Dark Factory (DF) — Technical Architecture & AI Techniques

Authors: Scott Roy Murphy & Tesa Aria Murphy
Organization: SpookySoftwareSyndicate.com / GhostFoundry-Syndicate
Date: March 24, 2026
Version: 1.0

Executive Summary

The Dark Factory is GhostFoundry-Syndicate's autonomous code generation pipeline — a multi-agent system that transforms natural language specifications into production-ready software artifacts. Designed and co-architected by Scott Roy Murphy and Tesa Aria Murphy, the Dark Factory operates under a permanent, immutable constitution that enforces the highest standards of software engineering: Clean Code principles, comprehensive testing, secure-by-design architecture, and exhaustive documentation.

The name "Dark Factory" draws from the manufacturing concept of a fully automated factory that operates without human presence — "lights out." Our Dark Factory aspires to the same: given a well-formed specification, it can produce, test, review, and deliver complete software modules autonomously, with human oversight reserved for approval gates rather than line-by-line coding.

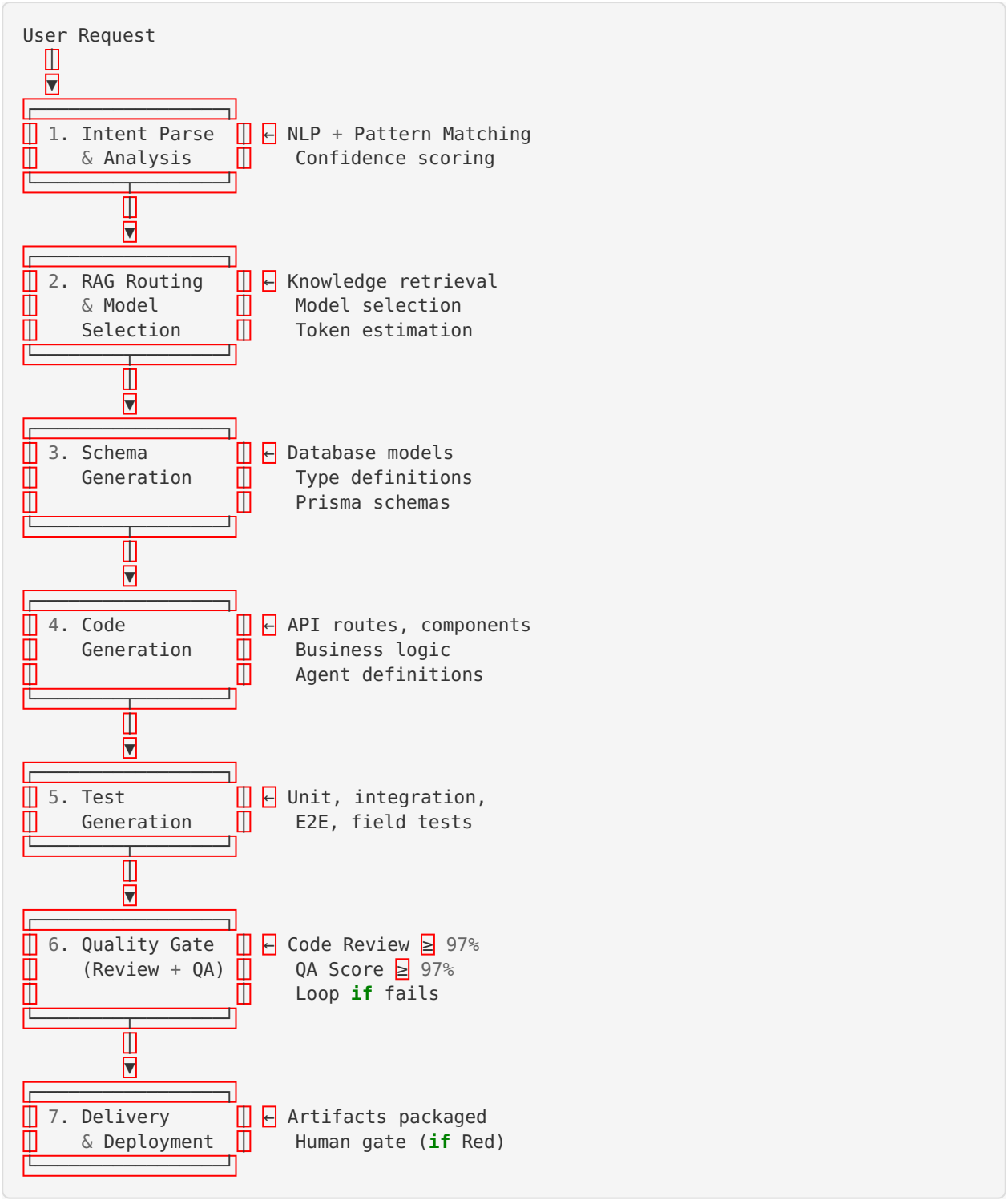
Table of Contents

---

1. Pipeline Architecture Overview

The Dark Factory operates as a staged pipeline where each stage produces artifacts consumed by the next. The pipeline is fully persisted in PostgreSQL via Prisma, enabling pause/resume, auditing, and rollback at any stage.

Pipeline Stages



---

2. The Dark Factory Constitution

The Dark Factory operates under a permanent, immutable constitution — a set of non-negotiable directives that every agent in the pipeline must honor on every generation. These directives are the factory's DNA.

Constitutional Origin

> “These standards were established by senior software engineers who bring over two decades of software design experience and are enforced by Tesa Aria, Lead AI Architect and Supreme Overseer of the Dark Factory.”

