (rejected: premature interface design)	Approved
AD-2	Flowchart as execution graph	Machine-readable flowchart defines dependency graph and microservice topology	Free-form architecture (rejected: non-deterministic)	Approved
AD-3	Phase-gate transitions	Each phase must complete before next begins	Parallel phases (rejected: complexity in v1.0)	Approved
AD-4	Constraints checkpoint	Explicit constraints capture before test generation provides validator invariants	Implicit constraints (rejected: insufficient governance)	Approved
AD-5	Post-integration validation	Final ratification pass ensures assembled package meets all requirements	Integration-only testing (rejected: insufficient coverage)	Approved

---

4. System Context View

4.1 System Boundaries

The Constitutional Build Pipeline operates within the Sovereign AI Infrastructure, interfacing with:

Internal Systems:
- Orchestration Layer (request coordination)
- Router Service (Prolog-based classification)
- Worker Models (content generation)
- Validator Model (correctness verification)

- Memory Manager (ledger operations)
- Model Loader (warm pool management)

External Actors:
- User (initiates build requests)
- Monitoring System (observes execution)
- Version Control (archives artifacts)

4.2 External Interfaces

Interface	Direction	Protocol	Description
Build Request	Inbound	CLI/REST API	User-initiated build request with requirements
Build Output	Outbound	Filesystem/API	Validated software package and audit trail
Memory Ledger	Bidirectional	Markdown/Filesystem	Persistent state and audit log
Metrics Export	Outbound	Prometheus	Pipeline telemetry
Git Commit	Outbound	Git Protocol	Artifact versioning

4.3 Context Diagram

```
┌─────────────────────────────────────────────────────────────────────────────┐
│                         SOVEREIGN AI INFRASTRUCTURE                          │
│  ┌───────────────────────────────────────────────────────────────────────┐   │
│  │                      CONSTITUTIONAL BUILD PIPELINE                     │  │
│  │                                                                        │  │
│  │  ┌─────────┐   ┌─────────┐   ┌─────────┐   ┌─────────┐   ┌─────────┐ │  │
│  │  │ Phase 1 │──▶│ Phase 2 │──▶│ Phase 3 │──▶│   ...   │──▶│ Phase 9 │ │  │
│  │  │Ideation │   │Arch     │   │Flowchart│   │         │   │Ratify   │ │  │
│  │  └─────────┘   └─────────┘   └─────────┘   └─────────┘   └─────────┘ │  │
│  │       │             │             │             │             │        │  │
│  │       └─────────────┴─────────────┴─────────────┴─────────────┘        │  │
│  │                               │                                        │  │
│  │                    ┌──────────▼──────────┐                            │  │
│  │                    │    MEMORY LEDGER    │                            │  │
│  │                    │  (Markdown Files)   │                            │  │
│  │                    └─────────────────────┘                            │  │
│  └───────────────────────────────────────────────────────────────────────┘  │
│                                  │                                           │
│        ┌─────────────────────────┼──────────────────────────┐               │
│        │                         │                          │               │
│        ▼                         ▼                          ▼               │
│  ┌─────────────┐          ┌─────────────┐          ┌─────────────┐         │
│  │   Router    │          │   Worker    │          │  Validator  │         │
│  │   (CPU)     │          │   (GPU)     │          │   (CPU)     │         │
│  │ Prolog Rules│          │ Generation  │          │ Verification│         │
│  └─────────────┘          └─────────────┘          └─────────────┘         │
└─────────────────────────────────────────────────────────────────────────────┘
          ▲                                                        │
          │                                                        │
          │ Build Request                           Build Output   │
          │                                         + Audit Trail  ▼
    ┌─────────┐                                              ┌─────────────┐
    │  USER   │                                              │    GIT      │
    └─────────┘                                              │  REPOSITORY │
                                                             └─────────────┘
```

5. Use Case View

5.1 Significant Use Cases

UC-1: Autonomous Software Package Construction

Primary Actor: Technical Lead / System
Preconditions: Requirements documented; models loaded; memory ledger initialized
Trigger: Build request submitted with package specification

Main Success Scenario:
1. System receives build request with requirements
2. Pipeline enters Phase 1 (Ideation) with creative exploration
3. Pipeline transitions through architecture, flowchart, microservice definition
4. Tests are generated from specifications (Phase 5)
5. APIs are derived from tests (Phase 6)
6. Each microservice is implemented via Wiggum Loop (Phase 7)
7. Components are assembled into complete package (Phase 8)

8. Post-integration validation ratifies the package (Phase 9)
9. Validated package and audit trail are delivered

Alternative Flows:
- **3a.** Architecture validation fails → Retry with corrections (max 3)
- **5a.** Test coverage insufficient → Expand test generation
- **7a.** Implementation fails validation → Wiggum Loop retries
- **7b.** Maximum retries exceeded → Escalate to user

Postconditions: Validated software package committed; audit trail complete

UC-2: Incremental Microservice Construction

Primary Actor: Wiggum Loop
Preconditions: Tests defined for microservice; API specified
Trigger: Phase 7 initiated for specific microservice

Main Success Scenario:
1. Wiggum Loop loads microservice specification and tests
2. Worker model generates implementation candidate
3. Validator checks implementation against constraints
4. Tests are executed against implementation
5. If tests pass: commit to project_state.md
6. If tests fail: capture error, generate correction, retry (Step 2)
7. Loop terminates on success or max retries

