PRISM: A Source-First Enterprise Architecture Standard and Framework Family for the Modern Organization

Vikram Goud Palakurthi

Independent Researcher

https://orcid.org/0009-0002-3370-0671

https://prism-framework.org

Author Note

Vikram Goud Palakurthi https://orcid.org/0009-0002-3370-0671. The author reports no conflicts of interest and received no funding for this work. Correspondence: vikram.palakurthi@gmail.com

Abstract

The current enterprise architecture landscape, while solid in theory has a gap problem, not a theory problem. The major frameworks such as Zachman, TOGAF, ArchiMate — are rigorous, they are well-maintained, and widely documented. They were also designed for organizations with dedicated EA practitioners, have separate governance processes, and architecture as a separate discipline. However, most engineering teams do not work that way. Much of the architecture work ends up in Visio files, draw.io and shared drives, which gets stale quickly within months, and ends up describing a system that no longer exists.

PRISM takes a different starting point: architecture artifacts as plain-text YAML files, which are governed by git, validated by JSON Schemas, and authored by anyone on the team. The framework organizes the enterprise across five interconnected layers which includes, People and Place, Reality, Intent, Signal, and Movement, and additionally extends into six focused disciplines covering infrastructure, cybersecurity, AI architecture, application architecture, data, and product validation. By design, all seven frameworks share the same authoring conventions throughout.

This paper introduces the PRISM framework family: its design rationale, the structure of each framework, how they connect, and where the current boundaries are. PRISM is presented as an open standard at version 1.0, and is designed to be refined through practitioner feedback and adoption over time.

Keywords: enterprise architecture, PRISM, source-first architecture, git-native governance, schema validation, enterprise architecture landscape, infrastructure landscape, cybersecurity posture, AI architecture, application architecture, data architecture, product validation

PRISM: A Source-First Enterprise Architecture Standard and Framework Family for the Modern Organization

Introduction

Architecture has a rot problem. You request a diagram, someone draws it in Visio or a SaaS tool, and it lives behind a login or in a shared drive. Six months later, half of what it describes has changed. The diagram hasn't. When someone finally asks for 'the current state architecture,' they get a stale file or a meeting to schedule a meeting.

That's not a tool problem. It's a philosophy problem. Architecture was treated as documentation when it should have been treated as source code.

I built PRISM because I wanted enterprise architecture to work the way code works. You don't ask permission to read the codebase. You don't schedule a meeting to find out what a system does. You simply open the repository. PRISM brings that discipline to enterprise architecture where your organization's structure, systems, data flows, security posture, infrastructure, AI architecture, and product decisions are all represented as plain-text YAML files, committed to git, versioned, reviewable, and always current. There is no proprietary tooling, no centralized gatekeeper and no diagram that only one person knows how to update.

To be clear, the existing frameworks are not the problem. Zachman (1987) built the first rigorous ontology for information systems architecture. TOGAF (The Open Group, 2018) gave the field a structured development method. ArchiMate (The Open Group, 2019) gave it a visual language. These are foundational contributions and PRISM builds on all of them. But they were designed for settings with dedicated EA practitioners, has separate governance processes, and where organizational structures large enough to operate both. Most engineering teams don't work that way and that’s a gap.

The name PRISM is earned, not assigned. A prism refracts a single beam of light into its constituent parts, where what enters as white exits as a spectrum. PRISM does the same for an enterprise: one complex organization is broken into five layers, where each answers a question the others do not. That core structure also extends into six focused disciplines where each applying the same source-first approach to a specific domain. The whole lives in a git repository. Any team member can author it and Continuous Integration (CI) can validate it.

This paper presents PRISM in full, including its core principles, each framework, how they connect, and where the current limits are. Section 2 reviews the existing frameworks PRISM inherits from. Section 3 describes the four shared design principles. Sections 4 through 10 introduce each framework. Section 11 covers cross-framework integration. Section 12 discusses applications beyond IT. Section 13 names the limitations directly. Section 14 concludes.

Background and Related Work

Enterprise Architecture Frameworks

John Zachman introduced the first systematic ontology for information systems architecture back in the year 1987 also known as the Zachman Framework which can be used to organize architectural artifacts along two dimensions, six interrogatives (What, How, Where, Who, When, Why) and six stakeholder perspectives from executive to operations. Its contribution is completeness and the framework ensures no architectural question goes unnamed. PRISM's five-layer structure draws directly from these interrogatives by grouping tightly coupled concerns rather than keeping each dimension separate.

