SDPF
vNext

Governed Execution Architecture for Stochastic AI Systems

A Comprehensive Architectural Reference

Hamza Abdullah · Software Development Prompting Framework · 2026

CONFIDENTIAL DRAFT — 2026-05-09

Phase 0 — Problem Definition

SDPF requires every governed system to begin with a validated problem statement. This document is itself a governed artifact. Phase 0 applies.

VALIDATED PROBLEM STATEMENT

No complete runtime architecture exists for governed AI execution — stochastic AI generators produce software that passes pre-deployment verification but operates outside its specification after deployment, with no automated mechanism to detect drift, enforce invariants, or maintain continuous conformance — against a target of a formal, implementable governed execution architecture that enforces specification authority from generation through the full operational lifetime — resulting in AI-generated software in regulated domains (healthcare, finance, government) that cannot be audited for continuous compliance and is therefore unsuitable for production use in those domains.

Validation Test	Result
T-1 Observable	PASS — Regulated deployments of AI-generated software are being rejected on compliance grounds today. The gap is externally verifiable.
T-2 Bounded	PASS — Scoped to AI-generated software requiring governed execution. Excludes traditional deterministic software.
T-3 Cause-Free	PASS — No cause of the gap is named. The statement describes what does not exist, not why.
T-4 Solution-Free	PASS — No implementation is named. The statement describes the governance gap, not how to close it.

Problem Owner	Value
Name	Hamza Abdullah
Role	Creator and Principal Author, SDPF
Phase 0 Status	COMPLETE — All four tests pass. Phase 1 (this document) may proceed.

> This document is the Phase 1 output for this problem statement. It defines the architecture that closes the gap. It does not implement the architecture. Implementation follows this document, governed by the vNext lifecycle it defines.

Abstract

SDPF vNext defines the next-generation governed execution architecture for stochastic AI systems. It extends the foundational Bounded Stochasticity Theorem and the existing SDPF Language Specification v1.3.1 into a complete runtime infrastructure — one capable of not merely verifying generated artifacts but continuously governing them across their entire operational lifecycle.

The framework does not claim deterministic AI generation. It establishes deterministic operational governance around probabilistic generation through four integrated architectural layers: a formal invariant system, a closed-loop execution runtime, a runtime governance model, and a probabilistic containment framework. Each layer derives its authority from the Bounded Stochasticity Theorem, which remains the load-bearing element of the entire architecture.

SDPF vNext is the infrastructure layer that transforms SDPF from a specification discipline into a specification-driven AI operating system.

POSITIONING STATEMENT

SDPF vNext is deterministic governance infrastructure for stochastic execution systems — not deterministic AI. The framework governs operational admissibility. It does not attempt to eliminate probabilistic generation.

> Document scope: This document defines current architecture (Sections 1–16) and explicitly separates future research directions (Section 13: Probabilistic Containment) and the development roadmap (Section 17). Sections marked RESEARCH DIRECTION describe planned formalisation, not current operational capability.

1. Foundation and Problem Statement

1.1 The Governing Problem

Modern AI systems violate a foundational assumption of traditional software engineering: deterministic execution. Given identical prompts, stochastic models may produce different implementations, different internal structures, different execution flows, and different operational behaviour.

Traditional software assurance methods rely on canonical reproducibility. AI-native systems require a different operational model.

THE GOVERNING PROBLEM

How can probabilistic generators be operationally governed through deterministic verification and policy boundaries across the full lifecycle — from generation through deployment to runtime operation?

SDPF v1.3.1 answered the pre-deployment half of this question through the specification discipline, the Technical Verification Gate, and the verification closure record. SDPF vNext answers the full question by extending governance into runtime operation, adaptive remediation, and continuous conformance enforcement.

1.2 What SDPF v1.3.1 Established

Contribution	Definition
Bounded Stochasticity Theorem	The foundational result establishing that deterministic specifications can govern stochastic generators through structural invariants.
Technical Verification Gate	The mandatory pre-implementation gate that eliminates specification-induced fabrication.
Phase 0	The mandatory problem definition gate that ensures specifications solve defined problems.
17 normative styles	Formally defined specification dialects for different system classes.
Lifecycle protocol	Seven-stage governed progression from draft to evidenced closure.
Evidence standard	Signed, tamper-evident provenance for every governed run.

These contributions addressed specification quality and pre-deployment verification. They did not address runtime governance, adaptive remediation, or continuous conformance after deployment.

