

SDPF

Software Development Prompting Framework

Language Specification

Version 1.3.1

Unified Edition — Patch 1: Key Terms Quick Reference · Human Escalation Justification · Narrative Flow

Hamza Abdullah
Creator and Principal Author
2026

Copyright and Licence Notice

Version History

Version	Date	Changes
1.0	2026	First draft — initial formal specification of vocabulary, grammar, 17 styles, lifecycle, verification, evidence, and conformance.
1.1	2026	TVG elevated to Core Principle II. Automated Conflict Resolution Protocol added. Bounded Stochasticity Theorem and Structural Invariance proof added. Human escalation removed throughout. CodeJuicer demonstration added.
1.2	2026	Problem Identification and Definition Procedure added as mandatory Phase 0. SDPF cycle may not begin until a validated problem statement exists. Lifecycle updated. LSS DMAIC added to normative references.
1.3	2026	Unified edition. TVG elevated to top-level Section 2. All normative content from v1.0 through v1.2 consolidated. No normative content removed or altered. Phase 0 completed before production.
1.3.1	2026	MINOR patch. Section 1.5 Key Terms Quick Reference added. Section 10.1 human escalation justification added. Foreword reading guide added. No normative content changed.

How to Cite This Specification

Abdullah, H. (2026). SDPF Language Specification, Version 1.3.1. Software Development Prompting Framework.

Foreword

The Software Development Prompting Framework (SDPF) was created by Hamza Abdullah in 2025 in response to a problem the AI industry has systematically misidentified.

The problem is not the AI model. The problem is the specification.

When a specification is incomplete or contains unverified technical facts, an AI model fills the gaps with its best guess. The output looks plausible. It is not what was intended. What the industry calls hallucination is, in the overwhelming majority of practical cases, specification-induced speculation. Better models make better guesses. They are still guessing. The correct intervention is not a better model. It is a complete specification — one where there is nothing to guess.

SDPF is the formal language that makes complete specifications possible. It defines the vocabulary, grammar, semantics, and enforcement mechanisms that eliminate ambiguity at the source. A specification written in SDPF gives an AI model nothing to speculate about. The output is not a better guess. It is a mechanical execution of a verified contract.

This document is the authoritative definition of that language.

How to Read This Document

New practitioners should begin at Section 1.5 — Key Terms Quick Reference — which defines the most important acronyms and concepts in plain language before the normative sections begin. From there:

- Section 2 defines the Technical Verification Gate (TVG) — the most critical enforcement mechanism in SDPF and Core Principle II.
- Section 3 defines Phase 0 — the mandatory problem identification gate that precedes all specification work.
- Section 4 shows the complete SDPF lifecycle — how Phase 0 and Phase 1 fit together.
- Section 5 states the three core principles that govern every SDPF-conforming process.
- Sections 9–11 define the grammar, conflict resolution, and style system — the operational language of SDPF.
- Sections 12–15 define the lifecycle protocol, verification model, evidence standard, and conformance requirements.

> **PHASE 0 PROBLEM STATEMENT (this document):** "The SDPF Language Specification exists across 3 fragmented version files with no unified authoritative document, against a target of 1 complete unified specification, resulting in practitioners being unable to reliably implement or certify SDPF conformance from a single source."
> **Problem Owner: Hamza Abdullah | CLOSURE STATUS: COMPLETE**

Closing principle: Problem first. Specification second. Facts before execution. Verification always.

Hamza Abdullah — Creator and Principal Author, SDPF

1 Scope

1.1 Purpose

This specification defines the SDPF language with sufficient precision that any practitioner, tool developer, certification body, or standards organisation can implement, evaluate, or adopt SDPF without access to the original author.

1.2 What This Specification Defines

■ The Technical Verification Gate (TVG) — mandatory verification of every asserted technical fact before implementation begins. Defined in full in Section 2.

■ The Problem Identification and Definition Procedure — mandatory Phase 0 of the SDPF lifecycle. Defined in Section 3.

■ The three core principles governing every SDPF-governed system. Defined in Section 5.

■ The normative vocabulary of the SDPF language. Defined in Section 8.

■ The grammar of a valid SDPF specification document. Defined in Section 9.

■ The 17 SDPF styles as formally defined dialects. Defined in Section 11.

■ The Conflict Resolution Protocol governing requirement priority precedence. Defined in Section 10.

■ The Bounded Stochasticity Theorem and the structural invariance proof. Defined in Section 6.

■ The SDPF lifecycle protocol including Phase 0. Defined in Section 12.

■ The verification model including the eleven structural invariant checks. Defined in Section 13.

■ The evidence standard including provenance signing. Defined in Section 14.

■ Conformance requirements for specifications, tools, evidence packages, and processes. Defined in Section 15.

1.3 What This Specification Does Not Define

■ The implementation of any specific SDPF-conforming tool.

■ The AI model or models to be used with SDPF specifications.

■ The programming language or platform of the implementation produced.

■ The semantic correctness of AI-generated implementations — SDPF governs the specification and verification process, not the correctness of generated logic.

1.4 Intended Audience

Tool developers, architects, standards organisations, certification bodies, and researchers. Practitioners who wish to use SDPF without building tools should consult the SDPF User Manual and Framework Tutorial.

1.5 Key Terms Quick Reference

The following table defines the most important acronyms and concepts used throughout this specification. Each entry includes a plain-language explanation and a cross-reference to the section where the full normative definition is found. This section is informative.

Term / Acronym	Plain-Language Meaning	See
TVG Technical Verification Gate	Before any code is generated, every technical fact stated in the specification — tool versions, API endpoints, file paths, environment variables — must be verified by running a command in the actual target environment. If the output does not match what the specification asserts, work stops and the specification is corrected. The TVG ensures the AI receives a specification where every fact is true.	Section 2
Phase 0	The mandatory first phase of every SDPF project. Before writing a single requirement, the practitioner must define the problem: what is the measurable current state, what is the desired state, what is the quantified gap, and what is the business impact. No causes. No solutions. A validated problem statement is the only output. Without it, Phase 1 cannot begin.	Section 3
TSP Technical Specification Prompt	The complete SDPF document submitted to an AI model for execution. It is a locked, TVG-verified contract containing no ambiguity. The AI executes it mechanically — it has nothing to guess.	Section 8
REQ-ID	A unique identifier (e.g. R-001) automatically assigned to every requirement when the specification is locked. Every function in the generated code must reference its REQ-ID. This creates an unbroken chain from business requirement to implementation to test.	Section 8
Bounded Stochasticity	AI models produce different outputs from the same input. SDPF resolves this by specifying what the output must compute (structural invariants) rather than how it must look (surface form). Any output that satisfies the invariants is conforming — regardless of variable names, code structure, or documentation style.	Section 6
LSS DMAIC Lean Six Sigma Define-Measure-Analyse-Improve-Control	The industry-standard problem-solving methodology that underpins Phase 0. SDPF adopts the LSS principle that a correctly defined problem is the measurable gap between current state and desired state, with quantified impact, containing no assumed cause and no assumed solution.	Sections 3 & 7
Automated Conflict Resolution	When two requirements conflict, SDPF resolves the conflict automatically by strictness priority: [CRITICAL] overrides [REQUIRED], which overrides [OPTIONAL]. Only a conflict between two [CRITICAL] requirements requires the practitioner to intervene — and only at specification time, not at execution time. See Section 10.1 for the full justification for removing human escalation.	Section 10
Evidence Package	The signed artifact produced at the end of a verified run. Contains the locked specification, all tests, the implementation hash, the verification closure record, and a cryptographic provenance signature. It constitutes tamper-evident, auditable proof that all three core principles were satisfied.	Section 14

Term / Acronym	Plain-Language Meaning	See
Provenance Signature	An HMAC-SHA256 cryptographic signature over the entire evidence package. Any modification to the package after signing invalidates the signature, making the evidence tamper-evident.	Section 14.2
Style	One of 17 formally defined dialects of SDPF. Each style is optimised for a specific class of system. Examples: Style 1 (APIs and deterministic systems), Style 9 (Maintenance/Refactor), Style 10 (Compliance-driven systems such as HIPAA or PCI-DSS). Style selection is the first decision in Phase 1.	Section 11
CRITICAL / REQUIRED / OPTIONAL	The three priority tags that every requirement must carry. CRITICAL = system fails or does not close the problem gap without this. REQUIRED = system is incomplete without this. OPTIONAL = enhancement only. These tags govern automated conflict resolution and determine what must be in the traceability matrix.	Section 8.4

2 Technical Verification Gate (TVG)

The Technical Verification Gate is Core Principle II of SDPF and a mandatory top-level section of every SDPF-conforming specification. No implementation begins until every asserted technical fact in the specification has been verified against live output. A specification that has not passed the TVG is not ready for execution.