TOGAF at version 9.2 provides both a domain taxonomy i.e., Business, Application, Data, Technology and a development process, which is the Architecture Development Method.

It's the most widely documented EA framework that is in active use. PRISM's temporal model (baseline, transition, target), directly inherits TOGAF's concept of current, transition, and target architecture states. The Reality layer in PRISM maps across all four TOGAF domains without enforcing their separation, which in practice tends to produce four half-maintained documents instead of one maintained landscape.

ArchiMate provides a standardized visual modeling notation for enterprise architectures, with a formal language for describing relationships across business, application, and technology layers. PRISM takes a text-first approach with YAML files rather than modeling diagrams, to prioritize version control compatibility and schema validation. The two work well together: PRISM captures the source-of-truth landscape, ArchiMate renders it for stakeholder communication when needed, provided it supports PRISM standard in the future.

Domain-Specific Standards

NIST CSF (National Institute of Standards and Technology, 2018) and ISO/IEC 27001 (International Organization for Standardization, 2022) are the primary structured frameworks for cybersecurity governance. DAMA-DMBOK (DAMA International, 2017) covers data management comprehensively. NIST AI RMF (National Institute of Standards and Technology, 2023) and the EU AI Act (European Parliament, 2024) establish governance requirements for AI systems.

While these are authoritative and actively maintained standards. PRISM's focused disciplines i.e., PRISM/CY for cybersecurity, PRISM/D for data, PRISM/AI for AI architecture are specifically designed to work alongside them. PRISM describes the landscape and the domain standards define the controls and compliance obligations within it. The intent is that a

PRISM/CY landscape can be read alongside an ISO 27001 audit without the two being in tension.

PRISM's Starting Point

Ross, Weill, and Robertson (2006) found that having architectural clarity is associated with significantly better business outcomes. The barriers to that clarity are rarely intellectual, instead they are practical. Most teams do not maintain architecture because maintaining it requires a separate discipline, separate tooling, and a governance process that runs alongside engineering rather than inside it.

PRISM's starting question was: what would architecture look like if it lived where the code lives? The Technology Acceptance Model (Davis, 1989) establishes that perceived usefulness and ease of use eventually determine the adoption. Armenakis, Harris, and Mossholder (1993) demonstrated that organizational readiness for change is measurable. These both findings have shaped how PRISM was designed i.e., for not just what it covers, but how easy it should be to start.

Shared Design Principles

Four principles run consistently across PRISM and all its focused disciplines. These are not preferences rather they are the constraints that make the rest of the design coherent.

Source-First

In the world or PRISM, architecture artifacts are plain-text files i.e., YAML for structured data, Markdown for narrative. The file is the source of truth and not a rendering of something stored elsewhere. YAML was the right choice because it reads as structured prose as it supports comments and does not require quoting most values, and handles multi-line rationale fields with block scalars. A business analyst can author a PRISM artifact in VS Code without

knowing what a schema is. JSON Schema handles validation on the machine side to enable strict, tooling-native, which is not something a human author needs to manually verify.

Git-Native

I chose to treat git not as a backup mechanism but as the governance engine. With respect to PRISM, branches represent architectural exploration. Commits are decisions. Pull requests are governance checkpoints. Tags mark published baselines and targets. The audit trail is the git history. Engineering teams already manage code this way. PRISM applies the same model to architecture artifacts so governance happens inside the existing workflow and not in a parallel process where teams have to context-switch into.

Schema-Based Validation

Every artifact type in every PRISM framework validates against a published JSON Schema. A CI pipeline can reject a malformed artifact before it's merged without requiring a practitioner review. Schemas are versioned alongside the framework, and artifacts declare the schema version they target. That inherently creates a contract between the framework and its implementations and it evolves predictably without silently breaking existing artifacts.

Domain Agnosticism

Every PRISM framework defines its concepts without assuming an information technology context. 'System' can mean a software application, a hospital department, or a production line. 'Stakeholder' can mean an engineer, a clinician, or a factory floor supervisor. 'Signal' can mean a REST API, an HL7 message, or an EDI transaction. This generalization is meant to reduce the translation overhead that currently forces IT and non-IT teams to maintain separate architectural terminologies. And PRISM is developed to work across non-IT domains at scale.