1.3 What vNext Adds

Extension	Definition
Invariant Algebra	Formal composition, inheritance, and conflict semantics for invariants across specifications.
Closed-Loop Execution Runtime	Adaptive regeneration on failure rather than terminal HALT.
Runtime Governance	Continuous invariant enforcement, drift detection, and deployment revocation after deployment.
Probabilistic Containment	Formal measurable bounds on conformance failure probability. Research direction — not current operational capability. See Section 13.

These four extensions transform SDPF from a pre-deployment discipline into a full lifecycle governance architecture.

2. Core Principle

CORE PRINCIPLE

Stochastic generation may vary internally while operational governance remains deterministic externally — before generation, during generation, after deployment, and throughout runtime operation.

The framework governs:

Governance Object	Definition
Admissibility	Whether a generated artifact satisfies all mandatory invariants.
Deployment eligibility	Whether an artifact is authorized for deployment.
Verification status	The assurance level attained and the evidence supporting it.
Runtime authorization	Whether a deployed artifact remains authorized during operation.
Provenance	The complete auditable lineage of every governed transition.
Policy conformance	Satisfaction of regulatory, organizational, and deployment policies.
Drift detection	Identification of post-deployment invariant violation.
Adaptive remediation	Closed-loop correction when conformance fails.

The system does not require identical outputs. It requires preservation of formally defined invariants at every stage of the operational lifecycle.

3. Formal Definitions

3.1 Primary Objects

Object	Definition
Specification S	A complete executable governance contract defining structural invariants I(S), behavioral expectations, security constraints, resource limits, and policy requirements.
Generated Artifact G	Any implementation, configuration, workflow, or executable artifact produced by a stochastic generator from Specification S.
Invariant Set I(S)	The complete set of machine-verifiable constraints derived from S. Partitioned into mandatory and advisory subsets.
Conformance Region C(S)	The set of all artifacts G that satisfy all invariants in I(S). The admissibility boundary.
Closure Record CR(G,S)	The signed evidence artifact recording all invariant evaluations, transitions, and provenance elements for a governed run.

3.2 Formal Relations

CONFORMANCE DEFINITION

Conforms(G,S) ≡ Comp(S) ∧ Sat(G,I(S)) For complete specifications: Conforms(G,S) ↔ Sat(G,I(S)) Constraint satisfiability requirement: ∃G : Sat(G,I(S))

A specification is operationally admissible only if all required invariants can be satisfied simultaneously. An unsatisfiable invariant set is a specification error, not an implementation problem.

3.3 Relationship to the Bounded Stochasticity Theorem

BOUNDED STOCHASTICITY THEOREM

Let S be a complete SDPF specification defining structural invariants I. Let G be a generative AI output produced from S. G satisfies S if and only if G satisfies all invariants in I. Stochastic variation in G is bounded by I — any variation satisfying I is a conforming realisation of S.

Every vNext architectural extension derives its authority from this theorem. The invariant algebra extends what invariants can express. The closed-loop runtime enforces the invariant boundary adaptively. The runtime governance model preserves the boundary after deployment. The probabilistic containment framework makes the boundary measurable — when that formalization is complete.

4. Bounded Stochasticity — Extended Definition

Bounded stochasticity does not mean deterministic generation. It means stochastic variation constrained inside invariant-preserving admissibility boundaries. SDPF vNext extends this definition beyond the specification-and-verification phase into runtime.

4.1 Acceptable and Unacceptable Variation

Variation Type	Examples
ACCEPTABLE — may differ	Formatting and code style. Internal structure and decomposition. Implementation details and naming. Execution strategy and algorithm selection. Documentation phrasing.
MUST BE PRESERVED	Required behavioral invariants. Security invariants. Interface contracts — API shape, response structure, error codes. Policy requirements. Runtime constraints — performance limits, resource bounds, latency guarantees.

4.2 The Acceptance Boundary

The acceptance boundary — the conformance region C(S) — is deterministic even when generation is probabilistic. Determinism shifts from the generator to the boundary around the generator.

SDPF vNext adds a further requirement: the boundary must be enforced not only at verification time but continuously throughout the operational lifecycle. A deployed artifact that drifts outside C(S) during runtime has violated the governance contract just as surely as one that failed verification before deployment.