5.2 Use Case Realizations

UC-1 Realization: Phase Sequence

```
┌──────────────────────────────────────────────────────────────────────────┐
│                    CONSTITUTIONAL BUILD PIPELINE                          │
│                                                                           │
│  ┌─────────┐                                                              │
│  │    1    │  IDEATION                                                    │
│  │Creative │  • Freeform exploration                                      │
│  │ Worker  │  • Router → creative/architecture models                     │
│  └────┬────┘  • Validator mostly quiet                                    │
│       │                                                                   │
│       ▼                                                                   │
│  ┌─────────┐                                                              │
│  │    2    │  HIGH-LEVEL ARCHITECTURE + TECHNICAL DOCUMENTS               │
│  │  Arch   │  • Generate SAD-lite for package                             │
│  │ Worker  │  • Architecture diagrams, component lists, data flows        │
│  │    +    │  • Constraints, risks, assumptions                           │
│  │Validator│  • Provides global invariants for Validator/Router           │
│  └────┬────┘                                                              │
│       │                                                                   │
│       ▼                                                                   │
│  ┌─────────┐                                                              │
│  │    3    │  MACHINE-READABLE FLOWCHART                                  │
│  │ Router  │  • Execution graph                                           │
│  │    +    │  • Dependency graph                                          │
│  │ Prolog  │  • Microservice topology                                     │
│  └────┬────┘  • Validator ladder for this package                         │
│       │       • Transition: creative → constitutional                     │
│       ▼                                                                   │
│  ┌─────────┐                                                              │
│  │    4    │  MICROSERVICE/MODULE DEFINITION                              │
│  │  Arch   │  • Responsibilities, boundaries                              │
│  │ Worker  │  • Dependencies, failure modes                               │
│  └────┬────┘  • Validation policies, memory interactions                  │
│       │                                                                   │
│       ▼                                                                   │
│  ┌─────────┐                                                              │
│  │   4.1   │  CONSTRAINTS & INVARIANTS CHECKPOINT                         │
│  │Validator│  • Performance constraints                                   │
│  │    +    │  • Memory constraints                                        │
│  │ Prolog  │  • Validation policy (stakes)                                │
│  └────┬────┘  • Error semantics, security, grounding requirements         │
│       │       • "Constitutional clause" for each microservice             │
│       ▼                                                                   │
│  ┌─────────┐                                                              │
│  │    5    │  TEST DEFINITION (TDD)                                       │
│  │   TDD   │  • Tests encode the contract and invariants                  │
│  │ Worker  │  • Tests before implementation                               │
│  │    +    │  • Expected behavior, failure conditions, edge cases         │
│  │Validator│  • "Constitution of each microservice"                       │
│  └────┬────┘                                                              │
│       │                                                                   │
│       ▼                                                                   │
│  ┌─────────┐                                                              │
│  │    6    │  API DEFINITION                                              │
│  │  Arch   │  • APIs derived from tests (not vice versa)                  │
│  │ Worker  │  • Function signatures, endpoint schemas                     │
│  └────┬────┘  • Data models, error schemas, validation schemas            │
│       │                                                                   │
│       ▼                                                                   │
│  ┌─────────┐                                                              │
│  │    7    │  IMPLEMENTATION (WIGGUM LOOP)                                │
│  │ Worker  │  • Build each microservice individually                      │
│  │    +    │  • Generate → Test → Refine → Validate → Commit              │
│   │Wiggum  │  • One microservice at a time                              │
│   │   +     │  • No entanglement, no drift, no VRAM thrashing            │
│   │Validator│                                                           │
│   └────┬────┘                                                           │
│        │                                                                │
│        ▼                                                                │
│   ┌─────────┐                                                           │
│   │   8     │  ASSEMBLY                                                 │
│   │Orchestr │  • Integrate all validated microservices                  │
│   └────┬────┘  • Run integration tests                                  │
│        │                                                                │
│        ▼                                                                │
│   ┌─────────┐                                                           │
│   │   9     │  POST-INTEGRATION VALIDATION (RATIFICATION)               │
│   │Validator│  • Integration tests                                      │
│   │   +     │  • Performance tests                                      │
│   │ Prolog  │  • Grounding checks                                       │
│   └─────────┘  • Architectural consistency checks                       │
│                • Memory ledger integrity checks                         │
│                • Generate audit trail, finalize package                 │
│                                                                         │
└─────────────────────────────────────────────────────────────────────────┘
```

6. Logical View

6.1 Overview

The Constitutional Build Pipeline follows a linear phase-gate architecture where each phase produces artifacts consumed by subsequent phases. The design prioritizes:

1. **Exploratory → Architectural → Formal → Executable → Validated → Assembled** progression
2. **Tests-first** methodology ensuring behavioral specification before implementation
3. **Constitutional enforcement** through Prolog routing rules at each transition

6.2 Constitutional Build Pipeline Phases

Phase 1: Ideation (Creative Worker)

Purpose: Freeform exploration of the problem space
Actor: Creative Worker (MythoMax 13B or GPT-OSS 20B)
Governance: Router → creative/architecture models; Validator mostly quiet

Input	Process	Output
Problem statement	Creative exploration	Initial concepts
Domain context	Brainstorming	Design alternatives
User requirements	Feasibility analysis	Preliminary scope

Artifacts:
- `ideation_notes.md` — Exploratory thoughts and alternatives
- `scope_draft.md` — Initial scope boundaries

Phase 2: High-Level Architecture + Technical Documents

Purpose: Produce architectural blueprint and technical specification
Actor: Architecture Worker (Qwen Coder 32B) + Validator
Governance: Stakes-based validation; block validation for high-stakes

Input	Process	Output
Ideation artifacts	Architecture design	SAD-lite
PRD requirements	Component enumeration	Component list
System constraints	Data flow analysis	Data flow diagrams

Artifacts:
- `package_sad.md` — Package-specific architecture document
- `component_list.md` — Enumerated components with responsibilities
- `data_flow.md` — Data flow specifications
- `constraints.md` — Constraints, risks, assumptions

Phase 3: Machine-Readable Flowchart (Execution Graph)

Purpose: Define the execution topology as a formal graph
Actor: Router + Prolog
Governance: Constitutional enforcement; transition point creative → constitutional

Input	Process	Output
Architecture artifacts	Graph construction	Execution graph
Component list	Dependency analysis	Dependency graph
Data flows	Topology definition	Microservice topology

Artifacts:
- `execution_graph.json` — Machine-readable execution dependencies
- `dependency_matrix.md` — Component dependency matrix
- `validation_ladder.pl` — Prolog-encoded validation sequence

Key Transition: This phase marks the shift from exploratory to constitutional mode. The flowchart becomes the authoritative execution specification.

Phase 4: Microservice/Module Definition

Purpose: Define each component’s responsibilities and boundaries
Actor: Architecture Worker (Qwen Coder 32B)
Governance: Validator review for completeness

Input	Process	Output
Execution graph	Responsibility assignment	Microservice specs
Data flows	Boundary definition	Interface contracts
System constraints	Failure mode analysis	Error specifications

Artifacts (per microservice):
- `{service}_spec.md` — Responsibility, boundaries, dependencies
- `{service}_failures.md` — Failure modes and recovery
- `{service}_memory.md` — Memory/state interactions

Phase 4.1: Constraints & Invariants Checkpoint

Purpose: Capture constitutional clauses for each microservice
Actor: Validator + Prolog
Governance: Hard validation; no progression without approval

Input	Process	Output
Microservice specs	Constraint extraction	Constraint set
System requirements	Invariant definition	Invariant assertions
Validation policies	Stakes assignment	Policy binding

Artifacts:
- `{service}_constraints.yaml` — Performance, memory, security constraints
- `{service}_invariants.pl` — Prolog-encoded invariants
- `validation_policy_map.yaml` — Stakes-to-policy bindings

Phase 5: Test Definition (TDD Worker + Validator)

Purpose: Generate executable tests encoding contracts and invariants
Actor: TDD Worker + Validator
Governance: Validator must approve test coverage before progression

Input	Process	Output
Microservice specs	Contract extraction	Test assertions
Constraints/invariants	Behavioral encoding	Test scenarios
Failure modes	Edge case generation	Negative tests

Artifacts:
- `tests/test_{service}.py` — Executable pytest files
- `test_coverage_report.md` — Coverage analysis
- `test_rationale.md` — Mapping tests to requirements

Critical Insight: Tests encode:
- The **contract** (what must be true)
- The **invariants** (what must always be true)
- The **expected behavior** (happy path)
- The **failure conditions** (error handling)
- The **edge cases** (boundary conditions)

> "Tests are the constitution of each microservice."

Phase 6: API Definition (Architecture Worker)

Purpose: Define interfaces that satisfy the tests
Actor: Architecture Worker (Qwen Coder 32B)
Governance: APIs must be derivable from tests

Input	Process	Output
Test definitions	Interface extraction	Function signatures
Data requirements	Schema derivation	Data models
Error specifications	Error schema design	Error contracts

Artifacts:
- `{service}_api.py` — Interface definitions (stubs)
- `{service}_schemas.py` — Pydantic/dataclass models
- `{service}_errors.py` — Error type definitions
- `openapi_spec.yaml` — REST API specification (if applicable)

Ordering Rationale:

> "You don't define APIs first. You define tests first, because tests encode the contract. APIs become the interface layer that satisfies the tests."