PRISM: Enterprise Architecture

The core PRISM framework organizes the enterprise across five interconnected layers and a temporal meta-dimension. The acronym PRISM maps to its layers: People and Place (P), Reality (R), Intent (I), Signal (S), and Movement (M).

Six concerns appear consistently in EA practice: Who, Where, What, How, Why, When and PRISM framework introduces If-Then, Exchange, Pulse, and Legacy. PRISM by design consolidates them into five layers by grouping the concerns that are tightly coupled in practice, not because the concerns are identical, but because separating them produces maintenance overhead without any analytical benefit. Separating risks from goals will give us rationale without accountability, or accountability without context. The grouping decisions are practical, not hypothetical.

P — People and Place

The foundation layer asks: who is involved in this enterprise, and where do they operate? It covers three sub-dimensions. Stakeholders capture roles, organizational units, external parties, and regulators at the role level and not the individual level, because architecture decisions are rarely individual specific. Geographies cover physical and logical operating locations, regulatory zones, and cloud infrastructure, where cloud regions and availability zones carry the same governance consequences as physical data centers and appear as geographic entities with cloud provider and region identifier fields. Environments capture operating contexts i.e., (production, staging, development, disaster recovery) as distinct from location, since the same environment type often exists across multiple geographies.

This layer draws from Zachman's Who and Where interrogatives and extends them to support non-IT organizational domains.

R — Reality

The operational layer asks: what does the enterprise have, and how does it run? Reality covers capabilities i.e., (business outcomes independent of delivery mechanism), systems (operational units like applications, teams, physical machines, manual processes), data assets (core information with classification and compliance framework fields), and processes (how work flows). The TOGAF's four architecture domains map across this layer without enforcing their separation. In practice this separation keeps Business, Application, Data, and Technology as isolated artifacts which produces four half-maintained documents for this reason the Reality layer collapses that separation deliberately.

The layer also includes AI-native system types such as ai-agent, large language model (llm), and agentic-workflow as named entries. AI systems are operational units in most enterprises now and will need to appear in the landscape with the same clarity as any other system. Lifecycle states include decommissioning and decommissioned, to enable blast radius analysis when dependencies are retired or when marked retiring.

I — Intent

The strategic layer asks: why does the enterprise operate this way, and when do its decisions matter? I have designated Intent optional in PRISM as landscapes can be useful without it, and forcing teams to declare strategic rationale before they have mapped their operational reality could create friction that kills adoption early.

However, organizations that skip Intent will lose the ability to trace architectural decisions to business rationale. That traceability matters most when someone asks, six months after a system was built, why it exists. Intent covers drivers (regulatory, market, cost, risk, and strategic forces), goals (desired outcomes with measurable success metrics), timelines,

constraints, and risks. Architectural risks are modeled as condition-consequence pairs i.e., an if_condition identifies a specific event, a then_impact describes what it does to goals or operations. That structure makes risks navigable rather than just listed.

This layer draws on Zachman's Why and When interrogatives and aligns with TOGAF's Architecture Vision phase.

S — Signal

The exchange layer asks: how do the parts communicate and transfer value? Signal covers APIs (synchronous interfaces with protocol and service agreement fields), events (asynchronous exchanges with message patterns and brokers), data contracts (formal agreements about quality, freshness, ownership, and format at system boundaries), integration patterns, and service agreements (SLA, SLO, and SLI commitments between producers and consumers).

Exchange contracts are among the most important architectural decisions an enterprise makes, and consistently the most under-documented on many occasions. Undocumented exchange contracts are implicit agreements about how systems will behave toward each other. And they represent structural governance risk that compounds as organizational complexity grows over time. The Signal layer makes those agreements explicit and auditable.

M — Movement

Movement is where PRISM diverges most visibly from existing frameworks. It integrates two dimensions that architecture practice has historically left out. Which include, the human adoption signal and the chronological architectural journey. I keep them in the same layer because a migration plan that ignores adoption readiness is considered incomplete, and the adoption data without trajectory context is uninterpretable which creates risks.