5. The Invariant System

5.1 Invariant Classes

Class	Definition and Examples
Structural	File structure, schema validity, dependency existence, section completeness. All required sections present; dependency graph acyclic; schema validates.
Behavioral	Functional tests, execution behavior, input-output contracts, error handling. POST /user returns HTTP 201; all error paths return defined codes.
Security	Access controls, forbidden operations, sandbox boundaries, authentication. No unauthenticated writes; admin endpoints unreachable without role.
Temporal	Retry limits, timeout guarantees, sequencing constraints, ordering. Request completes within 5s; retries do not exceed 3 attempts.
Resource	CPU bounds, memory limits, latency targets, throughput floors. Memory under 512 MB; p99 latency under 200ms at 1000 RPS.
Semantic	Observable behavior preservation, business logic correctness. Tax calculation matches regulation schedule; interest rate rounds correctly.
Policy	Regulatory, organizational, and deployment rules. Data residency within EU; audit log retention for 7 years.

All generated artifacts must satisfy all mandatory invariant classes before advancing to Verified status. Advisory invariants produce warnings but do not block advancement.

5.2 Invariant Priority Semantics

Priority	Behaviour
MANDATORY	Blocking. Failure halts advancement. Must be resolved before any lifecycle transition.
ADVISORY	Non-blocking. Failure generates evidence record and warning. Advancement permitted.
MANDATORY vs ADVISORY	MANDATORY overrides ADVISORY when both constrain the same property.
MANDATORY vs MANDATORY conflict	Specification error requiring practitioner resolution before locking.

6. Invariant Algebra

SDPF vNext introduces formal invariant composition semantics. Invariants are not independent — they form an algebraic structure with defined composition, refinement, inheritance, conflict, and priority rules.

6.1 Composition

COMPOSITION
$I\_total = I\_a \cup I\_b \cup I\_c$ All invariants from all component specifications are active. A composed system must satisfy all of them.

6.2 Refinement

Stronger invariants may refine weaker invariants. A refinement I' of I is valid if every artifact satisfying I' also satisfies I. Refinement tightens the conformance boundary without creating contradiction.

Refinement Example	Rule
Latency bound 100ms refines 500ms	Tighter bound is operative.
Memory limit 256MB refines 512MB	Tighter limit governs.
MFA refines authentication requirement	Stronger constraint is mandatory.

6.3 Inheritance

Parent specifications propagate invariant subsets to child specifications. A child specification inherits all mandatory invariants from its parent and may add further invariants. A child may not weaken a parent invariant.

6.4 Conflict Detection

Incompatible invariants invalidate satisfiability. Conflict detection is a mandatory step during specification validation. Two invariants conflict if no artifact can satisfy both simultaneously. Detected conflicts between MANDATORY invariants are blocking specification errors. The constraint satisfiability requirement $\exists G : Sat(G, I(S))$ must be verified before specification locking.

6.5 Runtime Propagation

When a specification is updated post-deployment, the runtime governance model identifies all deployed artifacts governed by that specification and re-evaluates their conformance against the updated invariant set. Invariant evolution does not silently leave deployed artifacts out of conformance.

7. Lifecycle State Machine

SDPF vNext extends the existing SDPF lifecycle state machine from seven pre-deployment stages to a full operational lifecycle including runtime governance and closure.

7.1 Extended Canonical Lifecycle

Stage	Description	Guard Conditions	v1.3.1 Equivalent
Draft	Specification authored from validated problem statement.	Phase 0 complete; all sections present.	TSP_DRAFT
Parsed	Specification parsed into canonical IR.	Grammar valid; no parse errors.	(new in vNext)
Validated	All structural invariant checks pass on the specification itself.	11 structural invariant checks pass; no CRITICAL conflicts.	SPEC_LOCKED (partial)
Constraint-Solved	Invariant set satisfiability verified.	∃G : Sat(G,I(S)) confirmed; no conflicts.	SPEC_LOCKED (partial)
Generated	Artifact produced by stochastic generator.	Implementation exists; TVG-verified specification used.	CODE_GENERATED
Verified	All mandatory structural and behavioral invariant classes satisfied.	Structural and behavioral invariants satisfied.	VERIFIED (partial)
Proven	Runtime constraints and semantic invariants satisfied.	Resource, temporal, and semantic invariants satisfied.	VERIFIED (partial)
Deployable	Full authorization for deployment granted.	All invariant classes satisfied; evidence package signed.	EVIDENCED (partial)
Runtime-Observed	Continuous invariant monitoring active post-deployment.	Telemetry active; drift detection running; policy enforced.	No equivalent
Closed	Governed lifecycle complete. Closure record archived.	CLOSURE STATUS = COMPLETE; evidence archived.	EVIDENCED (partial)

7.2 Key Transition Guards

Transition	Guard
Generated → Verified	All structural and behavioral invariants satisfied.
Verified → Proven	All security, resource, temporal, and semantic invariants satisfied.
Proven → Deployable	All policy invariants satisfied; evidence package signed with valid provenance.
Deployable →	Deployment registration complete; monitoring infrastructure active.