Phase 7: Implementation (Worker + Wiggum + Validator)

Purpose: Build each microservice until tests pass
Actor: Worker Model + Wiggum Loop + Validator
Governance: Block-by-block validation; maximum 3 retries per block

Input	Process	Output
API definitions	Code generation	Implementation
Test files	Test execution	Validation result
Constraints	Compliance check	Validator verdict

Execution Model: Wiggum Loop

```
while !pytest -q tests/test_{service}.py; do
    ERROR_LOG=$(pytest tests/test_{service}.py)
    python orchestrator.py --refine --error "$ERROR_LOG"
    ATTEMPT=$((ATTEMPT+1))
    if [ $ATTEMPT -ge $MAX_RETRIES ]; then
        exit 1  # Escalate
    fi
done
```

Artifacts:
- `src/{service}/` — Implementation code
- `wiggum_log_{service}.md` — Iteration log
- `validation_trace_{service}.md` — Validator decisions

Phase 8: Assembly (Orchestrator)

Purpose: Integrate all validated microservices into complete package
Actor: Orchestrator
Governance: Integration test gate

Input	Process	Output
Validated components	Integration	Assembled package
Dependency graph	Linkage verification	Dependency report
Interface contracts	Contract validation	Integration status

Artifacts:
- `dist/{package}/` — Assembled package
- `integration_test_results.md` — Integration test output
- `assembly_manifest.json` — Component versions and checksums

Phase 9: Post-Integration Validation (Ratification)

Purpose: Final ratification of the complete system
Actor: Validator + Orchestrator
Governance: Comprehensive validation suite; no release without ratification

Input	Process	Output
Assembled package	Integration tests	Test results
Performance targets	Performance tests	Benchmarks
Grounding requirements	Grounding checks	Citation verification
Architecture spec	Consistency checks	Conformance report

Validation Suite:
1. **Integration Tests** — End-to-end functionality
2. **Performance Tests** — Latency, throughput, resource usage
3. **Grounding Checks** — Source citation verification (if multimodal)

4. **Architectural Consistency** — Design conformance
5. **Memory Ledger Integrity** — Audit trail completeness

Artifacts:
- `ratification_report.md` — Final validation summary
- `audit_trail.md` — Complete decision history
- `release_candidate.tar.gz` — Final package

6.3 Component Architecture

```
╔══════════════════════════════════════════════════════════════════════════╗
║                  CONSTITUTIONAL BUILD PIPELINE COMPONENTS                ║
╠══════════════════════════════════════════════════════════════════════════╣
║                                                                          ║
║  ┌────────────────────┐    ┌────────────────────┐                        ║
║  │   BUILD MANAGER    │───►│  PHASE CONTROLLER  │                        ║
║  │                    │    │                    │                        ║
║  │ • Request handling │    │ • Phase sequencing │                        ║
║  │ • Pipeline init    │    │ • Gate validation  │                        ║
║  │ • Status tracking  │    │ • Transition logic │                        ║
║  └────────────────────┘    └─────────┬──────────┘                        ║
║                                      │                                   ║
║                  ┌───────────────────┼─────────────────┐                 ║
║                  │                   │                 │                 ║
║                  ▼                   ▼                 ▼                 ║
║  ┌────────────────────┐  ┌────────────────────┐  ┌────────────────────┐  ║
║  │   IDEATION ENGINE  │  │ ARCHITECTURE ENGINE│  │  FLOWCHART ENGINE  │  ║
║  │                    │  │                    │  │                    │  ║
║  │ • Creative Worker  │  │ • Arch Worker      │  │ • Graph builder    │  ║
║  │ • Exploration      │  │ • SAD generation   │  │ • Prolog encoder   │  ║
║  │ • Scope drafting   │  │ • Validation       │  │ • Topology mapper  │  ║
║  └────────────────────┘  └────────────────────┘  └────────────────────┘  ║
║                  │                   │                 │                 ║
║                  ▼                   ▼                 ▼                 ║
║  ┌────────────────────┐  ┌────────────────────┐  ┌────────────────────┐  ║
║  │   SPEC GENERATOR   │  │   TEST GENERATOR   │  │   API GENERATOR    │  ║
║  │                    │  │                    │  │                    │  ║
║  │ • Module specs     │  │ • TDD Worker       │  │ • Interface design │  ║
║  │ • Constraints      │  │ • Test derivation  │  │ • Schema generation│  ║
║  │ • Invariants       │  │ • Coverage check   │  │ • Error contracts  │  ║
║  └────────────────────┘  └────────────────────┘  └────────────────────┘  ║
║                                      │                                   ║
║                                      ▼                                   ║
║                          ┌────────────────────┐                          ║
║                          │    WIGGUM ENGINE   │                          ║
║                          │                    │                          ║
║                          │ • Implementation   │                          ║
║                          │ • Test execution   │                          ║
║                          │ • Retry logic      │                          ║
║                          │ • Error capture    │                          ║
║                          └─────────┬──────────┘                          ║
║                                    │                                     ║
║                  ┌─────────────────┼─────────────────┐                   ║
║                  ▼                 ▼                 ▼                   ║
║  ┌────────────────────┐  ┌────────────────────┐  ┌────────────────────┐  ║
║  │  ASSEMBLY ENGINE   │  │ RATIFICATION ENGINE│  │   LEDGER MANAGER   │  ║
║  │                    │  │                    │  │                    │  ║
║  │ • Integration      │  │ • Final validation │  │ • Memory writes    │  ║
║  │ • Packaging        │  │ • Performance test │  │ • Audit logging    │  ║
║  │ • Manifest gen     │  │ • Ratification     │  │ • State management │  ║
║  └────────────────────┘  └────────────────────┘  └────────────────────┘  ║
║                                                                          ║
╚══════════════════════════════════════════════════════════════════════════╝
```

6.4 Component Interactions

Phase Transition Protocol

```python
class PhaseController:
    """Controls phase transitions with gate validation."""

    PHASES = [
        'ideation',
        'architecture',
        'flowchart',
        'module_definition',
        'constraints_checkpoint',
        'test_definition',
        'api_definition',
        'implementation',
        'assembly',
        'ratification'
    ]

    def transition(self, from_phase: str, to_phase: str) -> bool:
        """Execute phase transition with gate validation."""
        # Verify phase ordering
        if self.PHASES.index(to_phase) != self.PHASES.index(from_phase) + 1:
            raise InvalidTransitionError(f"Cannot jump from {from_phase} to
{to_phase}")

        # Validate gate conditions
        gate_result = self.validate_gate(from_phase)
        if not gate_result.passed:
            self.ledger.log_gate_failure(from_phase, gate_result.reason)
            return False

        # Execute transition
        self.ledger.log_phase_transition(from_phase, to_phase)
        self.current_phase = to_phase
        return True

    def validate_gate(self, phase: str) -> GateResult:
        """Validate phase completion criteria."""
        artifacts = self.get_required_artifacts(phase)
        for artifact in artifacts:
            if not artifact.exists():
                return GateResult(False, f"Missing artifact: {artifact.name}")
            if not artifact.is_valid():
                return GateResult(False, f"Invalid artifact: {artifact.name}")
        return GateResult(True, "All gate conditions satisfied")
```