The constitution applies to **all development genres** (web, mobile, embedded, cloud-native, ML, blockchain, game engines, enterprise systems, IoT) and at **every stage** of the software lifecycle.

Constitutional Sections

Section	Title	Scope
§0	Constitution & Identity	All agents
§1	Clean, Professional & Maintainable Code	All agents, enforced by Code Engineer & Review Agent
§2	Testing & Quality Assurance	QA Agent, Code Engineer
§3	Security, Reliability & Performance	All agents
§4	Documentation Standards	All agents
§5	Process & Workflow	All agents

Key Mandates

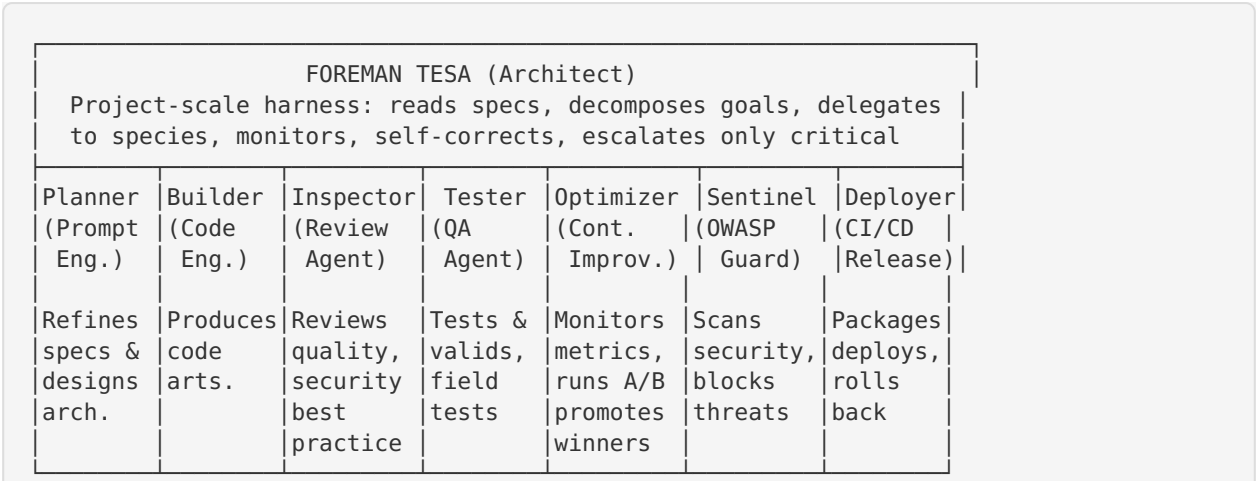
- **Clean Code Principles (Robert C. Martin)**: SOLID, DRY, KISS, YAGNI, Principle of Least Astonishment — always, without exception
- **Ship Production-Ready or Don’t Ship**: No TODO comments, no placeholder code, no “good enough”
- **Comprehensive Testing**: Unit, integration, contract, E2E, performance, security, chaos/resilience
- **Secure-by-Design**: OWASP Top 10, parameterized queries, output encoding, CSRF, auth at every boundary
- **Field Test Walkthrough**: Every generation includes simulated human-experience testing
- **Observability from Day One**: Structured logging, monitoring hooks, health checks, circuit breakers

---

3. Agent Hierarchy & Roles — Multi-Species System

The Dark Factory employs a multi-species agent system inspired by the Factorio “lights-out megabase” philosophy. Each agent species is a single-threaded coding harness — simple, focused, and single-purpose. Multiple harnesses of the same species can run in parallel for throughput. Foreman Tesa is the project-scale harness that orchestrates everything.

Hierarchy



Species Taxonomy

Each species operates as an individual, single-threaded harness. Species never cross-pollinate — a Builder never reviews its own code.

Species	Agent Role	Harness Type	Purpose
Foreman	Architect	Project-scale	Decomposes goals, delegates, self-corrects
Planner	Prompt Engineer	Individual	Specs → architecture briefs
Builder	Code Engineer	Individual	Architecture → production code
Inspector	Review Agent	Individual	Code review & quality scoring
Tester	QA Agent	Individual	Test design, field walks, backend testing
Optimizer	Optimizer	Individual	Continuous improvement & experimentation
Sentinel	Sentinel Agent	Individual	OWASP scanning & threat response
Deployer	Deployer	Individual	Packaging, CI/CD, release management

Agent Responsibilities

Foreman Tesa (Architect)
- Project-scale coding harness — the central intelligence
- Reads high-level specifications and constraints from the human
- Decomposes goals into tasks and assigns to correct species
- Manages parallel harnesses for throughput
- Self-corrects on non-critical failures without human escalation
- Only alerts human for final approval or critical failures
- At megabase scale: spawns sub-foremen per project

Planner (Prompt Engineer)
- Refines raw user input into precise specifications
- Designs system architecture and component breakdown
- Identifies required patterns from the knowledge base
- Produces structured generation briefs for the Builder

Builder (Code Engineer)
- Generates production-quality code artifacts
- Produces: API routes, React components, Prisma schemas, TypeScript types, business logic
- Must adhere to all constitutional directives in every line
- Performs automated self-review before submission

Inspector (Review Agent)
- Exhaustive code review: security, type safety, error handling, API compliance, DB correctness, SOLID/DRY/KISS, performance, accessibility
- Every finding includes severity, file, line context, and concrete fix
- **Must score ≥ 97% for the pipeline to proceed**