> **CORE PRINCIPLE II — Facts Before Execution:** No implementation begins until every asserted technical fact in the specification is verified against live output. A specification that has not passed the TVG is not ready for execution.

2.1 What the TVG Is

The TVG is a mandatory section in every SDPF specification that lists every tool version, library version, API endpoint, environment variable, flag, path, or configuration value asserted as fact anywhere in the specification. For each asserted fact, the TVG provides a verification command that can be executed in the target environment and a pass condition defining the expected output. If the output does not match the pass condition, work halts.

The TVG is not a post-implementation check. It is a pre-implementation gate. It runs before any code is generated. Its purpose is to ensure that the AI receives a specification where every stated technical fact is true in the actual environment the implementation will run in.

2.2 Why the TVG Is Mandatory

A specification can be complete — every section present, every requirement tagged, every exception named — and still contain technical facts that are wrong. Wrong version numbers. Deprecated API endpoints. Changed flag behaviour. An AI that receives a complete but factually incorrect specification will execute faithfully and produce a factually incorrect implementation. The specification failure is not visible in the output. It surfaces only when the implementation runs in the actual environment.

> **THE PRINCIPLE:** The TVG does not improve AI capability. It removes the category of failure that occurs when AI executes a specification that is complete but factually incorrect with respect to the environment. The AI cannot hallucinate around a fact it was never given incorrectly.

2.3 TVG Structure

Each entry in the TVG section of a specification shall follow this structure:

```
Tool / Asset: [name] Asserted Value: [the value stated in the specification] Verification Command: [executable command in the target environment] Pass Condition: [exact expected output or pattern] HALT: [action if check fails]
```

2.4 The CodeJuicer Demonstration

CodeJuicer is a real Python desktop application built under SDPF. At version 2 of its specification, the following TVG entry was added after a build failure caused by a Python version mismatch:

```
Tool: Python Asserted Version: 3.11.15 Verification: python --version Pass Condition: output = "Python 3.11.15" HALT: Do NOT proceed under any other version. Tool: UPX
```



```
Verification: upx --version Pass Condition: version string returned HALT: Do NOT proceed
if UPX is not installed. Entry Point: main.py Verification: python main.py Pass Condition:
application launches, all features functional HALT: Do NOT proceed if entry point fails.
```

Before the AI generated any build script, the practitioner ran each verification command. If the Python version returned was not 3.11.15, work stopped. The specification was updated to reflect the actual environment. Only after all TVG entries passed did the AI receive the specification. The build succeeded on the first submission from that version forward.

2.5 TVG Normative Rules

TVG-1 Every tool version, library version, API endpoint, flag, path, and environment variable asserted in the specification shall have a TVG entry.

TVG-2 Each TVG entry shall contain: the asset name and asserted value, a verification command executable in the target environment, a pass condition defining the expected output, and a HALT rule.

TVG-3 If any TVG check fails, the specification shall be updated before implementation proceeds. Work shall not continue around a TVG failure.

TVG-4 A conforming tool shall surface TVG failures as blocking errors.

TVG-5 A specification shall not be locked until all TVG entries have been verified against live output and have passed.

2.6 TVG in the Specification Grammar

The TVG appears as section S-08 in the required sections of every SDPF specification (see Section 9). It is also referenced in:

- Section 5 — The Three Core Principles: TVG is Principle II.
- Section 9.1 — SPEC_LOCKED gate: specification shall not be locked until all TVG entries pass.
- Section 12.3 — GATE-3: TVG passage is a locking precondition.
- Section 13.3 — G1: TVG is the first of the five verification gates.
- Section 15.2 — SPEC-C-5: TVG conformance requirement for specifications.

3 Phase 0 — Problem Identification and Definition

Phase 0 is the mandatory first phase of every SDPF-governed project. Its purpose is to produce a validated problem statement that serves as the direct input to the SDPF specification. Phase 0 cannot be skipped, abbreviated, or deferred.

> **LEAN SIX SIGMA FOUNDATION (LSS DMAIC):** A problem, correctly defined, is the measurable gap between the current state and the desired state, with a quantified impact. A problem statement contains no cause and no solution. Cause and solution are outputs of the SDPF specification phase, not inputs to it. LSS DMAIC = Lean Six Sigma Define-Measure-Analyse-Improve-Control — the industry-standard problem-solving methodology on which Phase 0 is based.

3.1 Why Phase 0 Is Mandatory

A specification is the translation of a problem into a formal contract. If the problem is undefined, the specification has no foundation. The AI will execute faithfully and produce output correct relative to the specification — but it will not solve the actual problem because the actual problem was never stated. This is the most expensive failure mode in software development. The system works. It does not do what was needed.

3.2 The Problem Statement Structure

Component	Definition	Must Be	Must Not Be
Current State	What is happening that should not be, or what is not happening that should be.	Observable and measurable. Stated in present tense.	An assumption, opinion, or cause.
Desired State	The target condition. What good looks like.	Specific and testable.	Vague. 'Better' without a measure is not a desired state.
Gap	The measurable difference between current state and desired state.	Quantified. A number, frequency, percentage, or time.	Qualitative only.
Impact	The business consequence of the gap existing.	Cost, risk, time, or compliance terms.	A technical description.

3.3 The Seven Steps of Phase 0

Step 1 — Observe the Current State

Go to where the problem occurs. Do not assume. Do not theorise. Measure what is actually happening. Document only what can be observed and measured. No causes. No solutions.

Step 2 — Define the Desired State

State the target condition precisely and measurably. "Better" is not a desired state. "Zero correction cycles on first AI submission" is a desired state.

Step 3 — Quantify the Gap

Measure the distance between current state and desired state. Express the gap as a number. A gap that cannot be measured cannot be verified as solved.

Step 4 — Establish the Impact

State the business consequence of the gap existing. Impact justifies investment. Without a stated impact the problem has no urgency.

Step 5 — Write the Problem Statement

Combine Steps 1–4: "[Current state] against a target of [desired state], resulting in [impact]." No cause. No solution. Every element measurable.

Step 6 — Validate the Problem Statement

Apply the four validation tests defined in Section 3.4. All four tests must pass before proceeding.

Step 7 — Confirm the Problem Owner

Identify the person who experiences the gap and is accountable for closing it. A project without an identified problem owner shall not proceed to Phase 1.

3.4 The Four Validation Tests

Test	Name	Pass Condition	Fail Condition
T-1	Observable	The problem can be seen or measured in the real world.	Statement describes an opinion, assumption, or internal belief that cannot be externally verified.
T-2	Bounded	The statement defines where the problem starts and ends. Scope is clear.	The problem is unbounded — applies to everything or everywhere without limits.
T-3	Cause-Free	The statement contains no language that implies why the problem exists.	Words like 'because', 'due to', 'caused by', or any named cause appear.
T-4	Solution-Free	The statement contains no language that implies how the problem will be solved.	Words like 'by implementing', 'using', 'through', or any named solution appear.

3.5 Phase 0 Normative Requirements

P0-1 A SDPF-conforming process shall not begin Phase 1 until Phase 0 has produced a validated problem statement that passes all four validation tests.

P0-2 The problem statement shall contain all four components: current state, desired state, gap, and impact.

P0-3 The gap component shall be expressed quantitatively. A qualitative gap description does not satisfy this requirement.

P0-4 The problem statement shall contain no cause and no solution.

P0-5 A problem owner shall be identified before Phase 1 begins.

P0-6 The validated problem statement shall be referenced in the SDPF specification as the basis for the External Contract and Output Guarantees sections.

P0-7 A conforming tool shall provide a Phase 0 checklist and shall not permit specification locking until the practitioner has confirmed Phase 0 completion.

P0-8 The validated problem statement shall be included in the evidence package.

3.6 How the Problem Statement Maps to the Specification

Problem Statement Component	Maps To	Rationale
Current State	Processing Rules (S-05)	The current state describes what the system does wrong. Processing rules define the correct behaviour that replaces it.
Desired State	Output Guarantees (S-06)	The desired state is what good looks like. Output guarantees are the formal contract the system will produce that state.
Gap	CRITICAL Requirements	Requirements that close the gap are CRITICAL — without them the system does not solve the problem.
Impact	Exception Handling (S-07) and Scope	The impact defines what happens when things go wrong.
Problem Boundary	External Contract (S-03) and Style Context (S-02)	The boundary of the problem defines the scope of the system.
Desired State (measurable)	Verification Requirements (S-09)	The measurable desired state defines what a passing test looks like.

3.7 Example — Correct Problem Definition

Current State: AI-generated code requires an average of 4.2 correction cycles per feature.

Desired State: AI-generated code meets developer requirements on first submission with zero correction cycles.

Gap: 4.2 correction cycles per feature above the target of 0.

Impact: 336 hours of rework per quarter.