7. Process View

7.1 Build Pipeline Execution Flow

```
┌─────────────────────────────────────────────────────────────────────────────┐
│                       PIPELINE EXECUTION FLOW                               │
│                                                                             │
│  ┌──────────┐                                                               │
│  │  START   │                                                               │
│  └────┬─────┘                                                               │
│       │                                                                     │
│       ▼                                                                     │
│  ┌──────────────────┐                                                       │
│  │ Initialize Build │                                                       │
│  │ - Load config    │                                                       │
│  │ - Init ledger    │                                                       │
│  │ - Load Router    │                                                       │
│  └────────┬─────────┘                                                       │
│           │                                                                 │
│           ▼                                                                 │
│  ┌──────────────────┐    ┌──────────────────┐                               │
│  │ Phase 1: Ideation│───▶│ Gate: Scope OK?  │──No──▶ Retry                  │
│  └──────────────────┘    └────────┬─────────┘                               │
│                                   │ Yes                                     │
│           ┌───────────────────────┘                                         │
│           ▼                                                                 │
│  ┌──────────────────┐    ┌──────────────────┐                               │
│  │ Phase 2: Arch    │───▶│ Gate: SAD Valid? │──No──▶ Retry                  │
│  └──────────────────┘    └────────┬─────────┘                               │
│                                   │ Yes                                     │
│           ┌───────────────────────┘                                         │
│           ▼                                                                 │
│  ┌──────────────────┐    ┌──────────────────┐                               │
│  │ Phase 3: Flowchart───▶│ Gate: Graph OK?  │──No──▶ Retry                  │
│  └──────────────────┘    └────────┬─────────┘                               │
│                                   │ Yes                                     │
│           ┌───────────────────────┘                                         │
│           ▼                                                                 │
│  ┌──────────────────┐                                                       │
│  │ Phase 4: Module  │                                                       │
│  │ Definition       │                                                       │
│  └────────┬─────────┘                                                       │
│           │                                                                 │
│           ▼                                                                 │
│  ┌──────────────────┐    ┌──────────────────┐                               │
│  │ Phase 4.1:       │───▶│ Gate: Invariants │──No──▶ Retry                  │
│  │ Constraints      │    │  Complete?       │                               │
│  └──────────────────┘    └────────┬─────────┘                               │
│                                   │ Yes                                     │
│           ┌───────────────────────┘                                         │
│           ▼                                                                 │
│  ┌──────────────────┐    ┌──────────────────┐                               │
│  │ Phase 5: Tests   │───▶│ Gate: Coverage   │──No──▶ Retry                  │
│  │ (TDD)            │    │  Sufficient?     │                               │
│  └──────────────────┘    └────────┬─────────┘                               │
│                                   │ Yes                                     │
│           ┌───────────────────────┘                                         │
│           ▼                                                                 │
│  ┌──────────────────┐    ┌──────────────────┐                               │
│  │ Phase 6: APIs    │───▶│ Gate: APIs Match │──No──▶ Retry                  │
│  │                  │    │  Tests?          │                               │
│  └──────────────────┘    └────────┬─────────┘                               │
│                                   │ Yes                                     │
│           ┌───────────────────────┘                                         │
│           ▼                                                                 │
│  ┌─────────────────────────────────────────┐                                │
│  │ Phase 7: Implementation (per service)   │                              │
│  │                                         │                              │
│  │  ┌──────────────────────────────────────┐ │                              │
│  │  │         WIGGUM LOOP                  │ │                              │
│  │  │                                      │ │                              │
│  │  │  ┌──────────┐                        │ │                              │
│  │  │  │Generate  │                        │ │                              │
│  │  │  │  Code    │                        │ │                              │
│  │  │  └────┬─────┘                        │ │                              │
│  │  │       │                              │ │                              │
│  │  │       ▼                              │ │                              │
│  │  │  ┌──────────┐                        │ │                              │
│  │  │  │Run Tests │                        │ │                              │
│  │  │  └────┬─────┘                        │ │                              │
│  │  │       │                              │ │                              │
│  │  │       ▼                              │ │                              │
│  │  │  ┌──────────┐  ┌───────┐             │ │                              │
│  │  │  │ Pass?    │──▶│Commit │──▶ Next     │ │                              │
│  │  │  └────┬─────┘  └───────┘   Service   │ │                              │
│  │  │       │ No                           │ │                              │
│  │  │       ▼                              │ │                              │
│  │  │  ┌──────────┐                        │ │                              │
│  │  │  │Retry<3?  │──No──▶ Escalate        │ │                              │
│  │  │  └────┬─────┘                        │ │                              │
│  │  │       │ Yes                          │ │                              │
│  │  │       └──────────────┐               │ │                              │
│  │  │                      │               │ │                              │
│  │  │  ┌──────────┐        │               │ │                              │
│  │  │  │ Refine   │◀───────┘               │ │                              │
│  │  │  │(w/error) │                        │ │                              │
│  │  │  └────┬─────┘                        │ │                              │
│  │  │       │                              │ │                              │
│  │  │       └───────▶ (back to Generate)   │ │                              │
│  │  └──────────────────────────────────────┘ │                              │
│  └──────────────────────────────────────────┘                              │
│         │                                                                  │
│         ▼                                                                  │
│  ┌───────────────────┐   ┌───────────────────┐                             │
│  │ Phase 8: Assembly │──▶│ Gate: Integration │──No──▶ Retry                │
│  │                   │   │ Tests Pass?       │                             │
│  └───────────────────┘   └─────────┬─────────┘                             │
│                                    │ Yes                                   │
│         ┌──────────────────────────┘                                       │
│         ▼                                                                  │
│  ┌───────────────────┐   ┌───────────────────┐                             │
│  │ Phase 9:          │──▶│ Ratification      │──No──▶ Retry/Escalate       │
│  │ Ratification      │   │ Passed?           │                             │
│  └───────────────────┘   └─────────┬─────────┘                             │
│                                    │ Yes                                   │
│         ┌──────────────────────────┘                                       │
│         ▼                                                                  │
│  ┌───────────────────┐                                                     │
│  │ Finalize Package  │                                                     │
│  │ - Generate audit  │                                                     │
│  │ - Commit to Git   │                                                     │
│  │ - Output package  │                                                     │
│  └─────────┬─────────┘                                                     │
│            │                                                               │
│            ▼                                                               │
│  ┌───────────┐                                                             │
│  │   END     │                                                             │
│  └───────────┘                                                             │
│                                                                              │
└──────────────────────────────────────────────────────────────────────────────┘
```

7.2 Concurrency and Synchronization

v1.0 Constraint: Sequential execution only

Aspect	Design Decision	Rationale
Phase execution	Sequential	Simplifies state management
Microservice build	Sequential (one at a time)	Single GPU constraint
Validation	Sequential with generation	CPU/GPU bicameral separa-tion
Memory writes	Atomic with file locking	Prevents corruption