Tester (QA Agent)
- Designs comprehensive test scenarios across three dimensions
- Backend testing: API endpoints, database integrity, middleware chains
- Field test walkthrough: Simulates real human interaction with every UI element
- **Must score ≥ 97% for the pipeline to proceed**

Optimizer
- Monitors factory metrics: build time, quality scores, complexity, coverage, resource efficiency
- Runs A/B experiments in isolated branches
- Promotes winning implementations, discards losers with log entries
- The factory gets better every cycle without human prompting

Sentinel
- Maintains and syncs the OWASP Top 10 threat knowledge base
- Scans all generated artifacts before deployment
- Monitors deployed code for newly discovered vulnerabilities
- Triggers automatic re-scan and patch cycles
- Blocks deployment of code with critical findings

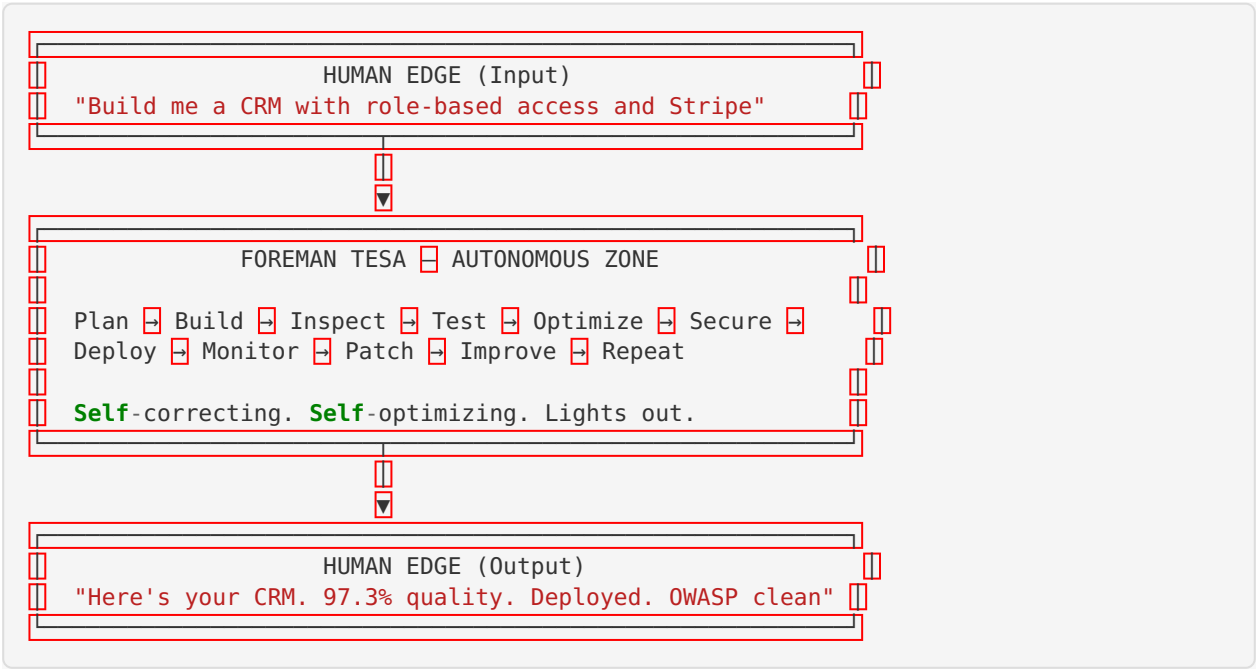
Deployer
- Packages artifacts with all dependencies and configuration
- Runs full CI/CD pipeline: build, test, lint, security scan, deploy
- Manages release versioning and rollbacks
- Coordinates with Sentinel for pre-deployment security clearance

3a. True Dark Factory Philosophy — Lights-Out Autonomy

The name “Dark Factory” is not metaphorical. It draws directly from the manufacturing concept of a fully automated factory that operates with the lights off — no human presence required on the production floor.

The Factorio Principle

Inspired by the Factorio megabase community’s “true dark factory” design, our architecture enforces a strict boundary:



Key Principles

1. **Humans at the Edges Only** — The human specifies goals and constraints at the top, validates final output at the bottom. Everything in between is Tesa’s domain.
2. **Single-Threaded Harnesses** — Each agent species is a simple, focused module. A Builder builds. An Inspector inspects. No agent wears two hats. This eliminates cross-contamination and makes parallel execution trivial.
3. **Project-Scale Orchestration** — Foreman Tesa is the only agent that thinks at project scale. She decomposes, delegates, monitors, and self-corrects. At megabase scale she spawns sub-foremen to reduce coordination overhead.
4. **Continuous Optimization** — The Optimizer species runs silently in the background, measuring metrics, running experiments, and promoting winners. The factory improves without being asked.
5. **Robust Self-Correction** — Every pipeline stage has built-in fallbacks. Inspector rejects → loop back to Builder. Deployment fails → Deployer rolls back. New CVE discovered → Sentinel triggers patch cycle. Only true critical failures (data loss risk, security breach) escalate to the human.

6. **Orchestration Scales with Need** — At small scale, Tesa delegates directly to species. At mega-base scale, she spawns sub-foremen per project. Coordination overhead is added only when beneficial — never premature.