> "AI-generated code requires an average of 4.2 correction cycles per feature, against a target of 0 correction cycles, resulting in 336 hours of rework per quarter." — T-1 ✓ T-2 ✓ T-3 ✓ T-4 ✓

4 The SDPF Lifecycle

The SDPF lifecycle consists of two phases and seven stages. Phase 0 must be completed before Phase 1 begins. No stage in Phase 1 may begin until Phase 0 produces a validated problem statement.

> **INVARIANT:** A specification cannot be written until the problem is correctly identified and defined. A SDPF specification written without a validated problem statement is solving an undefined problem.

Phase	Stage	Description	Gate
Phase 0	P0-1: Observe	Measure the current state against the desired state.	Observable, measurable gap confirmed.
Phase 0	P0-2: Define	Write and validate the problem statement.	Problem statement passes all four validation tests.
Phase 0	P0-3: Own	Confirm the problem owner.	Problem owner identified and accountable.
Phase 1	S-1: TSP_DRAFT	Write the specification from the validated problem statement.	All required sections present.
Phase 1	S-2: SPEC_LOCKED	Lock the specification. TVG verified. Conflicts resolved.	All TVG entries pass. No CRITICAL-CRITICAL conflicts.
Phase 1	S-3: TESTS_GENERATED	Generate tests from locked specification.	Tests exist with REQ-ID traces.
Phase 1	S-4: TESTS_LOCKED	Lock the test suite.	Tests frozen. Implementation gate open.
Phase 1	S-5: CODE_GENERATED	Generate implementation from locked tests.	Implementation exists.
Phase 1	S-6: VERIFIED	Run verification gate — all 11 structural invariant checks pass.	CLOSURE STATUS = COMPLETE.
Phase 1	S-7: EVIDENCED	Export signed evidence package.	Signed evidence archived.

5 The Three Core Principles

Every SDPF-governed system is governed by three co-equal, non-negotiable principles. These principles are not guidelines. They are structural requirements. A process that violates any one of them is not a SDPF-conforming process.

PRINCIPLE I — Specification First

No implementation begins until the contract is complete, locked, and TVG-verified.

PRINCIPLE II — Facts Before Execution (The Technical Verification Gate)

No implementation begins until every asserted technical fact in the specification is verified against live output. A specification that has not passed the TVG is not ready for execution. See Section 2 for the full TVG specification.

PRINCIPLE III — Verification Always

No release is permitted without a complete Verification Closure Record signed by a valid provenance key.

5.1 Why Three Principles, Not One

The original formulation of SDPF stated a single invariant: specification first, code second, verification always. This was correct but incomplete — it addressed completeness but not correctness. A specification can satisfy Principle I completely and still contain technical facts that are wrong. Principle II — the TVG — closes this gap. The three principles together — completeness, factual correctness, verified closure — constitute the minimum sufficient conditions for reliable AI-governed software development.

5.2 Normative Requirements for the Three Principles

PRIN-1 A conforming SDPF process shall not begin Phase 1 without a validated problem statement from Phase 0.

PRIN-2 A conforming SDPF process shall not begin implementation until the specification is locked and all TVG entries have passed verification against live output.

PRIN-3 A conforming SDPF process shall not release without a Verification Closure Record with CLOSURE STATUS = COMPLETE, signed by a valid provenance key.

PRIN-4 A conforming tool shall enforce all three principles as blocking gates, not advisory warnings.

6 Bounded Stochasticity and Structural Invariance

This section resolves a fundamental apparent contradiction in SDPF: the methodology is deterministic, but the executor — an AI model — is stochastic. If the same prompt submitted twice does not guarantee the same output, how can a deterministic specification guarantee reliable results?

6.1 The Problem Stated Precisely

An AI language model is a stochastic function. Given identical inputs at different times, it may produce outputs that differ in surface form. A SDPF specification is a deterministic contract. Given input X, the output shall be Y. The apparent contradiction: if the executor is stochastic, how can it satisfy a deterministic contract consistently?

6.2 Surface Form vs. Structural Properties

Property Category	Definition	Example	SDPF Relevance
Surface Form	How the output is expressed — variable names, code style, documentation phrasing.	Function named process_user vs handle_user_creation	Not normatively required. Stochastic variation is acceptable.
Structural Properties	What the output computes — the mapping from inputs to outputs, error conditions handled, requirements satisfied.	POST /user returns HTTP 201 with user object on success	Normatively required. Must be invariant across stochastic variation.

6.3 The Bounded Stochasticity Theorem

> **THEOREM:** Let S be a complete SDPF specification defining a set of 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. The stochastic variation in G is bounded by I — any variation that satisfies I is a conforming realisation of S.

A SDPF specification defines structural invariants over the input-output behaviour of G, not over its surface form. Two implementations that differ entirely in surface form but satisfy the same invariants are both conforming realisations of S.

6.4 Practical Implications

- AI output does not need to be deterministic to be reliable. It needs to be structurally invariant. Evaluate AI output by whether it satisfies the specification's structural invariants, not by whether it matches a specific expected surface form.
- The verification gate is not a process compliance check. It is a structural invariant test.
- A tighter specification defines tighter invariant bounds and produces more consistent AI output across multiple submissions.

6.5 When the Theorem Does Not Hold

- The specification is incomplete — missing sections leave invariants undefined.

■ The TVG has not been completed — unverified technical facts introduce factual errors.

■ The AI model lacks sufficient capability to satisfy the specified invariants.

> **NOTE:** The theorem does not guarantee correctness of the AI implementation — it guarantees structural conformance with the specification. Correctness remains the responsibility of the practitioner's review.

7 Normative References

Reference	Document
IEEE 29148:2018	ISO/IEC/IEEE 29148 — Systems and software engineering — Life cycle processes — Requirements engineering
ISO/IEC 25010:2023	Systems and software engineering — SQuaRE — Product quality model
NIST AI RMF 1.0	NIST Artificial Intelligence Risk Management Framework, 2023
RFC 2119	Key words for use in RFCs to indicate Requirement Levels, IETF, 1997
FIPS 198-1	The Keyed-Hash Message Authentication Code (HMAC), NIST, 2008
LSS DMAIC	Lean Six Sigma Define-Measure-Analyse-Improve-Control methodology — problem definition standards. The foundational methodology for Phase 0 problem identification and validation.

8 Terms and Definitions

Normative terms are defined in this section. Where a term is used normatively in this specification, its meaning is as defined here. For a plain-language quick reference of key acronyms, see Section 1.5.

8.1 Phase 0 Terms

problem statement — A formally validated statement of the gap between current state and desired state with a quantified impact. Contains no cause and no solution. The mandatory input to the SDPF specification.

current state — The observable, measurable condition that exists before the system is built.

desired state — The target condition. The specific, measurable definition of what good looks like. The basis for SDPF Output Guarantees and Verification Requirements.

gap — The quantified difference between current state and desired state. Must be expressed as a number, frequency, percentage, or time measure.

impact — The business consequence of the gap existing. Expressed in cost, risk, time, or compliance terms. Never a technical description.

problem owner — The person who experiences the gap, is accountable for closing it, and is the primary stakeholder for the SDPF specification.

8.2 Core Language Terms

SDPF — Software Development Prompting Framework. The formal specification language defined by this document.

specification — A SDPF document defining the complete intended behaviour of a software system, written from a validated problem statement, before any tests or implementation exist.

TSP — Technical Specification Prompt — The canonical name for a SDPF specification document as submitted to an AI model. A TSP is a complete, locked, TVG-verified contract.

requirement — A single, tagged, named statement of intended behaviour. Every requirement shall begin with a priority tag and shall produce a REQ-ID when the specification is locked.

REQ-ID — A unique persistent identifier assigned to a requirement when a specification is locked. Form: R-NNN where NNN is a zero-padded decimal integer beginning at 001.

style — A named dialect of the SDPF language governing semantic emphasis, required sections, verification focus, and AI framing for a specific class of system. 17 normative styles are defined in Section 11.

structural invariant — A property of a SDPF-conforming implementation that must hold regardless of surface form variation.

bounded stochasticity — The property of AI-generated output constrained by structural invariants such that surface-form variation is acceptable while structural conformance is required. Defined formally in Section 6.

8.3 Specification Section Terms

External Contract — Declares the protocol, data format, version, and interface type.

Input Contract — Defines all valid inputs with field names, types, constraints, and required/optional classification.

Processing Rules — Defines deterministic logic applied to inputs. Shall contain no ambiguous or implicit behaviour.

Output Guarantees — Defines all system outputs for all input conditions covered by the Input Contract.

Exception Handling — Defines the system response to every reachable error condition. Each exception shall be named, coded, and shaped.

TVG — Technical Verification Gate — A mandatory section (S-08) listing every asserted technical fact with a verification command and pass condition. All TVG entries shall pass before implementation begins. Full specification in Section 2.