Movement has two sub-dimensions. Pulse captures the present-tense human signal i.e., adoption levels on a six-level scale from none to embedded, sentiment, change readiness, and ethics markers (which are known concerns around bias, fairness, privacy, and accessibility named at the architecture level). The change readiness dimension draws on the measurement framework of Armenakis, Harris, and Mossholder (1993). Davis (1989) established that perceived usefulness and ease of use determine whether people actually adopt new tools. And PRISM treats those adoption signals as an architectural measurement worth capturing, and not as a project management concern.

Trajectory captures the chronological arc which includes baseline and target descriptions in plain language, active transition programs with status tracking, and a decisions log that references git commit identifiers to make the decision record part of the version history. You can trace an architectural choice from the landscape artifact back to the exact commit where it was made.

Temporal Meta-Dimension

Every PRISM artifact declares one of three temporal states: baseline (the architecture as it exists today), transition (an intermediate state during planned change), or target (the desired future state). This is expressed through git tags and branch naming conventions and not separate documents. Published baselines are tagged prism/baseline/YYYY-qN, targets are tagged prism/target/YYYY-qN, and transition work lives on branches named prism/transition/<program-name>. Each layer can be in a different temporal state simultaneously.

Table 1

Mapping of Nine Architectural Dimensions to the Five PRISM Layers

Architectural Dimension	**PRISM Layer**	**Sub-Dimension**
Who — people, roles, org units	P — People and Place	stakeholders[]
Where — geographies, environments	P — People and Place	geographies[], environments[]
What — capabilities, products	R — Reality	capabilities[]
How — systems, processes	R — Reality	systems[], processes[]
Why / When — rationale, timing	I — Intent	drivers[], timelines[]
If-Then — risks, goals, constraints	I — Intent	risks[], goals[], constraints[]
Exchange — APIs, events, data flow	S — Signal	apis[], events[], data_contracts[]
Pulse — adoption, ethics, readiness	M — Movement (Pulse)	adoption_signals[], sentiment[], ethics_markers[]
Legacy — trajectory, decisions	M — Movement (Trajectory)	baseline, transition_programs[], decisions[]

Note. Dimensions drawn from established EA practice. Layer groupings reflect architectural cohesion where tightly coupled concerns are grouped together.

Table 2

PRISM Positioning Relative to Zachman and TOGAF

Dimension	**Zachman (1987)**	**TOGAF 9.2**	**PRISM 1.0**
Primary mode	Ontological reference	Process model	Authorable landscape
Artifact format	Framework-defined	Framework-defined	YAML + Markdown
Version control	Out of scope	Out of scope	Core governance mechanism
Authoring target	EA practitioners	EA practitioners	Any team member
Domain scope	Information systems	Enterprise IT	Domain-agnostic
Human adoption dimension	Out of scope	Out of scope	Built-in (M layer)
Temporal model	Implied by rows	ADM phases	Baseline / Transition / Target
Governance mechanism	External to framework	ADM phase gates	Git pull requests

Note. This table describes different design emphases, not deficiencies. Each framework serves its intended context. Zachman (1987). TOGAF (The Open Group, 2018).

PRISM/I: Infrastructure Landscape Framework

The question that prompted PRISM/I was one I kept running into, and that is we have a clear organizational architecture, but nobody can tell you what infrastructure is actually running, who owns it, or what's connected to what. That gap matters because infrastructure is where

architectural decisions become operational reality. A security policy means nothing if the practitioners enforcing it cannot navigate the landscape it applies to.

PRISM/I organizes infrastructure concerns across six dimensions: Who, Where, Runtime, State, Control, and Operations.

Who covers teams and providers that are responsible for infrastructure ownership, not usage. Who uses a database is different from who is accountable for its availability, patching, and cost. Every PRISM/I artifact declares an owner as self (the primary organization's teams), internal (another internal team), or external (a vendor or third party).

Where covers accounts, regions, environments, and network zones. Compliance zones such as GDPR-scoped EU regions, SOX-scoped financial accounts must be auditable independently of team structure. That's why Who and Where are separate dimensions as ownership and location answer different questions.