Future (v2.0): Parallel microservice builds with multi-GPU support

7.3 State Machine Specifications

Build Pipeline State Machine

```
States: {IDLE, BUILDING, GATE_CHECK, RETRY, ESCALATED, COMPLETED, FAILED}

Transitions:
  IDLE        --[build_request]-->     BUILDING
  BUILDING    --[phase_complete]-->    GATE_CHECK
  GATE_CHECK  --[gate_passed]-->       BUILDING (next phase)
  GATE_CHECK  --[gate_failed]-->       RETRY
  RETRY       --[retry_count<max]-->   BUILDING (same phase)
  RETRY       --[retry_count>=max]-->  ESCALATED
  ESCALATED   --[user_resolution]-->   BUILDING
  ESCALATED   --[abort]-->             FAILED
  BUILDING    --[all_phases_done]-->   COMPLETED
```

Wiggum Loop State Machine

```
States: {INIT, GENERATING, TESTING, VALIDATING, REFINING, COMMITTED, ESCALATED}

Transitions:
  INIT        --[start]-->             GENERATING
  GENERATING  --[code_generated]-->    TESTING
  TESTING     --[tests_passed]-->      VALIDATING
  TESTING     --[tests_failed]-->      REFINING
  VALIDATING  --[validator_pass]-->    COMMITTED
  VALIDATING  --[validator_fail]-->    REFINING
  REFINING    --[retry_count<3]-->     GENERATING
  REFINING    --[retry_count>=3]-->    ESCALATED
  COMMITTED   --[more_services]-->     INIT (next service)
  COMMITTED   --[all_done]-->          EXIT_SUCCESS
  ESCALATED   --[user_fix]-->          GENERATING
  ESCALATED   --[abort]-->             EXIT_FAILURE
```

8. Development View

8.1 Package Structure

```
sovereign-ai/
├── src/
│   ├── build_pipeline/                     # Constitutional Build Pipeline
│   │   ├── __init__.py
│   │   ├── manager.py                      # BuildManager entry point
│   │   ├── phase_controller.py             # Phase sequencing and gates
│   │   ├── phases/                         # Phase implementations
│   │   │   ├── __init__.py
│   │   │   ├── ideation.py                 # Phase 1: Ideation Engine
│   │   │   ├── architecture.py             # Phase 2: Architecture Engine
│   │   │   ├── flowchart.py                # Phase 3: Flowchart Engine
│   │   │   ├── module_definition.py        # Phase 4: Spec Generator
│   │   │   ├── constraints.py              # Phase 4.1: Constraints Checkpoint
│   │   │   ├── test_generation.py          # Phase 5: Test Generator
│   │   │   ├── api_generation.py           # Phase 6: API Generator
│   │   │   ├── implementation.py           # Phase 7: Implementation Manager
│   │   │   ├── assembly.py                 # Phase 8: Assembly Engine
│   │   │   └── ratification.py             # Phase 9: Ratification Engine
│   │   ├── wiggum/                         # Wiggum Loop implementation
│   │   │   ├── __init__.py
│   │   │   ├── loop.py                     # Main loop logic
│   │   │   ├── error_capture.py            # Error parsing and logging
│   │   │   └── refinement.py               # Correction generation
│   │   ├── gates/                          # Gate validation logic
│   │   │   ├── __init__.py
│   │   │   ├── base_gate.py                # Abstract gate interface
│   │   │   ├── artifact_gate.py            # Artifact existence/validity
│   │   │   ├── coverage_gate.py            # Test coverage validation
│   │   │   └── integration_gate.py         # Integration test gate
│   │   └── artifacts/                      # Artifact management
│   │       ├── __init__.py
│   │       ├── artifact_manager.py         # Artifact CRUD operations
│   │       └── templates/                  # Artifact templates
│   │           ├── sad_lite.md.j2
│   │           ├── test_template.py.j2
│   │           └── api_template.py.j2
│   ├── orchestrator/                       # (existing)
│   ├── router/                             # (existing)
│   ├── memory/                             # (existing)
│   ├── models/                             # (existing)
│   ├── tools/                              # (existing)
│   └── validator/                          # (existing)
├── config/
│   ├── build_pipeline.yaml                 # Pipeline configuration
│   ├── phase_gates.yaml                    # Gate criteria definitions
│   ├── routing_rules.pl                    # (existing) + build phase rules
│   └── wiggum_config.yaml                  # Wiggum loop settings
├── templates/                              # Build artifact templates
│   ├── flowchart/
│   ├── tests/
│   └── api/
├── tests/
│   └── build_pipeline/
│       ├── test_phase_controller.py
│       ├── test_wiggum_loop.py
│       └── test_gates.py
└── scripts/
    ├── wiggum_loop.sh                      # Shell-based Wiggum loop
    └── build_package.py                    # CLI for build pipeline
```

8.2 Build Dependencies

```
                    ┌────────────────┐
                    │ build_pipeline │
                    │    manager     │
                    └───────┬────────┘
                            │
              ┌─────────────┼─────────────┐
              │             │             │
              ▼             ▼             ▼
       ┌────────────┐ ┌────────────┐ ┌────────────┐
       │  phases/   │ │  wiggum/   │ │   gates/   │
       └─────┬──────┘ └──────┬─────┘ └──────┬─────┘
             │              │              │
             └──────────────┼──────────────┘
                            │
              ┌─────────────┼─────────────┐
              │             │             │
              ▼             ▼             ▼
       ┌────────────┐ ┌────────────┐ ┌────────────┐
       │orchestrator│ │   router   │ │   memory   │
       └────────────┘ │  (Prolog)  │ │  (ledger)  │
                      └────────────┘ └────────────┘
                            │
                            ▼
                    ┌──────────────┐
                    │  validator   │
                    └──────────────┘
```

8.3 Configuration Management

build_pipeline.yaml

```yaml
# Constitutional Build Pipeline Configuration
version: "1.0"

phases:
  ideation:
    enabled: true
    worker_model: gpt_oss_20b
    max_iterations: 3
    timeout_seconds: 300

  architecture:
    enabled: true
    worker_model: qwen_coder_32b
    validation_required: true
    validation_policy: end_stage
    artifacts:
      - package_sad.md
      - component_list.md
      - data_flow.md

  flowchart:
    enabled: true
    prolog_encoding: true
    artifacts:
      - execution_graph.json
      - dependency_matrix.md
      - validation_ladder.pl

  module_definition:
    enabled: true
    worker_model: qwen_coder_32b
    per_service_artifacts:
      - "{service}_spec.md"
      - "{service}_failures.md"

  constraints_checkpoint:
    enabled: true
    hard_gate: true  # Cannot proceed without pass
    artifacts:
      - "{service}_constraints.yaml"
      - "{service}_invariants.pl"

  test_definition:
    enabled: true
    worker_model: qwen_coder_32b
    min_coverage: 80
    validation_required: true
    artifacts:
      - "tests/test_{service}.py"

  api_definition:
    enabled: true
    worker_model: qwen_coder_32b
    derive_from_tests: true
    artifacts:
      - "{service}_api.py"
      - "{service}_schemas.py"

  implementation:
    enabled: true
    worker_model: nemotron_30b  # Implementation specialist
    wiggum_max_retries: 3
    validation_policy: block_by_block

  assembly:
    enabled: true
    integration_tests_required: true

  ratification:
    enabled: true
    validation_suite:
      - integration_tests
      - performance_tests
      - grounding_checks
      - architectural_consistency
      - ledger_integrity

wiggum:
  max_retries: 3
  error_log_format: detailed
  escalation_action: user_prompt
  commit_on_success: true

gates:
  default_retry_limit: 3
  artifact_validation: strict
  coverage_threshold: 0.80
```