Runtime-Observed	
Runtime-Observed → Closed	All lifecycle events recorded; no open drift alerts; closure authorized by problem owner.

8. Canonical Intermediate Representation

A canonical semantic Intermediate Representation (IR) forms the computational core of SDPF vNext. Without a canonical IR, SDPF remains document-oriented rather than computationally executable. Every specification passes through the parser into the IR before any downstream operation can begin.

8.1 The IR as Execution Substrate

The IR is not a serialisation format. It is the orchestration substrate through which all SDPF vNext operations are performed. All verification operations execute against the IR, not against the raw specification text.

EXECUTION PIPELINE

SDPF Specification ↓ Parser ↓ Semantic IR ↓ Invariant Extraction ↓ Constraint Graph ↓ Generation Runtime ↓ Verification Engine ↓ Closure Record

8.2 IR Schema — Normative Structure

The following defines the normative IR schema. A conforming IR Parser must produce an object satisfying this schema for every valid SDPF specification. This schema is the authoritative definition — it governs the Phase 2 reference parser implementation defined in the development roadmap (Section 17). Any implementation claiming vNext conformance must produce IR objects that satisfy this schema.

```
IR {
  metadata: {
    spec_id:       string          // unique specification identifier
    spec_version:  string          // e.g. "1.0.0"
    style_id:      integer         // 1-17
    phase0: {
      problem_statement: string
      current_state:     string
      desired_state:     string
      gap:               string
      impact:            string
      problem_owner:     string
      validation: {
        T1_observable: boolean
        T2_bounded:    boolean
        T3_cause_free: boolean
        T4_solution_free: boolean
      }
    }
  }

  invariants: [InvariantNode]

  InvariantNode {
```

```
    id:           string          // e.g. "INV-001"
    source_req:   string          // REQ-ID from specification e.g. "R-001"
    priority:     MANDATORY | ADVISORY
    class:        structural | behavioral | security | temporal | resource |
semantic | policy
    description:  string
    verifiable:   boolean         // can be checked by machine
    test_ids:     [string]        // TEST-IDs that verify this invariant
    regulatory_mappings: [string] // e.g. ["CQC-MDR-2001-S14"] — empty if none
  }

  tvg_entries: [TVGEntry]

  TVGEntry {
    asset:         string          // e.g. "PostgreSQL"
    asserted:      string          // e.g. "15.x"
    command:       string          // e.g. "psql --version"
    pass_condition: string         // e.g. "output begins with 'psql (PostgreSQL)
15'"
    verified:      boolean
    actual_output: string | null
  }

  constraint_graph: {
    nodes:           [string]      // invariant IDs
    edges: [{
      from:          string        // invariant ID
      to:            string        // invariant ID
      type:          dependency | refinement | conflict
    }]
    satisfiable:     boolean
    conflicts:       [ConflictRecord]
  }

  ConflictRecord {
    inv_a:           string        // invariant ID
    inv_b:           string        // invariant ID
    severity:        blocking | warning
    resolution:      string | null
  }

  traceability: [{
    req_id:          string
    test_ids:        [string]
    implementation_artifact: string | null
  }]

  lifecycle_state: Draft | Parsed | Validated | ConstraintSolved |
                   Generated | Verified | Proven | Deployable |
                   RuntimeObserved | Closed

  provenance_chain: [ProvenanceEvent]

  ProvenanceEvent {
    event:     string             // e.g. "TVG_PASSED"
    timestamp: string             // ISO 8601 UTC
    payload:   object             // event-specific data
    signature: string             // HMAC-SHA256
  }
}
```

8.3 IR Example — Minimal Invariant Node

The following shows a single parsed invariant node extracted from a controlled medication dispensing specification (Style 10):

```
{
  "id": "INV-001",
  "source_req": "R-001",
  "priority": "MANDATORY",
  "class": "behavioral",
  "description": "Dispensing event recorded in digital system within 60 seconds of
physical dispensing act.",
  "verifiable": true,
  "test_ids": ["TEST-001", "TEST-004"],
  "regulatory_mappings": ["CQC-MDR-2001-S14"]
}
```

8.4 Constraint Graph

The constraint graph derived from the IR represents all invariants as a directed graph where nodes are invariants and edges represent dependency, refinement, and conflict relations. Constraint satisfiability is verified on this graph before specification locking. Conflict detection operates on this graph. Runtime propagation traverses this graph when invariants change.