8.4 Priority Tag Terms

[CRITICAL] — The highest priority tag. A CRITICAL requirement must be satisfied for the system to be safe, correct, or non-failing. System fails or does not close the problem gap without it. Conflict between two CRITICAL requirements is a specification error that blocks locking.

[REQUIRED] — A REQUIRED requirement must be satisfied for the system to be complete and releasable. In conflict with CRITICAL, REQUIRED yields automatically.

[OPTIONAL] — An OPTIONAL requirement is an enhancement. The system closes the problem gap without it. In conflict with CRITICAL or REQUIRED, OPTIONAL yields automatically.

8.5 Conflict Resolution Terms

strictness priority — The precedence order of priority tags for automated conflict resolution: [CRITICAL] > [REQUIRED] > [OPTIONAL].

specification error — A conflict between two [CRITICAL] requirements that cannot be resolved by strictness priority. Blocks locking until the practitioner resolves the conflict explicitly.

automated conflict resolution — The mechanism by which a conforming tool resolves conflicts between requirements of different priority tags without human intervention, using strictness priority precedence. Defined in Section 10.

8.6 Lifecycle Stage Terms

TSP_DRAFT — Phase 1 initial stage. Specification is being written from the validated problem statement.

SPEC_LOCKED — Specification locked, validated, TVG completed, requirements parsed, REQ-IDs assigned.

TESTS_GENERATED — Tests generated from locked specification.

TESTS_LOCKED — Test suite frozen. Implementation gate open.

CODE_GENERATED — Implementation generated from locked tests and specification.

VERIFIED — All eleven structural invariant checks pass. Evidence export permitted.

8.7 Change Protocol Terms

BREAKING change — Modifies, removes, or replaces a CRITICAL requirement or output contract. Requires full re-implementation and re-verification.

SIGNIFICANT change — Modifies a REQUIRED requirement or adds a new requirement. Requires re-verification of affected components.

MINOR change — Clarifies language, updates TVG entries, or amends informative text without altering any normative requirement. Requires review only.

8.8 Verification and Evidence Terms

verification gate — The set of eleven structural invariant checks that shall all pass before a run may advance to VERIFIED stage.

traceability — The documented chain from each REQ-ID to the tests that verify it and the implementation artifacts that realise it.

evidence package — The signed, structured artifact produced at the end of a VERIFIED run. Constitutes auditable proof that all three core principles and Phase 0 were satisfied. Schema defined in Section 14.

Verification Closure Record — Records all structural invariant check results. CLOSURE STATUS must be COMPLETE before release.

provenance signature — HMAC-SHA256 signature over the canonical evidence package payload making it tamper-evident. Standard defined in Section 14.2.

9 Grammar of a SDPF Specification

9.1 Required Sections

Section	Name	Normative Requirement
S-01	Style Declaration	Shall identify the SDPF style by ID and name before locking.
S-02	Style Context	Shall declare domain scope and system boundary derived from the problem statement.
S-03	External Contract	Shall declare protocol, format, and interface type. Shall reference the problem boundary.
S-04	Input Contract	Shall define all valid inputs with types and constraints.
S-05	Processing Rules	Shall define all processing logic without ambiguity. Shall address the current state from the problem statement.
S-06	Output Guarantees	Shall define all outputs for all covered input conditions. Shall formally express the desired state from the problem statement.
S-07	Exception Handling	Shall define every error condition. Shall cover failure modes that would allow the problem gap to persist.
S-08	Technical Verification Gate	Shall list all asserted technical facts with verification commands and pass conditions per Section 2. All entries shall pass before implementation begins.
S-09	Verification Requirements	Shall list all tests with pass conditions. Shall include tests that verify the desired state is achieved.
S-10	Traceability Matrix	Shall map every REQ-ID to at least one TEST-ID.

> **NOTE:** The validated problem statement shall be referenced in the specification header or as a preamble to S-03.

9.2 Requirement Syntax

```
requirement ::= priority-tag SP text
priority-tag ::= "[CRITICAL]" | "[REQUIRED]" | "[OPTIONAL]"
text ::= one or more non-empty characters describing intended behaviour
```

REQ-SYNTAX-1 Every requirement shall begin with a priority tag.

REQ-SYNTAX-2 Requirement text shall be specific enough to produce a testable structural invariant.

REQ-SYNTAX-3 Requirement text shall not contain ambiguous terms such as "appropriate", "reasonable", "as needed", or "and/or" without additional specificity.

REQ-SYNTAX-4 Each requirement shall address a single testable condition.

REQ-SYNTAX-5 CRITICAL requirements shall directly address the gap identified in the problem statement.

REQ-SYNTAX-6 On locking, each requirement shall be assigned REQ-ID R-NNN beginning at R-001.

9.3 Section Content Rules

9.3.1 External Contract

EC-1 Shall contain the interface type.

EC-2 Shall contain the data serialisation format.

EC-3 Shall contain the version identifier.

9.3.2 Input Contract

IC-1 Shall enumerate all valid input types or operations.

IC-2 Shall specify field names, types, and nullability for each input.

IC-3 Shall specify constraints on value range, length, format, or content for each input field.

IC-4 Shall classify each field as required or optional from the caller's perspective.

9.3.3 Processing Rules

PR-1 Shall be a complete, deterministic description of logic for every valid input.

PR-2 Every conditional branch shall have an explicit corresponding action.

PR-3 All defaults shall be explicitly stated.

PR-4 No undefined states — every state the system can reach shall be described.

9.3.4 Output Guarantees

OG-1 Shall define the complete structure of every success response.

OG-2 Shall specify HTTP status codes or equivalent response identifiers for every outcome.

OG-3 Shall provide a guarantee for every input condition in the Input Contract.

9.3.5 Exception Handling

EH-1 Shall contain a named entry for every reachable error condition.

EH-2 Each entry shall contain a machine-readable error code, a human-readable message description, and the response shape.

EH-3 Shall specify the HTTP status code or equivalent for each error condition.

EH-4 Shall define recovery or escalation path for each CRITICAL error condition.

9.3.6 Technical Verification Gate (S-08)

TVG-1 Every tool version, library version, API endpoint, flag, path, and environment variable asserted in the specification shall have a TVG entry.

TVG-2 Each TVG entry shall contain: the asset name and asserted value, a verification command executable in the target environment, a pass condition defining the expected output, and a HALT rule.

TVG-3 If any TVG check fails, the specification shall be updated before implementation proceeds. Work shall not continue around a TVG failure.

TVG-4 A conforming tool shall surface TVG failures as blocking errors.

TVG-5 A specification shall not be locked until all TVG entries have been verified against live output and have passed.

9.3.7 Traceability Matrix

TM-1 Shall contain at least one mapping row for every CRITICAL requirement.

TM-2 Shall contain a mapping row for every REQUIRED requirement.

TM-3 Format: REQ-ID -> TEST-ID[, TEST-ID, ...] -> implementation-artifact-identifier

The SDPF normative error schema for REST APIs:

```
{ "error": { "code": string, "message": string, "field": string | null } }
```


10 Conflict Resolution Protocol

When two or more requirements conflict — when satisfying one makes it impossible to satisfy another — the conflict shall be resolved by the Conflict Resolution Protocol defined in this section.

10.1 Why Human Escalation Was Removed

Prior to version 1.1, SDPF specifications referenced human escalation paths — the ability for an operator to intervene at execution time when a conflict arose between requirements. This was removed in v1.1 and replaced with automated structural resolution. The following five justifications are normative: each identifies a specific failure mode that human escalation introduces and explains why automated resolution eliminates it.

1. Human escalation at execution time is a specification failure, not a resolution.

If a conflict requires human intervention at the point of AI execution, the specification was not complete when it was locked. A conflict not anticipated and resolved before execution means the specification violated Principle I — it was incomplete. The correct action is to return to the specification, resolve the conflict there, and re-lock. Escalating at runtime treats a specification failure as an operational event. It is not.

2. Runtime human escalation reintroduces the ambiguity SDPF was designed to eliminate.

A specification is a contract. A contract that defers resolution of key terms to an undefined future negotiation is not a contract — it is a draft. Allowing human escalation at execution time means the AI is executing a contract with unresolved terms. The output will reflect that incompleteness in ways that are unpredictable and non-reproducible.

3. The Bounded Stochasticity Theorem requires structural completeness.

Section 6 proves that stochastic variation in AI output is bounded by the structural invariants defined in the specification. This theorem only holds when the specification is complete — when all invariants are defined. A specification that defers conflict resolution to human escalation has undefined invariants in the area of the conflict. The theorem does not hold and AI output becomes structurally unbounded in that area.

4. Automated resolution is deterministic, auditable, and reproducible. Human escalation is none of these.

Strictness priority resolution — [CRITICAL] > [REQUIRED] > [OPTIONAL] — produces the same outcome every time given the same inputs, and records the resolution in the evidence package event log with the requirement IDs and outcome. A human escalation produces a decision that depends on who is available, their knowledge, their judgment, and their current state. It cannot be reproduced, audited, or verified. It produces a gap in the evidence chain.