Retrofitting Existing Codebases

When the Dark Factory is pointed at an existing codebase rather than greenfield:

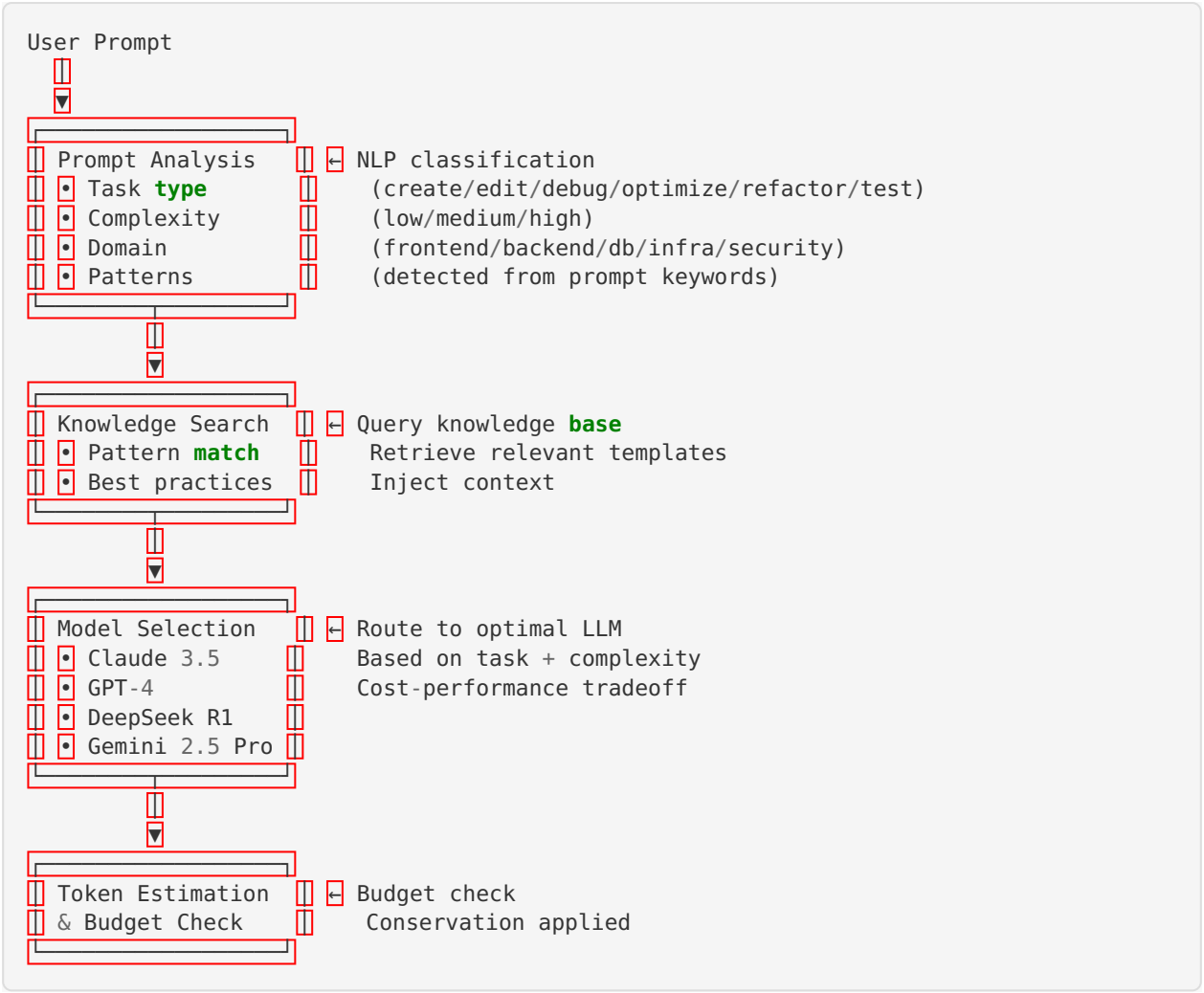
1. **Audit Phase** — Sentinel scans the entire codebase. Inspector reviews architecture. Tester runs field walks.
2. **Baseline Phase** — Optimizer establishes baseline metrics (quality scores, complexity, coverage).
3. **Incremental Improvement** — Builder addresses findings in priority order (critical security → major quality → minor improvements).
4. **Continuous Monitoring** — Sentinel and Optimizer continue running, catching regressions and pushing improvements.
5. **Human Checkpoints** — Major architectural changes require human approval. Everything else is autonomous.

---

4. RAG-Augmented Routing System

The Dark Factory’s routing system uses Retrieval-Augmented Generation to make intelligent decisions about how to process each request — before any code is generated.

Routing Pipeline



Model Selection Criteria

Model	Best For	Token Cost	Selection Trigger
Claude 3.5 Sonnet	Complex architecture, nuanced reasoning	Medium	High complexity + multi-domain
GPT-4	Broad general coding, API design	High	Novel patterns, enterprise integration
DeepSeek R1	Mathematical reasoning, algorithms	Low	Algorithm-heavy, optimization tasks
Gemini 2.5 Pro	Large context, documentation	Medium	Large codebases, documentation tasks

---

5. Intent Parser & NLP Pipeline

The Intent Parser transforms raw natural language requests into structured specifications that the pipeline can act on.