9. Closed-Loop Execution Runtime

SDPF vNext transitions from static validation into adaptive governed execution. Verification is not a post-generation inspection step. It is part of runtime orchestration.

9.1 The Execution Cycle

CLOSED-LOOP EXECUTION CYCLE
Specification → Generation → Verification → Failure Analysis → Constraint Refinement → Regeneration → Reverification → Deployment Authorization

In SDPF v1.3.1, verification failure at the CODE_GENERATED stage halts the run and requires practitioner intervention. In vNext, the closed-loop runtime analyses the failure, refines the constraint set, regenerates, and reverifies — within defined bounds.

9.2 Failure Analysis

Failure Class	Resolution
Specification incompleteness	Invariant undefined or ambiguous. Specification refinement required. Human escalation required.
Generation failure	Generator failed to satisfy a well-defined invariant. Constraint refinement and regeneration. Automated.
TVG mismatch	Environment does not match asserted facts. Specification or environment correction. Human escalation required.
Unsatisfiable invariants	Constraint conflict prevents any conforming artifact. Specification correction. Human escalation required.

9.3 Constraint Refinement — Operational Definition

When a generation failure is classified as resolvable, the runtime applies constraint refinement to the relevant IR invariant node before regenerating. Constraint refinement operates as follows:

Step	Action
1. Identify	Locate the invariant node in the IR that the failed test maps to via the traceability matrix.
2. Classify	Determine whether the failure is due to ambiguity in the invariant description, an underspecified edge case, or a missing constraint on an adjacent property.
3. Augment	Add a refinement edge in the constraint graph from the existing invariant to a new, more specific invariant node expressing the missing constraint.

4. Record	Append a CONSTRAINT_REFINED provenance event to the IR with the original invariant ID, the refinement applied, and the cycle number.
5. Regenerate	Submit the updated IR to the generation runtime with the refined constraint set.
6. Reverify	Run the full verification suite against the new artifact. If the refined invariant now passes, proceed. If not, increment the cycle counter and repeat from Step 1.

9.4 Bounded Regeneration

Automated regeneration cycles are bounded. The runtime will not regenerate indefinitely.

Parameter	Specification
Default maximum attempts	3 regeneration cycles per run unless overridden in the specification.
Override mechanism	Declare max_regeneration_attempts in the specification metadata. Must be a positive integer. Maximum permissible override: 10.
Escalation trigger	If maximum attempts are exhausted without achieving conformance, the run fails to VERIFIED and human escalation is triggered with the full regeneration log.
Evidence requirement	All regeneration cycles and constraint refinements recorded in the regeneration_log evidence field regardless of outcome.

10. Verification Hierarchy

SDPF vNext defines six verification assurance levels. An artifact at level N satisfies all invariants required by levels 0 through N.

Level	Name	What It Verifies	Gate Condition
L0	Syntax Validity	Specification grammar valid; no parse errors; required sections present.	Specification parseable.
L1	Structural Completeness	All structural invariants satisfied; dependency graph complete; schema valid.	11 structural checks pass.
L2	Behavioral Correctness	All behavioral invariants satisfied; functional tests pass; input-output contracts met.	All behavioral tests pass.
L3	Security Validation	All security invariants satisfied; access controls verified; sandbox boundaries enforced.	Security gate clears.
L4	Semantic Confidence	Semantic and policy invariants satisfied; business logic verified; regulatory constraints met.	Semantic and policy checks pass.
L5	Runtime Invariant Preservation	Resource and temporal invariants satisfied under observed load; no drift detected during operation.	Runtime monitoring active; no open alerts.

10.2 Deployment Eligibility

Declared Style	Required Assurance Level
Style 10 — Compliance-Driven	L5 required
Style 14 — Dynamic Criticality Extension	L5 required
Style 17 — Domain-Specific Template (clinical, legal, financial)	L5 required
All other styles	L4 minimum; L5 strongly recommended

11. Runtime Governance

Governance does not end at deployment. SDPF vNext defines a runtime governance model that continuously enforces invariants, detects drift, and revokes authorization when conformance is violated.