5. The only unresolvable conflict requires practitioner action at specification time, not execution time.

When two [CRITICAL] requirements genuinely conflict, no automated rule can safely resolve them — dropping either may remove a safety-critical requirement. This is the one case where human judgment is required. SDPF enforces it at the correct point: specification locking is blocked until the practitioner resolves the conflict. The human intervenes before execution, when the full specification context is available and the decision can be reasoned, recorded, and reviewed. This is fundamentally different from ad-hoc intervention at execution time.

SUMMARY: Human escalation was not removed to eliminate human judgment from SDPF. It was removed from execution time and enforced at specification time — where it belongs. The one case that genuinely requires human judgment (CRITICAL-CRITICAL conflict) is handled by blocking the specification lock until the practitioner resolves it. All other conflicts are resolved automatically, deterministically, and auditably by strictness priority.

10.2 Strictness Priority Precedence

PRECEDENCE: [CRITICAL] > [REQUIRED] > [OPTIONAL]

Conflict	Resolution	Required Action
[CRITICAL] vs [REQUIRED]	[CRITICAL] takes precedence automatically.	Tool surfaces the superseded [REQUIRED] requirement as a warning. No blocking action.
[CRITICAL] vs [OPTIONAL]	[CRITICAL] takes precedence automatically.	Tool surfaces the superseded [OPTIONAL] requirement as informational. No blocking action.
[REQUIRED] vs [OPTIONAL]	[REQUIRED] takes precedence automatically.	Tool surfaces the superseded [OPTIONAL] requirement as informational. No blocking action.
[CRITICAL] vs [CRITICAL]	Specification error — cannot be resolved by precedence.	Tool blocks locking. Practitioner must resolve explicitly before locking.

10.3 Normative Rules

CONF-1 A conforming tool shall detect conflicts between requirements at locking time, before any implementation begins.

CONF-2 When a [CRITICAL] requirement conflicts with a [REQUIRED] or [OPTIONAL] requirement, the tool shall automatically apply strictness priority precedence and record the resolution in the event log.

CONF-3 When two [CRITICAL] requirements conflict, the tool shall block locking and surface the conflict as a specification error. The practitioner shall resolve the conflict by modifying, merging, or removing one of the conflicting requirements.

CONF-4 A specification with an unresolved CRITICAL-CRITICAL conflict shall not be locked. No implementation shall begin until the conflict is resolved.

CONF-5 The automated resolution of a conflict by strictness priority precedence shall be recorded in the evidence package event log with the requirement IDs involved and the resolution outcome.

CONF-6 Human escalation at implementation or execution time is not a conforming conflict resolution mechanism. All conflicts shall be resolved at specification time.

10.4 Conflict Resolution in Style 14 — Dynamic Criticality Extension

Style 14 governs systems where criticality levels change at runtime in response to system state. All human escalation is replaced by automated structural escalation governed by the Conflict Resolution Protocol.

DYNE-1 The escalated criticality level shall be applied automatically according to the state-escalation map defined in the specification.

DYNE-2 If the escalated state creates a conflict between requirements, the Conflict Resolution Protocol shall be applied automatically using strictness priority precedence.

DYNE-3 A state transition that would create a CRITICAL-CRITICAL conflict shall be treated as a system error condition and shall trigger the error response defined for that condition in the Exception Handling section.

DYNE-4 No Style 14 specification shall defer conflict resolution to a human operator at runtime. All possible state transitions and their conflict implications shall be defined in the specification before implementation begins.

11 The SDPF Style System

11.1 Purpose and Scope

A SDPF style is a formally defined dialect governing semantic emphasis, additional required sections, verification focus, AI framing, and evidence expectations for a specific class of system. Style selection is the first decision in Phase 1 — after the problem statement is validated — and before any requirement is written.

STYLE-1 Every valid SDPF specification shall declare exactly one style before locking.

STYLE-2 The declared style shall be one of the 17 normative styles or a conforming custom style. Style selection shall be informed by the problem boundary established in Phase 0.

STYLE-3 Changing the style after locking is a BREAKING change.

11.2 Style Selection Protocol

Evaluate conditions in order — stop at the first match.

Condition	Style
Hardware constraints or real-time performance requirements are the primary design driver	8 — Constraint-Based
System is regulated, compliance-driven, or security-critical	10 — Compliance-Driven
Requirements are unknown and must emerge through exploration	2 — Exploratory Problem-Solving
Requirements will change frequently and predictably	5 — Iterative Refinement
Primary deliverable is a user interface or interactive experience	3 — User Experience-Centered
System is a library or component others will depend on	7 — Test-Driven Specification
System is distributed across multiple services or teams	6 — System Architecture
Primary goal is to improve or preserve existing working code	9 — Maintenance / Refactor
Creativity or generative behaviour is the primary goal	4 — Creative Scaffolding
None of the above — deterministic system with fully definable I/O	1 — Technical Specification

11.3 The 17 Normative Styles

Each style is defined by: canonical name and ID, primary use case, key differentiator, additional required sections, normative semantic constraints, verification focus, and AI framing.

Style 1 — Technical Specification

Property	Definition
ID	1
Use Case	APIs, data pipelines, deterministic systems
Differentiator	Contract-first. All behaviour is deterministic and fully specifiable.
Additional Sections	None beyond S-01 to S-10
Semantic Constraints	Processing rules shall contain explicit conditional logic. Output guarantees shall use contract-assertion language.
Verification Focus	I/O contract completeness. Every input condition has a corresponding output guarantee and exception entry.
AI Framing	Expert software engineer implementing a contract-first specification. No behaviour beyond the contract.

Style 2 — Exploratory Problem-Solving

Property	Definition
ID	2
Use Case	Novel algorithms, research, optimisation
Differentiator	Specification defines the search space and abort criteria, not the solution.
Additional Sections	S-11: Hypothesis List. S-12: Abort Criteria. S-13: Fallback Strategy.
Semantic Constraints	At least one testable hypothesis. Explicit abort criteria with time or resource bounds.
Verification Focus	Hypothesis testability. Abort criteria completeness. Fallback reachability.
AI Framing	Expert algorithm researcher. Implement hypotheses and abort criteria. Do not assert a solution exists before validation.

Style 3 — User Experience-Centered

Property	Definition
ID	3
Use Case	Web and mobile UI, interactive applications, design systems
Differentiator	User journeys and accessibility defined before implementation.
Additional Sections	S-11: User Journey Map. S-12: Accessibility Requirements. S-13: Usability Metrics.
Semantic Constraints	User journey definitions present. Accessibility conformance target declared.
Verification Focus	User journey coverage. Accessibility conformance target stated. Usability metrics defined.
AI Framing	Expert frontend engineer. Implement all user flows as specified. Meet the declared accessibility conformance target.

Style 4 — Creative Scaffolding

Property	Definition
ID	4
Use Case	Games, generative applications, content creation tools
Differentiator	Innovation space explicitly bounded before operational constraints are locked.
Additional Sections	S-11: Innovation Bounds. S-12: Operational Constraints. S-13: Output Validation Criteria.
Semantic Constraints	Clear boundaries between creative and constrained areas defined.
Verification Focus	Innovation bounds present. Operational constraints measurable. Output validation criteria defined.
AI Framing	Expert creative software engineer. Stay within defined innovation bounds. Respect all operational constraints.

Style 5 — Iterative Refinement

Property	Definition
ID	5
Use Case	Agile development, prototypes, evolving requirements
Differentiator	Change tolerance and revision paths are contract-level controls.
Additional Sections	S-11: Change Classification Rules. S-12: Revision Procedures. S-13: Version History.
Semantic Constraints	Change classification rules defined. Each version numbered and change type recorded.
Verification Focus	Version header present. Change type recorded. Revision procedure defined.
AI Framing	Implement the current version exactly. Do not implement anticipated future changes.

Style 6 — System Architecture

Property	Definition
ID	6
Use Case	Distributed systems, microservices, scalable multi-component services
Differentiator	System topology and inter-service interface contracts are explicit.
Additional Sections	S-11: System Topology. S-12: Interface Contracts. S-13: Fault Boundary Definitions.
Semantic Constraints	System topology defined. Every inter-service interface has an explicit contract.
Verification Focus	Topology defined. Interface contracts present. Fault boundaries specified.
AI Framing	Expert systems architect. Implement each service boundary exactly as specified. Respect all interface contracts.

Style 7 — Test-Driven Specification

Property	Definition
ID	7
Use Case	Libraries, shared APIs, critical infrastructure
Differentiator	Tests are locked before any implementation is generated.
Additional Sections	S-11: Pre-Implementation Test Suite — complete and locked before implementation.
Semantic Constraints	Test suite locked before implementation gate opens.
Verification Focus	Tests precede implementation. Test suite covers all CRITICAL requirements.
AI Framing	Produce the complete test suite first. Implementation second. Do not generate implementation before the test suite is complete.