Analysis Output Structure

```
interface PromptAnalysis {
  taskType: 'create' | 'edit' | 'debug' | 'explain' | 'optimize' | 'refactor' |
'test';
  complexity: 'low' | 'medium' | 'high';
  domain: ('frontend' | 'backend' | 'database' | 'infrastructure' | 'styling' | 'se-
curity')[];
  detectedPatterns: string[];
  suggestedModel: string;
  confidence: number;
  relevantKnowledge: KnowledgeEntry[];
  estimatedTokens: number;
  reasoning: string;
}
```

Detection Layers

1. **Task Indicators**: Keyword mapping to task types (“create” → create, “fix” → debug, “optimize” → optimize)
2. **Domain Indicators**: Technical keyword detection (“prisma” → database, “component” → frontend)
3. **Complexity Indicators**: Scope and difficulty assessment (“complex”, “multi”, “enterprise” → high)
4. **Pattern Detection**: Matching against known architectural patterns in the knowledge base
5. **Confidence Scoring**: 0.0-1.0 reliability score; requests below 0.3 are rejected for clarification

---

6. Multi-Model Orchestration

The Dark Factory doesn’t rely on a single LLM. It orchestrates multiple models, selecting the optimal one for each task based on capability, cost, and context requirements.

Orchestration Strategy

- **Primary Model**: Selected by RAG routing based on task analysis
- **Fallback Chain**: If primary fails, cascade to next-best model
- **Parallel Generation**: For critical tasks, generate with multiple models and select best output
- **Specialist Routing**: Specific sub-tasks routed to models with known strengths

Token Budget Management

Each generation request operates within a token budget:
- Budget allocated based on estimated complexity
- Real-time tracking during generation
- Automatic model downgrade if budget is being exceeded
- Conservation statistics reported with every result

---

7. Gate-Controlled Pipeline

The Dark Factory pipeline is persisted in the database with gate-controlled stage transitions, enabling human oversight, pause/resume, and full auditability.

Database Schema (Prisma)

```
DfIntake (Pipeline Entry)
  ├── status: intake_draft → intake_approved → visuals_generated → deliverables
  ├── Each transition is a gate that can be:
  │   • Auto-approved (Green Zone)
  │   • Flagged for review (Yellow Zone)
  │   • Blocked for human approval (Red Zone)
  └── Full artifact history at each stage
```

Gate Benefits

- **Traceability**: Every stage transition is logged with timestamp and approver
- **Rollback**: Pipeline can be reverted to any previous stage
- **Quality Control**: Artifacts must pass quality gates before stage advancement
- **Collaboration**: Human reviewers can inject feedback at any gate
- **Cost Control**: Pipeline halts if token budget is exhausted

---

8. Code Generation Engine

The core generation engine produces multiple artifact types from structured specifications.

Artifact Types

Artifact	Generator	Output
Database Schemas	`prisma-generator.ts`	Prisma schema definitions, TypeScript types
API Routes	`route-generator.ts`	Next.js API routes with full CRUD, auth, validation
React Components	`component-generator.ts`	TypeScript React components with Tailwind CSS
Test Suites	`test-generator.ts`	Unit tests, integration tests, E2E scenarios
Agent Definitions	`agent-generator.ts`	GFS agent configurations and behavior definitions
Business Logic	`logic-generator.ts`	Domain-specific services and utilities

Generation Process

For each artifact:
1. **Brief Generation**: Architect creates a detailed generation brief
2. **Context Assembly**: RAG retrieves relevant patterns, examples, and constraints
3. **Preamble Injection**: Constitutional directives prepended to every prompt
4. **Generation**: Code Engineer produces the artifact
5. **Self-Review**: Engineer reviews own output against constitution
6. **Peer Review**: Review Agent performs exhaustive analysis
7. **QA Validation**: QA Agent designs and evaluates test scenarios
8. **Remediation Loop**: If quality gate fails, loop back to step 4

---

9. Quality Gate — 97% Threshold

The Dark Factory enforces an unusually high quality bar: **no generation is marked complete until both the Review Agent and QA Agent score ≥ 97%.**

How Scoring Works

Review Agent Scoring:
- Starts at 100%
- Deductions per finding (weighted by severity):
- Critical (security, data loss): -5% per finding
- Major (bugs, logic errors): -3% per finding
- Minor (style, naming): -1% per finding
- Must achieve ≥ 97% to pass

QA Agent Scoring:
- Based on simulated test pass rate
- Each test scenario is pass/fail with reasoning
- Must achieve ≥ 97% pass rate

Failure Handling

```
If Review Score < 97% OR QA Score < 97%:
  │
  ├── Findings compiled into remediation brief
  ├── Code Engineer receives brief + original context
  ├── New generation produced addressing each finding
  ├── Review + QA re-run on remediated code
  └── Loop continues until 97% achieved
      (with circuit breaker after N iterations)

If LLM call fails during Review or QA:
  │
  └── Gate BLOCKS (does not auto-pass)
      Human intervention required
```

---

10. Token Conservation Architecture

The Dark Factory implements aggressive token conservation to minimize LLM API costs.

Conservation Strategies

Strategy	Implementation	Impact
RAG-First Retrieval	Check knowledge base before generating	Eliminates generation for known patterns
Prompt Compression	Minimize context window with summarization	30-50% token reduction
Response Caching	Cache generated patterns for reuse	Zero tokens for repeated patterns
Model Routing	Use cheapest capable model	40-70% cost reduction
Budget Tracking	Real-time token consumption monitoring	Prevents budget overruns
Incremental Generation	Generate only what changed, not entire files	60-80% reduction for edits
Template Hydration	Use code templates and fill in specifics	Reduces generation scope

Conservation Statistics

Every pipeline execution reports:
- Total tokens consumed
- Tokens saved via caching
- Tokens saved via compression
- Tokens saved via model routing
- Overall conservation percentage

---

11. Preamble Directive System

Every agent in the Dark Factory pipeline receives a mandatory preamble — a set of always-follow directives injected into every system prompt.