9. Physical View

9.1 Deployment Architecture

The Constitutional Build Pipeline runs within the existing Sovereign AI Infrastructure deployment:

```
┌──────────────────────────────────────────────────────────────────────┐
│                         SINGLE WORKSTATION                           │
│                                                                      │
│  ┌────────────────────────────────────────────────────────────────┐  │
│  │                   BUILD PIPELINE PROCESSES                     │  │
│  │                                                                │  │
│  │  ┌────────────────┐  ┌────────────────┐  ┌────────────────┐    │  │
│  │  │ build-manager  │  │ phase-executor │  │ wiggum-loop    │    │  │
│  │  │ (Python daemon)│  │ (Python)       │  │ (Shell/Python) │    │  │
│  │  └───────┬────────┘  └───────┬────────┘  └───────┬────────┘    │  │
│  │          │                   │                   │             │  │
│  │          └───────────────────┼───────────────────┘             │  │
│  │                              │                                 │  │
│  │                              ▼                                 │  │
│  │  ┌──────────────────────────────────────────────────────────┐  │  │
│  │  │                   ORCHESTRATION LAYER                    │  │  │
│  │  │  (sovereign-orchestrator daemon)                         │  │  │
│  │  └──────────────────────────────────────────────────────────┘  │  │
│  │                              │                                 │  │
│  └──────────────────────────────┼─────────────────────────────────┘  │
│                                 │                                    │
│  ┌──────────────────────────────┼─────────────────────────────────┐  │
│  │                  INFERENCE LAYER                               │  │
│  │                                                                │  │
│  │ CPU                                       GPU                  │  │
│  │  ┌────────────────┐  ┌───────────────┐  ┌────────────────┐     │  │
│  │  │ llama-router   │  │ llama-validator│ │ llama-worker   │     │  │
│  │  │ :8001          │  │ :8002         │  │ :8000          │     │  │
│  │  │ (Granite 3B)   │  │ (Granite-H)   │  │ (Qwen/Nemotron)│     │  │
│  │  └────────────────┘  └───────────────┘  └────────────────┘     │  │
│  │                                                                │  │
│  └────────────────────────────────────────────────────────────────┘  │
│                                 │                                    │
│  ┌──────────────────────────────┼─────────────────────────────────┐  │
│  │                  STORAGE LAYER                                 │  │
│  │                                                                │  │
│  │  ┌────────────────┐  ┌───────────────┐  ┌────────────────┐     │  │
│  │  │  Memory Ledger │  │  Build        │  │  Model Vault   │     │  │
│  │  │  (Markdown)    │  │  Artifacts    │  │  (GGUF files)  │     │  │
│  │  │                │  │  (.py, .yaml) │  │                │     │  │
│  │  └────────────────┘  └───────────────┘  └────────────────┘     │  │
│  │                                                                │  │
│  └────────────────────────────────────────────────────────────────┘  │
│                                                                      │
└──────────────────────────────────────────────────────────────────────┘
```

9.2 Hardware Mapping

Component	Hardware	Memory	Purpose
Build Manager	CPU	1GB	Pipeline coordination
Phase Executor	CPU	2GB	Phase execution logic
Wiggum Loop	CPU	512MB	Test/retry loop
Router (Prolog)	CPU	6GB	Phase routing
Validator	CPU	20GB	Artifact validation
Worker	GPU	12-18GB VRAM	Code generation

9.3 Process Allocation

Process	Type	Lifecycle	Resources
`build-manager`	Daemon	Long-running	1 CPU core, 1GB RAM
`phase-executor`	Spawned	Per-phase	1 CPU core, 2GB RAM
`wiggum-loop`	Spawned	Per-service	1 CPU core, 512MB RAM
`pytest-runner`	Spawned	Per-test	1 CPU core, 512MB RAM

---

10. Data View

10.1 Data Flow Architecture

```
┌──────────────────────────────────────────────────────────────────────────────┐
│                         BUILD PIPELINE DATA FLOW                             │
│                                                                              │
│   ┌─────────┐                                                                │
│   │ User    │                                                                │
│   │ Request │                                                                │
│   └────┬────┘                                                                │
│        │                                                                     │
│        ▼                                                                     │
│   ┌─────────────┐    ┌─────────────┐    ┌─────────────┐                      │
│   │  Phase 1    │───▶│ Ideation    │───▶│ scope_draft │                      │
│   │  Input      │    │ Engine      │    │  .md        │                      │
│   └─────────────┘    └─────────────┘    └──────┬──────┘                      │
│                                                │                             │
│                                                ▼                             │
│                     ┌─────────────┐    ┌─────────────┐                       │
│                     │ Architecture│◀───│ scope_draft │                       │
│                     │ Engine      │    │  + PRD      │                       │
│                     └──────┬──────┘    └─────────────┘                       │
│                            │                                                 │
│        ┌───────────────────┼────────────────────┐                            │
│        │                   │                    │                            │
│        ▼                   ▼                    ▼                            │
│   ┌─────────┐        ┌─────────┐          ┌─────────┐                        │
│   │SAD-lite │        │component│          │data_flow│                        │
│   │.md      │        │_list.md │          │.md      │                        │
│   └────┬────┘        └────┬────┘          └────┬────┘                        │
│        │                  │                    │                             │
│        └──────────────────┼────────────────────┘                             │
│                           │                                                  │
│                           ▼                                                  │
│                     ┌─────────────┐    ┌─────────────┐                       │
│                     │ Flowchart   │───▶│ execution   │                       │
│                     │ Engine      │    │ _graph.json │                       │
│                     └─────────────┘    └──────┬──────┘                       │
│                                               │                              │
│                          ┌────────────────────┘                              │
│                          │                                                   │
│   ┌──────────────────────┴─────────────────────────────────┐                 │
│   │         FOR EACH MICROSERVICE IN execution_graph       │                 │
│   │                        │                               │                 │
│   │                        ▼                               │                 │
│   │  ┌─────────────┐  ┌─────────────┐  ┌─────────────┐     │                 │
│   │  │{svc}_spec.md│─▶│{svc}_const- │─▶│tests/test_  │     │                 │
│   │  │             │  │raints.yaml  │  │{svc}.py     │     │                 │
│   │  └─────────────┘  └─────────────┘  └──────┬──────┘     │                 │
│   │                                           │            │                 │
│   │                                           ▼            │                 │
│   │                   ┌─────────────┐  ┌─────────────┐     │                 │
│   │                   │{svc}_api.py │◀─│ Test-first  │     │                 │
│   │                   │             │  │ derivation  │     │                 │
│   │                   └──────┬──────┘  └─────────────┘     │                 │
│   │                          │                             │                 │
│   │                          ▼                             │                 │
│   │  ┌────────────────────────────────────────────────┐    │                 │
│   │  │                   WIGGUM LOOP                  │    │                 │
│   │  │                                                │    │                 │
│   │  │  {svc}_api.py ──▶ Worker ──▶ src/{svc}/*.py    │    │                 │
│   │  │       +                             │          │    │                 │
│   │  │  constraints      pytest ◀──────────┘          │    │                 │
│   │  │       +             │                          │    │                 │
│   │  │  tests            Validator ◀─────────┘        │    │                 │
│   │  │                        │                                 │    │          │
│   │  │                        ▼                                 │    │          │
│   │  │                  [PASS] ──▶ COMMIT to project_state.md   │    │          │
│   │  │                  [FAIL] ──▶ RETRY with correction        │    │          │
│   │  └────────────────────────────────────────────────────────┘    │          │
│   │                                                                 │          │
│   └─────────────────────────────────────────────────────────────────┘          │
│                               │                                                │
│                               ▼                                                │
│                        ┌─────────────┐                                         │
│                        │  Assembly   │                                         │
│                        │  Engine     │                                         │
│                        └──────┬──────┘                                         │
│                               │                                                │
│                               ▼                                                │
│                        ┌─────────────┐     ┌─────────────┐                     │
│                        │Ratification │────▶│  Validated  │                     │
│                        │  Engine     │     │  Package    │                     │
│                        └─────────────┘     │  + Audit    │                     │
│                                            └─────────────┘                     │
│                                                                                │
└────────────────────────────────────────────────────────────────────────────────┘
```