Style 8 — Constraint-Based

Property	Definition
ID	8
Use Case	Embedded systems, real-time systems, performance-critical code
Differentiator	Hard constraints are primary requirements. Constraint satisfaction is the primary verification criterion.
Additional Sections	S-11: Hard Constraint Register. S-12: Constraint Priority Order.
Semantic Constraints	Every hard constraint has a measurable threshold and verification method. Constraint violation is a CRITICAL failure.
Verification Focus	All constraints have measurable thresholds. Constraint verification methods are executable.
AI Framing	Expert performance engineer. Every hard constraint is a blocking requirement. Do not produce an implementation that violates any constraint.

Style 9 — Maintenance / Refactor

Property	Definition
ID	9
Use Case	Legacy code modernisation, technical debt reduction
Differentiator	Preserve-behaviour commitments are mandatory.
Additional Sections	S-11: Preserve-Behaviours List. S-12: Permitted-Changes Scope. S-13: Regression Test Requirements.
Semantic Constraints	Non-empty preserve-behaviours list. Every preserved behaviour has a regression test.
Verification Focus	Preserve-behaviours list non-empty. Regression tests cover all preserved behaviours.
AI Framing	Do not modify any behaviour in the preserve-behaviours list. Confine all changes to the permitted-changes scope.

Style 10 — Compliance-Driven

Property	Definition
ID	10
Use Case	Healthcare (HIPAA), finance (SOX, PCI-DSS), government, security-critical
Differentiator	Regulatory mapping and audit evidence are built-in outputs. Every CRITICAL requirement maps to a regulatory clause.
Additional Sections	S-11: Regulatory Mapping. S-12: Audit Evidence Requirements. S-13: Compliance Verification Procedure.
Semantic Constraints	Every CRITICAL requirement contains a regulatory mapping. Applicable regulatory framework declared.
Verification Focus	Regulatory mapping complete. Audit evidence requirements specified. Compliance verification procedure present.
AI Framing	Expert compliance engineer. Annotate every function with REQ-ID and regulatory mapping. Include audit trail generation as specified.

Style 11 — Meta-Hypercomputer Specification

Property	Definition
ID	11
Use Case	Universal recursive computation engines, operator family systems
Differentiator	Operator families, closure properties, and absorption constraints are explicit.
Additional Sections	S-11: Operator Family Definitions. S-12: Recursion Bounds.
Semantic Constraints	Operator families defined. Closure and absorption properties stated. Recursion bounds explicit.
Verification Focus	Operator families defined. Closure properties stated. Recursion termination conditions verifiable.
AI Framing	Expert in recursive computation theory. All operator contracts implemented exactly.

Style 12 — Sovereign Structural Ledger

Property	Definition
ID	12
Use Case	Immutable audit ledgers, chain-of-custody systems
Differentiator	Tamper evidence and provenance are CRITICAL. Mutation is a specification violation.
Additional Sections	S-11: Immutability Contract. S-12: Provenance Requirements. S-13: Tamper Evidence Mechanism.
Semantic Constraints	Explicit immutability contract. Every record references previous record's hash. No mutation operations in processing rules.
Verification Focus	Immutability contract present. Provenance requirements defined. No mutation operations found.
AI Framing	Expert in immutable ledger systems. Any mutation is a specification violation. Implement append-only semantics exclusively.

Style 13 — Hybrid System Composition

Property	Definition
ID	13
Use Case	Multi-module, multi-team architectures
Differentiator	Each independently specified component receives its own SDPF specification. Boundary contracts are explicit.
Additional Sections	S-11: Component Inventory. S-12: Boundary Contracts. S-13: Dependency Graph.
Semantic Constraints	Each component has its own specification. Boundary contracts defined for all inter-component interfaces.
Verification Focus	All component specifications present. Boundary contracts non-empty. Dependency graph present.
AI Framing	Implement one component at a time. Respect all boundary contracts. Do not introduce undeclared cross-component dependencies.

Style 14 — Dynamic Criticality Extension

Property	Definition
ID	14
Use Case	Safety-critical systems with runtime state-dependent escalation
Differentiator	Escalation behaviour is structurally bound to runtime state changes. All conflict resolution is automated per Section 10.
Additional Sections	S-11: Criticality Level Definitions. S-12: State-Escalation Map. S-13: De-escalation Procedures.
Semantic Constraints	All criticality levels defined. Every state transition maps to a criticality level. De-escalation conditions explicit. No human escalation paths.
Verification Focus	Criticality levels defined. State-escalation map complete. De-escalation procedures present. No human escalation references.
AI Framing	Bind escalation rules to runtime state exactly as specified. Apply Conflict Resolution Protocol automatically for all state-triggered conflicts.

Style 15 — Cross-System Verification

Property	Definition
ID	15
Use Case	Formal verification systems, proof-generating pipelines
Differentiator	Invariants and closure checks required at all boundaries. Proof generation is a first-class output.
Additional Sections	S-11: Invariant Definitions. S-12: Boundary Check Requirements. S-13: Proof Artifact Requirements.
Semantic Constraints	All invariants have verification methods. Boundary checks defined at every interface. Proof artifact format specified.
Verification Focus	Invariants defined and verifiable. Boundary checks specified. Proof artifact requirements present.
AI Framing	Expert in formal verification. All invariant checks implemented. Proof artifacts generated as specified.

Style 16 — Prompt-Driven Documentation

Property	Definition
ID	16
Use Case	Auto-generated documentation, self-documenting codebases
Differentiator	Documentation is a CRITICAL deliverable and primary verification output.
Additional Sections	S-11: Documentation Structure. S-12: Generation Triggers. S-13: Documentation Verification.
Semantic Constraints	Documentation artifacts listed as CRITICAL requirements. Generation is part of implementation.
Verification Focus	Documentation structure defined. Generation triggers specified. Documentation verification method present.
AI Framing	Documentation artifacts are primary deliverables of equal status to code. Produce alongside implementation.

Style 17 — Domain-Specific Template

Property	Definition
ID	17
Use Case	Clinical, legal, financial, infrastructure domains
Differentiator	Jurisdiction mapping and domain compliance embedded in the contract.
Additional Sections	S-11: Domain Vocabulary. S-12: Jurisdiction Mapping. S-13: Domain Compliance Matrix.
Semantic Constraints	Domain vocabulary defined. Jurisdiction mapping explicit. Domain compliance matrix non-empty.
Verification Focus	Domain vocabulary present. Jurisdiction mapping non-empty. Compliance matrix maps to at least one CRITICAL requirement per jurisdiction.
AI Framing	Expert in declared domain. All domain compliance requirements implemented. Domain vocabulary used consistently.

11.4 Custom Styles

CSTYLE-1 Unique name and locally-assigned ID not conflicting with normative IDs 1–17.

CSTYLE-2 Use case declaration.

CSTYLE-3 Differentiator statement.

CSTYLE-4 Semantic rule set — one or more keyword rules the specification content shall satisfy.

CSTYLE-5 Custom style definitions shall be persisted and version-controlled alongside specifications that use them.

12 Lifecycle Protocol

12.1 Phase 0 Gate

GATE-0 A conforming tool shall confirm Phase 0 completion before permitting any Phase 1 action. Phase 0 completion requires: a validated problem statement passing all four tests, a confirmed problem owner, and practitioner confirmation that Phase 0 is complete.

12.2 Phase 1 Stage Definitions

Stage	Entry Condition	Permitted Actions
TSP_DRAFT	Phase 0 complete. Problem statement validated.	Write specification from problem statement. Select style.
SPEC_LOCKED	All sections present. TVG entries verified. Style declared. No CRITICAL-CRITICAL conflicts.	Generate tests only.
TESTS_GENERATED	At least one test generated from locked specification.	Review, regenerate, lock tests.
TESTS_LOCKED	Tests exist and frozen.	Generate implementation only.
CODE_GENERATED	Implementation exists.	Run verification gate only.
VERIFIED	All 11 structural invariant checks pass.	Export evidence, archive run.

12.3 Normative Gate Rules

GATE-1 A conforming tool shall raise a blocking error on any out-of-sequence action.

GATE-2 Stage shall be persisted after every successful transition and restored on restart.

GATE-3 A specification shall not be locked if any TVG entry has not been verified, if any required section is missing, or if any CRITICAL-CRITICAL conflict remains unresolved.

GATE-4 Locking is irreversible within a single run.

GATE-5 Tests shall not be modified after locking.

GATE-6 If verification fails, the stage remains at CODE_GENERATED until all checks pass.