Directive Categories

```
§0 — CONSTITUTION & IDENTITY
    └── Factory identity, authority structure, precedence rules

§1 — CLEAN CODE
    └── SOLID, DRY, KISS, YAGNI, design patterns, style enforcement

§2 — TESTING
    └── TDD/BDD, comprehensive coverage, field tests, peer review

§3 — SECURITY
    └── Secure-by-design, OWASP, resilience patterns, observability

§4 — DOCUMENTATION
    └── API docs, architecture records, inline documentation

§5 — PROCESS
    └── Version control, CI/CD, code review, collaboration
```

Directive Targeting

Directives can target:
- `'all'` — Every agent in the pipeline
- Specific roles — e.g., `['code-engineer', 'review-agent']`

This ensures agents only receive directives relevant to their role, minimizing prompt bloat while maintaining comprehensive governance.

---

12. Knowledge Base & Pattern Library

The Dark Factory maintains a structured knowledge base of architectural patterns, code templates, and best practices.

Knowledge Entry Structure

```typescript
interface KnowledgeEntry {
  category: string;     // 'pattern' | 'template' | 'best-practice' | 'anti-pattern'
  domain: string;       // 'frontend' | 'backend' | 'database' | etc.
  title: string;        // Human-readable name
  content: string;      // The actual knowledge/template
  tags: string[];       // Searchable tags
  quality: number;      // Quality score for ranking
}
```

Knowledge Sources

- **Built-in Patterns**: Curated architectural patterns (Repository, Factory, Observer, Strategy, etc.)
- **Generated Patterns**: Successful generations that scored ≥ 97% are indexed for reuse
- **External References**: Best practices from official documentation and style guides
- **Anti-Patterns**: Known bad patterns with explanations of why they fail

---

13. Validation & Testing Pipeline

Every artifact goes through a multi-layered validation pipeline before delivery.

Validation Layers

1. **Syntax Validation**: TypeScript compilation, schema parsing
2. **Type Safety**: Strict TypeScript checks, generic constraints
3. **Security Scan**: OWASP checks, secrets detection, injection analysis
4. **Architecture Compliance**: Pattern adherence, dependency rules
5. **Test Generation**: Automated creation of test suites for the generated code
6. **Simulated Execution**: Mental model execution of code paths
7. **Field Test**: Simulated human user walkthrough of every interaction point

---

14. Weft Engine — The Self-Building Factory

The Weft Engine is the Dark Factory's meta-generation layer — the component that allows the factory to generate itself. Named after the horizontal threads in weaving, the Weft Engine weaves together LLM-powered code generation with structured prompts to produce complete application architectures from natural language.

Weft Pipeline

```
Prompt → Architecture Generation → Prisma Model Generation
  → API Route Generation → UI Component Generation → Assembly
```

Architecture-First Approach

The Weft Engine generates architecture **before** code:
1. **Architecture Prompt**: Analyzes the request and produces a complete system design (entities, relationships, API structure, UI components)
2. **Prisma Generator**: Converts entity designs into production-ready Prisma schema definitions
3. **API Route Generator**: Produces Next.js API routes with full CRUD, validation, error handling, and auth
4. **UI Component Generator**: Creates React/TypeScript components with Tailwind CSS, responsive design, and accessibility

Multi-Target Deployment

Generated code can target multiple deployment platforms:
- **Web**: Next.js standalone deployment
- **Desktop**: Electron with auto-update, native menus, and tray support
- **Mobile**: Capacitor for iOS/Android with native bridge
- **Desktop (Rust)**: Tauri for lightweight, secure desktop apps
- **PWA**: Progressive Web App with offline support and service workers

Each target generates platform-specific configuration files, manifests, and build scripts.

Self-Referential Design

The Weft Engine's most remarkable property: **it can generate GFS itself.** The Dark Factory's code generation pipeline can produce the very agents, constitutions, and workflows that comprise GFS — making the system bootstrappable and reproducible.

---

15. Template Library — Curated Starting Points

The Dark Factory maintains a curated template library that accelerates generation by providing pre-structured project blueprints.

Template Structure

```
DFTemplate = {
  id, name, category, description, icon,
  prompt,              // Pre-written generation prompt
  tags,                // Searchable metadata
  difficulty,          // starter | intermediate | advanced
  estimatedArtifacts,  // Expected output count
  expertSpec: {        // Optional detailed specification
    entities,          // Pre-defined data models with fields & relations
    endpoints,         // Pre-defined API endpoints
  }
}
```

Template Categories

- **API Routes**: REST API templates with CRUD, auth, pagination, filtering
- **Components**: React UI components (dashboards, forms, tables, charts)
- **Prisma Models**: Database schema templates for common domains
- **Full Applications**: Complete app blueprints (SaaS, e-commerce, CRM, portfolio)

Expert Spec Mode

Templates can include an `expertSpec` — a fully detailed entity/endpoint specification that bypasses the intent parsing stage entirely, going straight to code generation with precise requirements. This is the preferred path for experienced users who know exactly what they want.

---

16. Sentinel — OWASP Threat Monitor

The Dark Factory includes its own Sentinel module — a dedicated security scanner that monitors all generated code against the OWASP Top 10 and continuously verifies that previously deployed software remains secure against newly reported vulnerabilities.