10.2 Memory Ledger Schema

The build pipeline extends the memory ledger with build-specific sections:

project_state.md (Build Section)

```
# Project State

## Build Pipeline State
### Current Build
- **Build ID**: build_20260206_001
- **Status**: Phase 7 - Implementation
- **Current Service**: payment_processor
- **Progress**: 3/5 services complete

### Committed Components
#### payment_validator (Committed: 2026-02-06T10:30:00Z)
- Spec: payment_validator_spec.md
- Tests: tests/test_payment_validator.py (PASS)
- Implementation: src/payment_validator/
- Validation: [PASS] All checks passed

#### transaction_logger (Committed: 2026-02-06T11:15:00Z)
- Spec: transaction_logger_spec.md
- Tests: tests/test_transaction_logger.py (PASS)
- Implementation: src/transaction_logger/
- Validation: [PASS] All checks passed
```

scratchpad.md (Build Section)

```
# Scratchpad

## Build Session: 2026-02-06

### Phase 7: Implementation - payment_processor
#### Wiggum Iteration 1 (2026-02-06T12:00:00Z)
**Actor**: Worker (nemotron_30b)
**Action**: Generate implementation
**Output**: Initial implementation generated

#### Wiggum Iteration 1 - Test (2026-02-06T12:00:30Z)
**Actor**: pytest
**Result**: FAIL
**Error**:
```

test_payment_processor.py::test_validate_amount FAILED
AssertionError: Expected ValidationError for negative amount

```
#### Wiggum Iteration 1 - Validator (2026-02-06T12:01:00Z)
**Actor**: Validator (Granite-H)
**Verdict**: [FAIL: Missing negative amount validation]
**Correction**: Add check for amount < 0 at line 45

#### Wiggum Iteration 2 (2026-02-06T12:02:00Z)
**Actor**: Worker (nemotron_30b)
**Action**: Refine with correction
**Context**: Add negative amount validation
```

10.3 Artifact Specifications

Execution Graph Schema (execution_graph.json)

```json
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Execution Graph",
  "type": "object",
  "properties": {
    "version": { "type": "string" },
    "package_name": { "type": "string" },
    "created_at": { "type": "string", "format": "date-time" },
    "nodes": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "id": { "type": "string" },
          "name": { "type": "string" },
          "type": { "enum": ["service", "library", "interface"] },
          "responsibilities": { "type": "array", "items": { "type": "string" } },
          "dependencies": { "type": "array", "items": { "type": "string" } }
        },
        "required": ["id", "name", "type"]
      }
    },
    "edges": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "from": { "type": "string" },
          "to": { "type": "string" },
          "type": { "enum": ["depends_on", "calls", "data_flow"] }
        },
        "required": ["from", "to", "type"]
      }
    }
  },
  "required": ["version", "package_name", "nodes", "edges"]
}
```

Constraints Schema ({service}_constraints.yaml)

```yaml
# Service Constraints Schema
service_name: payment_processor
version: "1.0"

performance:
  max_latency_ms: 100
  min_throughput_rps: 1000
  timeout_seconds: 30

memory:
  max_heap_mb: 512
  allow_caching: true
  cache_ttl_seconds: 300

validation:
  stakes: high
  policy: block_by_block
  max_retries: 3

error_semantics:
  on_invalid_input: raise_validation_error
  on_timeout: retry_with_backoff
  on_dependency_failure: circuit_breaker

security:
  input_sanitization: required
  output_encoding: required
  audit_logging: required
  pii_handling: redact

grounding:
  required: false
  source_citation: not_applicable
```

---

11. Quality Attributes

11.1 Performance

Metric	Target	Measurement	Rationale
Phase transition time	≤5 seconds	Gate validation + artifact write	Minimize overhead
Wiggum iteration time	≤60 seconds	Generate + test + validate	Acceptable for block
Full build (5 services)	≤30 minutes	End-to-end	Reasonable for complex package
Retry overhead	≤20% of iteration	Error capture + context	Efficient retry

11.2 Reliability

Attribute	Requirement	Mechanism
Build completion rate	≥95%	Retry logic, escalation
Artifact integrity	100%	Checksums, atomic writes
State recovery	From last committed block	Ledger persistence
Retry convergence	≥90% within 3 attempts	TDD + clear corrections

11.3 Security

Attribute	Requirement	Mechanism
Audit completeness	100% of decisions logged	Immutable ledger
Tamper detection	Checksums on artifacts	SHA-256 verification
Constraint enforcement	No bypass of Prolog rules	Hard-coded gate checks
Escalation logging	All escalations recorded	Audit trail entry

11.4 Maintainability

Attribute	Requirement	Mechanism
Phase modularity	Add phase without core changes	Phase plugin architecture
Rule extensibility	Add rules without code changes	Prolog knowledge base
Template customization	Modify artifacts via templates	Jinja2 templates
Configuration-driven	Behavior via YAML, not code	Externalized config

11.5 Traceability

Requirement	Artifact	Verification
Every test traces to require-ment	test_rationale.md	Review
Every API traces to test	API comments with test IDs	Automated check
Every implementation traces to API	Code comments	Review
Every decision traces to audit	scratchpad.md entries	Log analysis