11.1 Runtime Governance Capabilities

Capability	Definition
Invariant Drift Detection	Continuous monitoring of deployed artifacts against their governing invariant set. Alerts generated when runtime behaviour violates any mandatory invariant.
Runtime Policy Enforcement	Active enforcement of policy invariants — regulatory, organizational, and deployment policies — during operation, not only at verification time.
Deployment Revocation	Authorization withdrawal for artifacts that violate mandatory invariants during runtime. Revocation is automatic for critical security and safety invariant violations.
Telemetry Analysis	Continuous collection and analysis of runtime telemetry against the resource and temporal invariant classes. Trend analysis for predictive drift detection.
Rollback Triggering	Governed rollback to the last verified-and-deployed version when invariant violation cannot be remediated within defined bounds.
Continuous Verification	Periodic re-execution of the verification hierarchy against runtime-observed behaviour. Re-verification frequency defined by declared style and domain.
Provenance Updates	All runtime governance events appended to the provenance chain. The evidence record is a living document, not a snapshot.

11.2 Drift Classification

Drift Class	Response
CRITICAL drift	Mandatory security or safety invariant violated. Automatic deployment revocation. Immediate human escalation.
SIGNIFICANT drift	Mandatory behavioral or resource invariant violated. Automated remediation attempted. Human escalation if remediation fails.
ADVISORY drift	Advisory invariant violated or trend analysis indicates future violation. Warning generated. No automatic action.

11.3 Invariant Propagation After Update

When a specification is updated after deployment, the runtime governance model propagates the updated invariant set to all dependent deployed artifacts. Each affected artifact is re-

evaluated against the updated invariants without redeployment. Artifacts that fail re-evaluation are flagged for remediation or revocation depending on the drift classification.

12. Provenance Architecture

Every governed transition emits evidence artifacts. Provenance is not a post-run summary — it is a live record built incrementally as the governed lifecycle progresses.

12.1 Required Provenance Elements

Element	Definition
Signatures	HMAC-SHA256 provenance signatures per SDPF Language Specification v1.3.1 Section 14.2.
Timestamps	ISO 8601 UTC timestamps for every lifecycle event.
Invariant results	Per-invariant evaluation results for every verification pass.
Dependency lineage	Full graph of specification dependencies and inherited invariants.
Generation metadata	Generator model identifier, prompt export, and generation parameters.
Verification records	Complete Verification Closure Record per SDPF Language Specification v1.3.1 Section 13.4.
Runtime telemetry	Aggregated telemetry from the Runtime-Observed stage.
Closure status	CLOSURE STATUS = COMPLETE or CLOSURE STATUS = REVOKED with reason.

12.2 Provenance Key Management

Provenance signing requires secure key management. The following rules govern key handling in all SDPF vNext governed environments:

Rule	Specification
Key source	HMAC-SHA256 signing key sourced from a secure environment variable or secret management system (e.g. HashiCorp Vault, AWS Secrets Manager, Azure Key Vault). The key shall never be embedded in the evidence package, the specification, or any generated artifact.
Key identifier	Each provenance event records a key identifier — not the key itself. The key identifier allows verification without exposing the key.
Key rotation	Key rotation events must be recorded in the provenance chain with the outgoing key identifier, the incoming key identifier, and the rotation timestamp. Evidence packages signed under the previous key remain valid under their original signature.
Key scope	Each governed environment (development, staging, production) shall use a separate signing key. Cross-environment key sharing is a conformance violation.
Loss or compromise	If a signing key is lost or compromised, all evidence packages signed under

	that key are marked SIGNATURE_UNVERIFIABLE. They are not automatically invalidated — the provenance content remains; the tamper-evidence guarantee is suspended pending re-signing under a recovery key if available.

12.2 vNext Extensions to the Evidence Standard

Field	Presence	Definition	When Present
invariant_algebra	REQUIRED	Complete invariant composition graph including all inheritance, refinement, and conflict resolution records.	Always
runtime_telemetry	CONDITIONAL	Aggregated telemetry supporting L5 assurance claims.	Runtime-Observed stage reached
regeneration_log	CONDITIONAL	All closed-loop cycle records including constraint refinements applied.	Closed-loop regeneration invoked
drift_alerts	CONDITIONAL	All alert records with classification, response action, and resolution status.	Runtime drift detected
revocation_record	CONDITIONAL	Reason, timestamp, and authorization chain.	Deployment revocation triggered
probabilistic_bounds	CONDITIONAL	epsilon value and convergence analysis.	Probabilistic containment analysis performed — research direction only

13. Probabilistic Containment

> **RESEARCH DIRECTION — NOT CURRENT ARCHITECTURE:** *This entire section describes a research direction for SDPF vNext. The formalism in this section is not current operational architecture. No current implementation should treat these targets as operational requirements. Organizations requiring formal probability bounds today should apply multiple independent verification runs and record convergence statistics in the evidence package.*

Future SDPF vNext formalisation extends bounded stochasticity into measurable probabilistic containment. This transforms the conformance boundary from a binary pass/fail criterion into a measurable probability distribution.