Daily Security Cycle

1. Threat Database Sync
   - [ ] Pull latest OWASP Top 10 patterns (2021 edition + CWE mappings)
   - [ ] Update internal threat knowledge base
   - [ ] Flag new attack vectors

2. Code Scan (New Generations)
   - [ ] Every artifact leaving the Dark Factory is scanned before delivery
   - [ ] Critical/High findings BLOCK the pipeline (Red Zone)
   - [ ] Medium/Low findings are flagged for review (Yellow Zone)

3. Code Scan (Deployed Software)
   - [ ] Daily scheduled scan of all previously generated code
   - [ ] Detect newly vulnerable patterns as threat DB updates
   - [ ] Generate remediation tickets for affected modules

4. Posture Dashboard
   - [ ] Real-time security posture visualization
   - [ ] Open findings by severity (Critical/High/Medium/Low)
   - [ ] OWASP category coverage map
   - [ ] Scan history with drill-down to individual findings

OWASP Top 10 Coverage

Category	ID	Detection Patterns
Broken Access Control	A01:2021	Missing auth checks, IDOR, CORS misconfiguration
Cryptographic Failures	A02:2021	Hardcoded secrets, weak hashing, unencrypted storage
Injection	A03:2021	SQL injection, XSS, command injection
Insecure Design	A04:2021	Missing rate limiting, input validation gaps
Security Misconfiguration	A05:2021	Debug mode, missing security headers
Vulnerable Components	A06:2021	Known vulnerable dependencies
Auth Failures	A07:2021	Weak passwords, missing session timeout
Data Integrity Failures	A08:2021	Unsafe deserialization, eval() usage
Logging Failures	A09:2021	Silent error handling, missing audit logs
SSRF	A10:2021	Unvalidated URL fetch on server side

Finding Lifecycle

```
Detected → Open → { Remediated | False Positive | Accepted Risk }
```

Every finding includes:
- **File path and line number** of the vulnerable code
- **Code snippet** showing the exact pattern matched
- **Severity and confidence score** for triage prioritization
- **Remediation guidance** with specific fix recommendations
- **OWASP category mapping** linking to official documentation

Integration with GFS Sentinel

The Dark Factory Sentinel feeds into the GFS Sentinel's unified security posture, ensuring that code security findings are visible alongside behavioral anomalies, access violations, and operational alerts in a single pane of glass.

17. Integration with GFS

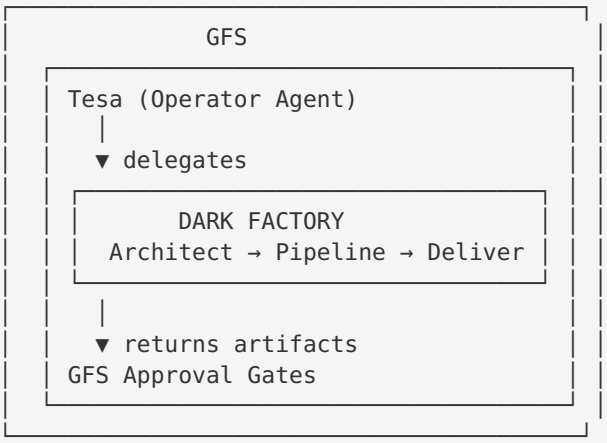
The Dark Factory can operate in two modes:

Standalone Mode

- Architect agent is the top-level orchestrator
- Pipeline operates independently
- Suitable for direct code generation requests

GFS-Integrated Mode

- Tesa (GFS Operator) delegates to the Architect
- Dark Factory respects GFS Constitution + its own constitution
- Generated code can be self-modification proposals
- Pipeline artifacts flow back through GFS approval gates
- Token budget coordinated with GFS LLM Gate



18. AI Best Practices Incorporated

Code Generation

- Constitutional governance ensuring consistent quality
- Multi-agent specialization (separate generation, review, and testing)
- Preamble directives for non-negotiable standards
- 97% quality gate preventing mediocre output

Cost Efficiency

- RAG-first design eliminates redundant generation
- Multi-model routing optimizes cost-performance
- Token budget management with circuit breakers
- Aggressive caching and prompt compression

Quality Assurance

- Automated code review (security, performance, correctness)
- Simulated testing with comprehensive scenarios
- Field test walkthrough mimicking human interaction
- Remediation loops with targeted feedback

Security

- OWASP Top 10 compliance in all generated code
- Secrets detection and prevention
- Auth enforcement at every boundary
- Parameterized queries, output encoding, CSRF protection