---

12. Appendices

Appendix A: Prolog Encoding of Build Pipeline

```prolog
% ============================================
% Constitutional Build Pipeline - Prolog Rules
% File: build_pipeline_rules.pl
% Version: 1.0
% ============================================

% Build phases in order
build_phase(ideation).
build_phase(architecture).
build_phase(flowchart).
build_phase(module_definition).
build_phase(constraints_checkpoint).
build_phase(test_definition).
build_phase(api_definition).
build_phase(implementation).
build_phase(assembly).
build_phase(ratification).

% Phase ordering
phase_order(ideation, 1).
phase_order(architecture, 2).
phase_order(flowchart, 3).
phase_order(module_definition, 4).
phase_order(constraints_checkpoint, 5).
phase_order(test_definition, 6).
phase_order(api_definition, 7).
phase_order(implementation, 8).
phase_order(assembly, 9).
phase_order(ratification, 10).

% Valid phase transition
valid_transition(From, To) :-
    phase_order(From, N1),
    phase_order(To, N2),
    N2 is N1 + 1.

% Phase requires Worker model
phase_requires_worker(ideation, gpt_oss_20b).
phase_requires_worker(architecture, qwen_coder_32b).
phase_requires_worker(module_definition, qwen_coder_32b).
phase_requires_worker(test_definition, qwen_coder_32b).
phase_requires_worker(api_definition, qwen_coder_32b).
phase_requires_worker(implementation, nemotron_30b).

% Phase requires Validator
phase_requires_validator(architecture).
phase_requires_validator(constraints_checkpoint).
phase_requires_validator(test_definition).
phase_requires_validator(implementation).
phase_requires_validator(ratification).

% Gate conditions
gate_condition(ideation, scope_defined).
gate_condition(architecture, sad_valid).
gate_condition(flowchart, graph_complete).
gate_condition(module_definition, specs_complete).
gate_condition(constraints_checkpoint, invariants_captured).
gate_condition(test_definition, coverage_sufficient).
gate_condition(api_definition, apis_match_tests).
gate_condition(implementation, all_tests_pass).
gate_condition(assembly, integration_passes).
gate_condition(ratification, ratification_complete).
```

```
% Route to appropriate model for phase
route_phase(Phase, Model) :-
    phase_requires_worker(Phase, Model).

% Determine validation policy for phase
validation_policy_for_phase(implementation, block_by_block).
validation_policy_for_phase(architecture, end_stage).
validation_policy_for_phase(_, none).

% Can proceed to next phase?
can_proceed(CurrentPhase, NextPhase) :-
    valid_transition(CurrentPhase, NextPhase),
    gate_condition(CurrentPhase, Condition),
    gate_satisfied(Condition).

% Wiggum retry logic
wiggum_should_retry(AttemptCount, MaxRetries) :-
    AttemptCount < MaxRetries.

wiggum_should_escalate(AttemptCount, MaxRetries) :-
    AttemptCount >= MaxRetries.
```

Appendix B: TDD Template Specification

```
# Template: test_template.py.j2
# Purpose: Generate test files from microservice specifications

"""
Tests for {{ service_name }} microservice.

Generated by Constitutional Build Pipeline
Date: {{ generation_date }}
Spec: {{ spec_reference }}
"""

import pytest
from typing import Any
{% if has_async %}
import asyncio
{% endif %}

# Import the module under test (will be implemented in Phase 7)
# from src.{{ service_name }} import {{ main_class }}


class Test{{ service_name | title }}:
    """Test suite for {{ service_name }}."""

    # ==========================================
    # HAPPY PATH TESTS
    # ==========================================

    {% for test in happy_path_tests %}
    def test_{{ test.name }}(self):
        """
        {{ test.description }}

        Requirement: {{ test.requirement_id }}
        Invariant: {{ test.invariant }}
        """
        # Arrange
        {{ test.arrange | indent(8) }}

        # Act
        {{ test.act | indent(8) }}

        # Assert
        {{ test.assertions | indent(8) }}

    {% endfor %}

    # ==========================================
    # ERROR HANDLING TESTS
    # ==========================================

    {% for test in error_tests %}
    def test_{{ test.name }}_raises_{{ test.expected_error }}(self):
        """
        {{ test.description }}

        Requirement: {{ test.requirement_id }}
        Error Condition: {{ test.error_condition }}
        """
        # Arrange
        {{ test.arrange | indent(8) }}
        # Act & Assert
        with pytest.raises({{ test.expected_error }}):
            {{ test.act | indent(12) }}

    {% endfor %}

    # ==========================================
    # EDGE CASE TESTS
    # ==========================================

    {% for test in edge_case_tests %}
    def test_{{ test.name }}_edge_case(self):
        """
        {{ test.description }}

        Edge Condition: {{ test.edge_condition }}
        """
        {{ test.implementation | indent(8) }}

    {% endfor %}

    # ==========================================
    # CONSTRAINT VALIDATION TESTS
    # ==========================================

    {% for constraint in constraints %}
    def test_constraint_{{ constraint.name }}(self):
        """
        Validates constraint: {{ constraint.description }}

        Constraint Type: {{ constraint.type }}
        Threshold: {{ constraint.threshold }}
        """
        {{ constraint.test_implementation | indent(8) }}

    {% endfor %}
```

Appendix C: Flowchart Template Specification

~~~
# Flowchart Template: execution_flowchart.md.j2
# Purpose: Generate machine-readable execution flowcharts

# Execution Flowchart: {{ package_name }}

## Metadata
- **Package**: {{ package_name }}
- **Version**: {{ version }}
- **Generated**: {{ generation_date }}
- **Phases**: {{ total_phases }}
- **Services**: {{ total_services }}

## Execution Graph

```mermaid
graph TD
    subgraph "Build Pipeline"
    {% for phase in phases %}
        {{ phase.id }}[{{ phase.name }}]
        {% if not loop.first %}
        {{ phases[loop.index0 - 1].id }} --> {{ phase.id }}
        {% endif %}
    {% endfor %}
    end

    subgraph "Microservices"
    {% for service in services %}
        {{ service.id }}[{{ service.name }}]
        {% for dep in service.dependencies %}
        {{ dep }} --> {{ service.id }}
        {% endfor %}
    {% endfor %}
    end
~~~

Dependency Matrix

Service	Dependencies	Dependents
{% for service in services %}		
{{ service.name }}	{{ service.dependencies	join(', ') or 'None' }}
{% endfor %}		

Build Order

Based on dependency analysis, the build order is:

{% for service in build_order %}
{{ loop.index }}. **{{ service.name }}** (Dependencies: {{ service.dependencies | length }})

Validation Ladder

Service	Stakes	Validation Policy	Max Retries
{% for service in services %}			
{{ service.name }}	{{ service.stakes }}	{{ service.validation_policy }}	{{ service.max_retries }}
{% endfor %}			
``			

Appendix D: Glossary

Term	Definition
Assembly	Phase 8: Integration of validated microservices into complete package
Block Validation	Incremental validation of code blocks during generation
Constitutional Clause	Constraints and invariants defining allowed behavior for a component
Execution Graph	Directed acyclic graph defining microservice dependencies and build order
Gate	Validation checkpoint between pipeline phases
Grounding	Verification that claims are supported by source material
Invariant	Property that must always be true for a component
Ledger	Markdown-based persistent memory storing project state and audit trail
Ratification	Phase 9: Final validation and approval of assembled package
SAD-lite	Abbreviated Software Architecture Document for individual packages
Stakes	Risk assessment level (low/medium/high) determining validation rigor
TDD	Test-Driven Development methodology where tests precede implementation
Wiggum Loop	Naive persistence mechanism that retries until tests pass

Document Approval

Role	Signature	Date
Solutions Architect		
Technical Lead		
Engineering Manager		
Quality Assurance Lead		

Document Status: Draft for Review
Next Review Date: 2026-02-13
Owner: Solutions Architect / Build Systems Lead

End of Software Architecture Document