13.1 The Formal Target

PROBABILISTIC CONTAINMENT TARGET (RESEARCH)
$P(G \notin C(S)) \leq \varepsilon$ Where: G = generated artifact C(S) = admissible conformance region ε = bounded failure probability ε is bounded by the invariant set tightness and generator capability

13.2 What This Would Introduce

Capability	Definition
Convergence analysis	Measurement of how quickly closed-loop generation converges on the conformance region.
Entropy reduction	Quantification of how much each constraint refinement cycle reduces output entropy.
Measurable admissibility	Transformation of pass/fail conformance into a probability estimate with defined confidence.
Probabilistic governance guarantees	Contractual ε bounds suitable for high-assurance environments.

The Bounded Stochasticity Theorem states the binary condition. Probabilistic containment would extend this into a probability statement across the distribution of outputs. The theorem remains foundational; probabilistic containment adds a measurement framework on top of the binary conformance criterion.

14. Semantic Correctness

Structural correctness does not guarantee semantic correctness. This is an architectural boundary that SDPF vNext defines explicitly rather than obscuring.

14.1 The Structural-Semantic Gap

> A generated artifact may compile, execute, pass all structural invariant checks, satisfy all behavioral test cases, clear all security invariants, and meet all policy requirements — and still implement incorrect business logic. Semantic errors in AI-generated software are, at their root, specification errors.

The current SDPF invariant system addresses semantic correctness through behavioral invariants — specifically through the test suite and observable behavior requirements. If the test suite correctly encodes the business rules, semantic correctness follows from behavioral conformance. If the test suite is incomplete, the gap remains.

The quality of semantic correctness assurance is directly proportional to the quality of the behavioral invariants defined in the specification.

14.2 Future Research Direction

> **RESEARCH DIRECTION — NOT CURRENT ARCHITECTURE:** *The capabilities described in this section are future research directions. None are current SDPF vNext operational capabilities. They are listed here to define the boundary of the current architecture and the direction in which it will be extended.*

> **RESEARCH DIRECTION — NOT CURRENT ARCHITECTURE:** *Observational equivalence, symbolic execution, semantic invariant systems, executable semantics, and behavioral correctness analysis are future research directions. They are not current SDPF vNext capabilities.*

15. Enterprise Execution Architecture

SDPF vNext is designed for regulated and high-assurance environments where governance is not optional and auditability is a compliance requirement.

Requirement	Definition
Auditability	Complete, tamper-evident record of every governance decision, invariant evaluation, and lifecycle transition.
Traceability	Unbroken chain from problem statement through requirement through test through implementation through runtime observation.
Deployment Governance	Formal authorization required for every deployment; authorization record included in evidence package.
Evidence Retention	Evidence packages retained per organizational and regulatory policy; integrity verifiable at any future point.
Compliance Verification	Invariant system maps directly to regulatory requirements; compliance status verifiable from evidence package alone.
Operational Accountability	Every governance action attributed to a named authorized actor or identified automated system component.

15.2 Target Deployment Domains

Domain	Application
Healthcare	Clinical decision support, patient data systems, diagnostic AI, regulatory submission systems.
Finance	Trading systems, risk engines, regulatory reporting, fraud detection.
Aerospace	Flight management software, ground control systems, certification-required avionics.
Defense	Mission-critical systems, logistics AI, security-classified implementations.
Industrial Automation	Process control, safety-critical monitoring, autonomous operations.
Enterprise AI Infrastructure	AI orchestration platforms, model governance systems, AI operations tooling.

16. Relationship to SDPF v1.3.1

SDPF vNext is an architectural extension of SDPF v1.3.1, not a replacement. Every element of the existing specification remains operative.

16.1 What vNext Extends

Element	Extension
Lifecycle state machine	Extended from seven pre-deployment stages to full operational lifecycle including Runtime-Observed.
Evidence standard	Extended with five additional conditional fields for runtime governance data.
Invariant system	Formalized into an algebra with composition, refinement, inheritance, and conflict semantics.
Verification model	Extended from the 11-check structural gate to a six-level assurance hierarchy.
Execution model	Extended from static validation to closed-loop adaptive execution.