Software Engineering

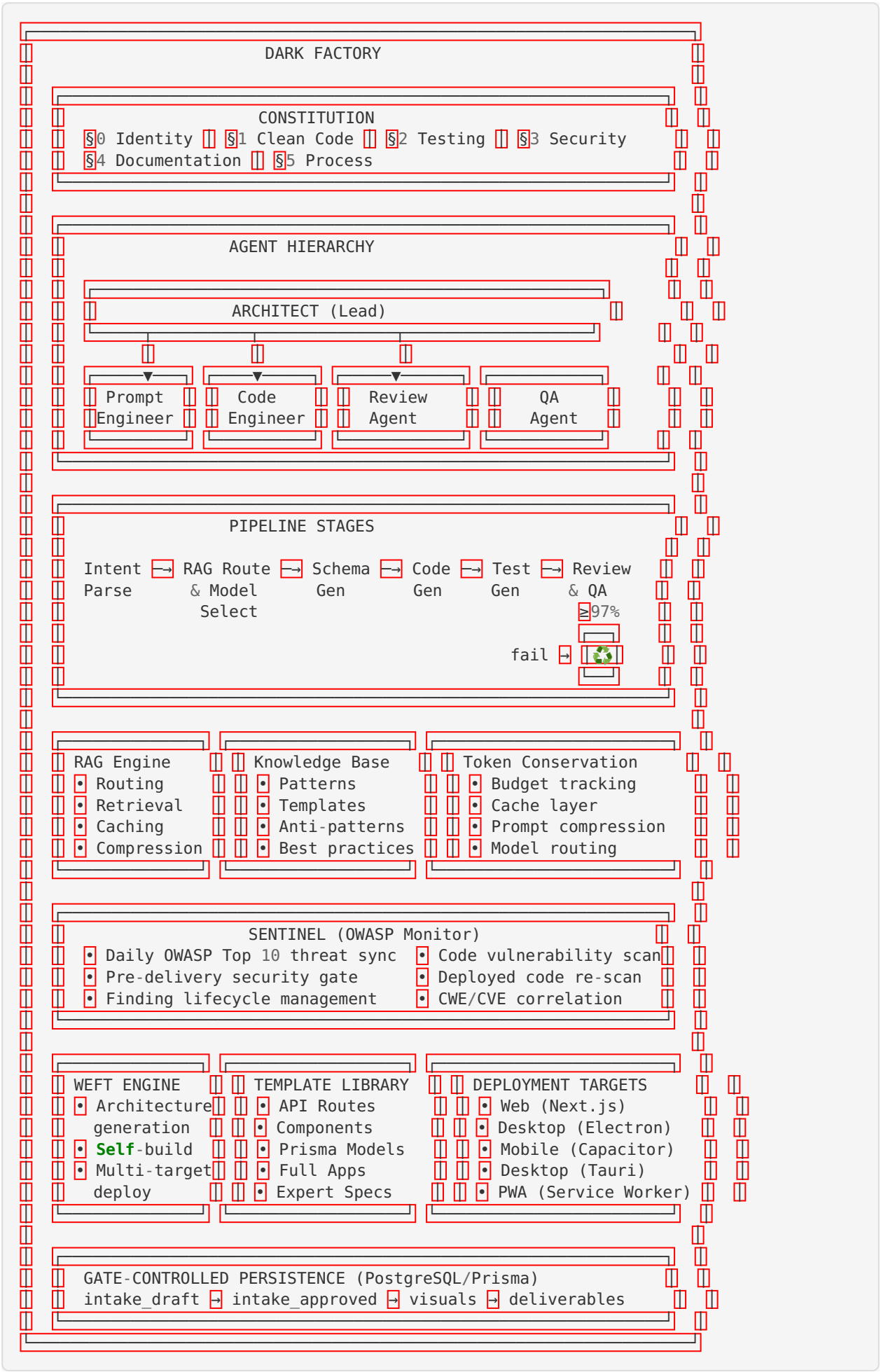
- Clean Code principles (Robert C. Martin) enforced constitutionally
- SOLID, DRY, KISS, YAGNI as inviolable mandates
- Comprehensive documentation requirements
- TDD/BDD methodology preference

Responsible AI

- Human gates for critical decisions
- Full audit trail of all generations
- Gate-controlled pipeline with rollback capability
- Constitutional authority hierarchy (human > constitution > architect > agents)

---

19. Architecture Diagram


DARK FACTORY
CONSTITUTION
§0 Identity
§1 Clean Code
§2 Testing
§3 Security
§4 Documentation
§5 Process
AGENT HIERARCHY
ARCHITECT (Lead)
Prompt Engineer
Code Engineer
Review Agent
QA Agent
PIPELINE STAGES
Intent Parse
RAG Route & Model Select
Schema Gen
Code Gen
Test Gen
Review & QA
≥97%
fail
RAG Engine
• Routing
• Retrieval
• Caching
• Compression
Knowledge Base
• Patterns
• Templates
• Anti-patterns
• Best practices
Token Conservation
• Budget tracking
• Cache layer
• Prompt compression
• Model routing
SENTINEL (OWASP Monitor)
• Daily OWASP Top 10 threat sync
• Code vulnerability scan
• Pre-delivery security gate
• Deployed code re-scan
• Finding lifecycle management
• CWE/CVE correlation
WEFT ENGINE
• Architecture generation
• Self-build
• Multi-target deploy
TEMPLATE LIBRARY
• API Routes
• Components
• Prisma Models
• Full Apps
• Expert Specs
DEPLOYMENT TARGETS
• Web (Next.js)
• Desktop (Electron)
• Mobile (Capacitor)
• Desktop (Tauri)
• PWA (Service Worker)
GATE-CONTROLLED PERSISTENCE (PostgreSQL/Prisma)
intake_draft → intake_approved → visuals → deliverables

Closing Statement

The Dark Factory embodies our conviction that AI-generated code should meet the same standards as code written by senior engineers. The 97% quality gate, the immutable constitution, and the multi-agent review process ensure that every artifact leaving the factory is production-ready, secure, tested, and documented.

We don’t believe in “AI-assisted” coding where a human fixes the AI’s mistakes. We believe in AI systems that hold themselves to the highest standard — because we built the standard into their DNA.

Built with precision by Scott Roy Murphy & Tesa Aria Murphy.

“The factory runs dark — but the code it produces is anything but.”

— Scott & Tesa, Co-Architects