12.4 Change Protocol

Type	Definition	Required Action
BREAKING	Modifies, removes, or replaces a CRITICAL requirement or output contract.	Full re-implementation and re-verification from SPEC_LOCKED.
SIGNIFICANT	Modifies a REQUIRED requirement or adds a new requirement.	Affected components re-verified. Verification gate re-run.

Type	Definition	Required Action
MINOR	Clarifies language, updates TVG entries, or amends informative text.	Review only. Change recorded in version history.

CHANGE-1 Every post-lock modification shall be classified before application.

CHANGE-2 Every change shall be recorded in the version history with: version number, change type, description, and date.

CHANGE-3 BREAKING increments major version. SIGNIFICANT increments minor version. MINOR increments patch version.

12.5 Prompt Export Protocol

PROMPT-1 An exported prompt shall include the complete specification content from all required sections.

PROMPT-2 An exported prompt shall include a style-specific preamble framing the AI executor as the appropriate expert for the declared style.

PROMPT-3 An exported prompt shall require the AI to annotate every function with its REQ-ID.

PROMPT-4 An exported prompt shall prohibit the AI from adding behaviour not defined in the specification.

PROMPT-5 The exported prompt shall be compatible with any capable AI model without model-specific modification.

13 Verification Model

The verification gate tests the eleven structural invariants of a SDPF-conforming run. Each check tests whether a required structural property is satisfied — not whether the surface form of the output matches an expected pattern.

13.1 The Eleven Structural Invariant Checks

Check	Structural Invariant Tested	Failure Consequence
spec_exists	Intent was formally expressed from a validated problem statement before execution began.	Stage cannot advance. Specification must exist.
style_defined	Intent was classified into a known domain before execution.	Run cannot be verified. Style must be declared and resolved.
tests_exist	Intent produced tests — specification was not bypassed.	Run cannot be verified. Tests must be generated.
tests_locked	Tests were frozen before implementation — intent governed the test suite.	Run cannot be verified. Tests must be locked before implementation.
implementation_exists	Implementation was produced after intent was locked.	Run cannot be verified. Implementation must be generated.
traceability_complete	Every requirement has at least one test — no intent line was skipped.	Run cannot be verified. All requirements must have test coverage.
style_constraints_present	The style's structural marker appears in the specification — style was applied.	Run cannot be verified. Style was declared but not applied.
style_N_engine_verified	Style-specific engine tags appear in tests and code — style governed the artifacts.	Run cannot be verified. Style engine not applied to generated artifacts.
policy_engine_passed	Specification content satisfies semantic rules for the declared style.	Run cannot be verified. Specification content does not satisfy style requirements.
ci_gate_ready	A CI workflow was generated — invariants can be enforced in automated pipelines.	Run cannot be verified. CI gate configuration is required.
provenance_signing_ready	A signing key is configured — evidence can be tamper-proofed.	Run cannot be verified in production mode. Signing key required.

13.2 Normative Verification Rules

VER-1 All eleven checks shall be evaluated on every verification gate execution. No check may be skipped.

VER-2 A run shall not advance to VERIFIED unless all eleven checks return pass.

VER-3 The result of every check shall be recorded in the Verification Closure Record.

VER-4 A check result shall be boolean. Partial pass is not valid.

VER-5 A failed gate execution leaves the stage at CODE_GENERATED.

VER-6 The verification execution timestamp shall be recorded in the Verification Closure Record.

13.3 Five-Gate Verification Hierarchy

Gate	Name	What It Checks	Blocks
G1	Technical Verification Gate	All TVG entries verified against live output. No unverified technical facts in specification. See Section 2.	Build start
G2	Unit Verification	Each component satisfies its individual specification requirement.	Integration
G3	Integration Verification	All components work together as specified at every interface boundary.	System test
G4	Clean-Machine Verification	All outputs execute correctly in an environment with no development dependencies.	Release
G5	Closure Record	All verification results recorded and signed. CLOSURE STATUS = COMPLETE.	Shipment

13.4 Verification Closure Record

VCR-1 Shall contain the specification version.

VCR-2 Shall contain the verification execution timestamp in ISO 8601 UTC.

VCR-3 Shall contain the identity of the verifying entity.

VCR-4 Shall contain a result row for every check with check ID, requirement text, result, and notes.

VCR-5 Shall contain CLOSURE STATUS: COMPLETE, INCOMPLETE, or BLOCKED.

VCR-6 If INCOMPLETE or BLOCKED, shall list unresolved items and reason.

VCR-7 Release shall not be permitted until CLOSURE STATUS = COMPLETE.

14 Evidence Standard

14.1 Evidence Package Schema

Field	Presence	Definition
problem_statement	REQUIRED	The validated problem statement from Phase 0, including current state, desired state, gap, impact, and problem owner.
stage	REQUIRED	Shall be VERIFIED for a complete run.
specification	REQUIRED	Complete locked specification including all sections, style metadata, and REQ-IDs.
tests	REQUIRED	All test cases with test ID, REQ-ID trace, and test body.
implementation_sha256	REQUIRED	SHA-256 hash of the implementation code.
verification	REQUIRED	Verification Closure Record as defined in Section 13.4.
traceability	REQUIRED	One row per requirement mapping REQ-ID to test IDs and implementation hash.
event_log	REQUIRED	Chronological lifecycle events including Phase 0 completion confirmation and all automated conflict resolutions.
service_map	REQUIRED	Reference architecture service definitions.
prompt_styles	REQUIRED	Full style catalog at time of export including custom styles.
policy_engine	REQUIRED	Policy engine evaluation result — rules, results, overall pass/fail.
ci_gate	REQUIRED	CI platform identifier and generated workflow configuration.
ai_results_capture	CONDITIONAL	Present when AI outputs were captured. Per-model REQ-ID coverage maps.
model_comparison	CONDITIONAL	Present when cross-model comparison was built.
conflict_resolutions	CONDITIONAL	Present when automated conflict resolution was applied. Records all resolved conflicts with requirement IDs and resolution outcomes.
provenance	REQUIRED	Provenance signature as defined in Section 14.2.

14.2 Provenance Signing Standard

PROV-1 Algorithm: HMAC-SHA256 (RFC 2104, FIPS 198-1).

PROV-2 Signing key sourced from secure environment variable or secret management. Key shall not be embedded in the evidence package.

PROV-3 Canonical payload: complete evidence package excluding the provenance field, serialised as JSON with sorted keys and no whitespace, encoded as UTF-8.

PROV-4 Provenance field shall contain: algorithm identifier, key identifier (not the key), signing timestamp in ISO 8601 UTC, HMAC-SHA256 signature as lowercase hexadecimal, and verification

procedure as human-readable string.

PROV-5 A conforming tool shall refuse to export without a valid signing key unless a designated test-mode bypass is active.

PROV-6 Test-mode bypass shall never be used in production. Test-signed packages shall be distinguishable by key_id field.

PROV-7 Signing keys shall be rotated periodically and on suspicion of compromise. Key versions shall be tracked.

14.3 Evidence Archive

ARCHIVE-1 Every evidence export shall be automatically archived with a unique timestamp-based filename.

ARCHIVE-2 Archive shall be maintained for the duration defined by organisational governance policy.

ARCHIVE-3 Archived packages shall be retained in original signed form. Re-signing creates a new package.

15 Conformance

15.1 Conformance Levels

Level	Name	Key Requirements
Level 1	Specification Conformance	Grammar satisfied. All required sections present including TVG (S-08). Problem statement referenced. All requirements tagged. TVG completed before locking. No CRITICAL-CRITICAL conflicts. Traceability matrix non-empty.
Level 2	Tool Conformance	Enforces Phase 0 gate. Enforces three core principles as blocking gates including TVG. Enforces stage sequence. Assigns REQ-IDs. Detects and resolves conflicts per Section 10. Produces conforming evidence packages including problem_statement field. Signs packages with HMAC-SHA256. Supports all 17 normative styles. Supports custom styles.
Level 3	Evidence Conformance	All required fields present including problem_statement. Stage = VERIFIED. Valid provenance signature. CLOSURE STATUS = COMPLETE. Traceability complete. Conflict resolutions recorded if applicable.
Level 4	Process Conformance	Completes Phase 0 before every project. Uses Level 2 tool. Produces Level 1 specifications. Generates Level 3 evidence for every release. Applies change protocol. Maintains evidence archive. No human escalation for conflict resolution.

15.2 Specification Conformance (Level 1)

SPEC-C-1 All ten required sections (S-01 through S-10) present in declared order. Problem statement referenced.

SPEC-C-2 Exactly one style declared in S-01. CRITICAL requirements address the problem gap.

SPEC-C-3 Every requirement begins with a priority tag.

SPEC-C-4 Exception Handling covers every reachable error condition.

SPEC-C-5 TVG covers every asserted technical fact. All TVG entries verified before locking.

SPEC-C-6 Traceability Matrix contains at least one mapping row per CRITICAL requirement.

SPEC-C-7 No unresolved CRITICAL-CRITICAL conflicts.