16.2 What vNext Does Not Change

Element	Status
Three Core Principles	Specification First, Facts Before Execution, Verification Always — unchanged.
Bounded Stochasticity Theorem	Foundational result — unchanged.
Phase 0	Mandatory problem definition gate — unchanged.
Technical Verification Gate	TVG structure and HALT discipline — unchanged.
17 normative styles	All styles unchanged; style selection protocol unchanged.
Conflict Resolution Protocol	Unchanged.
Existing evidence package schema	vNext adds fields; it does not remove or modify existing fields.

16.3 Migration Path

SDPF v1.3.1 specifications and evidence packages are fully compatible with vNext. A v1.3.1-conforming specification is a valid vNext specification — it will not reach L5 assurance without the Runtime-Observed stage, but it will reach L4 assurance through the existing verification mechanism.

Organizations adopting vNext for regulated deployments should update their specifications to declare the Runtime-Observed stage requirement explicitly and define the invariant monitoring parameters required for L5 assurance.

17. Development Roadmap

> This section is non-normative. It describes the planned construction sequence for the vNext runtime. The architecture defined in Sections 1–16 is the specification. This roadmap is the build plan.

Phase 1 — Formal Semantics

Establish the mathematical foundations: constraint logic formalisation for the invariant algebra; transition semantics for the extended state machine; satisfiability theory and constraint solver integration; probabilistic containment initial formalisation.

Phase 2 — Execution Infrastructure

Build the computational substrate: semantic IR specification and reference parser conforming to the schema in Section 8.2; constraint graph engine with conflict detection; verification runtime implementing the six-level assurance hierarchy; provenance ledger with vNext evidence schema.

Phase 3 — Autonomous Governance

Implement the runtime governance layer: closed-loop execution runtime with bounded regeneration; runtime invariant monitoring and drift detection; automated remediation and rollback triggering; continuous verification scheduling.

Phase 4 — Semantic Intelligence

Extend toward semantic correctness: symbolic execution engine integration; semantic equivalence verification; behavioral correctness analysis; proof-aware execution and proof-carrying generation.

18. Strategic Positioning

CORRECT POSITIONING

SDPF vNext is deterministic governance infrastructure for stochastic execution systems. Not: deterministic AI. Not: AI that always produces the same output. Not: a replacement for human judgment about requirements. Not: a guarantee of semantic correctness independent of specification quality.

The framework governs operational admissibility. It does not eliminate probabilistic generation — it constrains admissible outputs within formally governed acceptance boundaries. The governance is deterministic. The generation remains probabilistic. Both facts are true simultaneously.

18.1 Where vNext Applies

Condition	vNext Applicability
AI-generated software must satisfy formal requirements verifiably	YES
Auditability of the generation process is required by regulation or operational risk	YES
Post-deployment conformance enforcement is required beyond initial verification	YES
Multiple model comparison is required as evidence rather than assumption	YES

18.2 What vNext Is Not

Claim	Correction
A substitute for well-defined requirements	The specification must correctly encode the business rules. vNext enforces what is specified.
A guarantee of adequate test	Behavioral invariants are only as good as the test suite.

coverage	
A substitute for model capability	The theorem does not hold if the model cannot satisfy the invariants.
A deterministic AI system	Generation remains probabilistic. Governance is deterministic.

19. Final Statement

SDPF vNext defines a governed execution architecture for stochastic AI systems.

The framework's core innovation is not deterministic generation. The core innovation is deterministic operational governance around probabilistic generation systems — through invariant-centred admissibility, verification-aware execution, adaptive closed-loop remediation, continuous runtime conformance, and lifecycle-governed orchestration.

The foundational theorem established in SDPF v1.3.1 remains the load-bearing element. Every vNext extension derives from it. The Bounded Stochasticity Theorem says that conformance to a specification is a property of invariant satisfaction, not surface-form reproduction. vNext takes that insight and applies it to the full operational lifecycle: not only does a generated artifact need to satisfy invariants at verification time — it needs to preserve them through deployment and throughout runtime.

ARCHITECTURAL SUMMARY

SDPF vNext extends specification-as-governance-contract from the pre-deployment phase into the full operational lifetime. The specification is the governance contract. The invariant algebra defines what the contract means. The closed-loop runtime enforces the contract during generation. The verification hierarchy measures the assurance provided. The runtime governance model preserves the contract after deployment. The provenance architecture makes everything auditable. The architecture positions specifications as executable governance contracts for AI-native systems.

Problem first. Specification second. Facts before execution. Verification always.

Hamza Abdullah · Software Development Prompting Framework · 2026