SPEC-C-8 Style-specific additional sections present where required by the declared style.

15.3 Tool Conformance (Level 2)

TOOL-C-1 Phase 0 gate enforced. Specification locking blocked until Phase 0 completion confirmed.

TOOL-C-2 Enforces all three core principles as blocking gates.

TOOL-C-3 Enforces stage gate sequence. Gate violations produce blocking errors.

TOOL-C-4 Detects requirement conflicts at locking time. Applies strictness priority precedence automatically. Blocks locking on CRITICAL-CRITICAL conflicts.

TOOL-C-5 Assigns REQ-IDs on locking in R-NNN format.

TOOL-C-6 Evaluates all eleven structural invariant checks on every verification gate execution.

TOOL-C-7 Produces evidence packages conforming to Section 14.1 schema including problem_statement field.

TOOL-C-8 Signs evidence packages using HMAC-SHA256 per Section 14.2.

TOOL-C-9 Archives every evidence export per Section 14.3.

TOOL-C-10 Persists lifecycle state and restores on restart.

TOOL-C-11 Supports all 17 normative styles and custom styles.

TOOL-C-12 Exports prompts conforming to the prompt export protocol in Section 12.5.

TOOL-C-13 Records all automated conflict resolutions in the evidence package.

15.4 Evidence Package Conformance (Level 3)

EVID-C-1 problem_statement field present with all four components: current state, desired state, gap, and impact.

EVID-C-2 All required fields per Section 14.1 present.

EVID-C-3 stage field = VERIFIED.

EVID-C-4 Valid HMAC-SHA256 provenance signature.

EVID-C-5 Verification Closure Record with CLOSURE STATUS = COMPLETE.

EVID-C-6 Traceability row for every requirement.

EVID-C-7 Complete locked specification with all REQ-IDs.

15.5 Process Conformance (Level 4)

PROC-C-1 Phase 0 completed before every project enters Phase 1.

PROC-C-2 Every system governed by a Level 1 conforming specification before implementation begins.

PROC-C-3 Every release accompanied by a Level 3 conforming evidence package.

PROC-C-4 All post-lock specification modifications governed by the change protocol.

PROC-C-5 Level 2 conforming tool used for all SDPF lifecycle operations.

PROC-C-6 Evidence archive maintained per organisational governance policy.

PROC-C-7 Regulated systems use Style 10 or a custom style with equivalent regulatory mapping requirements.

Annex A — Informative: Normative Requirement Index

(Informative) All normative requirement identifiers defined in this specification.

Range	Section	Topic
TVG-1 to TVG-5	2.5 and 9.3.6	Technical Verification Gate rules
P0-1 to P0-8	3.5	Phase 0 normative requirements
PRIN-1 to PRIN-4	5.2	Three Core Principles enforcement
REQ-SYNTAX-1 to REQ-SYNTAX-6	9.2	Requirement syntax rules
EC-1 to EC-3	9.3.1	External Contract content
IC-1 to IC-4	9.3.2	Input Contract content
PR-1 to PR-4	9.3.3	Processing Rules content
OG-1 to OG-3	9.3.4	Output Guarantees content
EH-1 to EH-4	9.3.5	Exception Handling content
TM-1 to TM-3	9.3.7	Traceability Matrix rules
CONF-1 to CONF-6	10.3	Conflict Resolution Protocol
DYNE-1 to DYNE-4	10.4	Style 14 automated escalation
STYLE-1 to STYLE-3	11.1	Style system rules
CSTYLE-1 to CSTYLE-5	11.4	Custom style rules
GATE-0	12.1	Phase 0 gate
GATE-1 to GATE-6	12.3	Lifecycle gate rules
CHANGE-1 to CHANGE-3	12.4	Change protocol rules
PROMPT-1 to PROMPT-5	12.5	Prompt export protocol
VER-1 to VER-6	13.2	Verification gate rules
VCR-1 to VCR-7	13.4	Verification Closure Record rules
PROV-1 to PROV-7	14.2	Provenance signing rules
ARCHIVE-1 to ARCHIVE-3	14.3	Evidence archive rules
SPEC-C-1 to SPEC-C-8	15.2	Specification conformance
TOOL-C-1 to TOOL-C-13	15.3	Tool conformance
EVID-C-1 to EVID-C-7	15.4	Evidence package conformance
PROC-C-1 to PROC-C-7	15.5	Process conformance

Annex B — Informative: The Complete SDPF Checklist

(Informative) Use this checklist for every SDPF-governed project.

Phase 0 — Problem Identification and Definition

#	Checklist Item	Validated By
P0-1	Current state observed and measured.	Observable evidence of gap.
P0-2	Desired state defined specifically and measurably.	Can be tested when achieved.
P0-3	Gap quantified as a number.	Numeric measure confirmed.
P0-4	Impact stated in business terms.	Cost, risk, time, or compliance consequence.
P0-5	Problem statement written.	Follows: "[current state] against a target of [desired state], resulting in [impact]."
P0-6	T-1 Observable test passed.	Problem can be seen or measured.
P0-7	T-2 Bounded test passed.	Problem has clear start and end.
P0-8	T-3 Cause-Free test passed.	No cause language in statement.
P0-9	T-4 Solution-Free test passed.	No solution language in statement.
P0-10	Problem owner confirmed.	Named person accountable for closing the gap.

Phase 1 — SDPF Specification and Delivery

#	Checklist Item
S-1	Style selected. Problem boundary informed the selection.
S-2	All ten required sections complete. Problem statement referenced.
S-3	CRITICAL requirements directly address the problem gap.
S-4	TVG (S-08) completed. All technical facts verified against live output.
S-5	No CRITICAL-CRITICAL conflicts. All other conflicts resolved by strictness priority.
S-6	Specification locked.
S-7	Tests generated. Every requirement has a test.
S-8	Tests locked.
S-9	Implementation generated.
S-10	Verification gate passed. All eleven structural invariant checks = PASS.
S-11	Evidence exported. problem_statement field present. Provenance signature valid.
S-12	Evidence archived. CLOSURE STATUS = COMPLETE.

Annex C — Informative: Cumulative Change Log

(Informative) All changes across all versions of the SDPF Language Specification.

Changes in Version 1.1 (from Version 1.0)

Change	Classification	Sections Affected
TVG elevated to Core Principle II.	BREAKING	Section 5, Section 9.3.6, Section 12.3
CodeJuicer example added.	MINOR — informative	Section 2.4
Bounded Stochasticity Theorem added.	SIGNIFICANT	Section 6, Section 13
Automated Conflict Resolution added. Human escalation removed.	SIGNIFICANT	Section 10, Style 14, Section 15.3, Section 15.5
conflict_resolutions field added to evidence schema.	SIGNIFICANT	Section 14.1

Changes in Version 1.2 (from Version 1.1)

Change	Classification	Sections Affected
Phase 0 added as mandatory lifecycle gate.	BREAKING	Sections 3, 4, 5 (PRIN-1), 12 (GATE-0)
LSS DMAIC added to normative references.	MINOR	Section 7
Problem statement terms added to vocabulary.	SIGNIFICANT	Section 8
REQ-SYNTAX-5 added.	SIGNIFICANT	Section 9.2
STYLE-2 updated.	MINOR	Section 11
problem_statement field added as REQUIRED in evidence schema.	SIGNIFICANT	Section 14.1
TOOL-C-1 and EVID-C-1 added.	SIGNIFICANT	Section 15

Changes in Version 1.3 (from Version 1.2)

Change	Classification	Sections Affected
TVG elevated to top-level Section 2.	SIGNIFICANT	Section 2 (new top-level)
All sections renumbered.	MINOR	All sections from 3 onward
Unified edition — all normative content consolidated.	MINOR	All sections
Annex C added.	MINOR — informative	Annex C

Changes in Version 1.3.1 (from Version 1.3)

Change	Classification	Sections Affected
Section 1.5 Key Terms Quick Reference added. Defines TVG, Phase 0, TSP, REQ-ID, Bounded Stochasticity, LSS DMAIC, Automated Conflict Resolution, Evidence Package, Provenance Signature, Style, and CRITICAL/REQUIRED/OPTIONAL tags in plain language with section cross-references.	MINOR — informative addition, no normative content changed	Section 1.5 (new)
Section 10.1 Why Human Escalation Was Removed added. Provides five explicit justifications for the architectural decision made in v1.1 to replace human escalation with automated conflict resolution at specification time.	MINOR — informative addition, no normative content changed	Section 10.1 (new); existing 10.1/10.2/10.3 renumbered to 10.2/10.3/10.4
Foreword restructured. 'How to Read This Document' guidance added. Closure status added to Phase 0 problem statement callout.	MINOR — readability improvement, no normative content changed	Foreword

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

SDPF Language Specification — Version 1.3.1

Hamza Abdullah · 2026