Runtime covers the compute artifacts such as clusters, services, functions, virtual machines, queues, topics, streams, and workflow engines. State covers persistence such as databases, object stores, caches, data warehouses, data lakes, and search indexes. The separation reflects that execution and persistence carry different failure modes, scaling characteristics, and ownership patterns. Mixing them will only produces artifacts that answer neither question cleanly.

Control covers access and network policy, which includes VPCs, subnets, load balancers, service meshes, IAM roles, secret stores, and firewall rules. Control stands as its own dimension because access policy is cross-cutting. For example, a firewall rule applies simultaneously to Runtime and State artifacts, not to one or the other.

Operations layer covers delivery and observability such as CI/CD pipelines, container registries, infrastructure-as-code modules, metrics, logging, tracing, alerting, dashboards, runbooks, SLO definitions, and chaos engineering tooling. This layer is last because it depends on everything above it. For example, you can't define runbooks for systems you haven't inventoried.

PRISM/I landscapes cross-reference PRISM EA artifacts via a soft links field, this is to enable navigation between the organizational and infrastructure landscapes without circular schema dependencies.

PRISM/CY: Cybersecurity Landscape Framework

NIST CSF, ISO/IEC 27001, SOC 2, and CIS Controls each define what controls should exist and how to verify they're in place. These are necessary and are well-maintained. What they don't provide is a navigable posture description. For example, which identities have access to what, and what the attack surface looks like as a whole, which threats are actively documented, and how posture is changing over time as a versioned artifact you can review and commit.

PRISM/CY addresses that gap with a a text-first, git-native, schema-validated posture landscape that practitioners, GRC teams, and security leadership can navigate, version, and communicate. It works alongside control frameworks rather than replacing them.

PRISM/CY organizes the cybersecurity posture across five dimensions.

Identity covers who and what has access, and that includes IAM roles, service accounts, external identities, privileged access, and federation configurations along with fields for MFA enforcement, access level, PAM control status, and data sensitivity authorization.

Surface covers what's exposed like public endpoints, internal endpoints, legacy interfaces, data exposure points, and administrative consoles. In cloud-native environments,

identity boundaries are at least as significant as network boundaries. A misconfigured public storage bucket is a surface exposure without an identity vector; and a compromised credential is an identity risk without a public surface. Documenting them in separate dimensions makes both independently auditable, in other words, the zero-trust security model applied to landscape documentation.

Controls layer captures the defensive posture in two categories: first, the technical controls (WAF, SIEM, EDR, DLP, CSPM) and next the organizational controls (access reviews, training, vendor assessments, policy). Each control aligns to a NIST CSF function: identify, protect, detect, respond, or recover.

Threats documents known and assessed threats, which includes external actors, insider risks, supply chain risks, AI-specific risks, end-of-life technology risks, and known vulnerabilities. Each is recorded with structured likelihood, impact, and control references.

Posture synthesizes the other four: compliance framework status, maturity assessments, open security programs, and findings from audits, penetration tests, or cloud security posture management scans. Posture comes last because it draws on the other four dimensions as it's the view of where the organization currently stands, not the underlying evidence.

PRISM/AI: AI Architecture Landscape Framework

Artificial Intelligence (AI) systems are no longer edge cases in enterprise architecture. Organizations run foundation models, fine-tuned variants, retrieval-augmented generation pipelines, autonomous agents, and multi-step agentic workflows. NIST AI RMF (National Institute of Standards and Technology, 2023), ISO/IEC 42001:2023, and the EU AI Act (European Parliament, 2024) address risk management and compliance obligations for AI

systems. None of them addresses what the AI architecture landscape looks like as a navigable, versioned artifact.

PRISM/AI starts from a claim I think is underappreciated in EA practice i.e., models are architectural assets, not experiment outputs. A model version includes training data source, lifecycle stage, declared capability type, and trust controls. Together, these carry the same organizational weight as a database schema choice. PRISM/AI treats it accordingly i.e., versioned with declared dependencies, governed by trust controls, and observable in production.

PRISM/AI organizes AI architecture across five dimensions.

Models is the inventory of every AI model the organization deploys or depends on which includes foundation models, fine-tuned variants, custom-trained models, embedding models, classifiers, and explainability layers. Fields cover model provider, capability type, lifecycle stage (research through retired), training data source, and license.

Systems describes the AI-powered applications built on those models such as AI systems, RAG pipelines, copilots, classifier services, autonomous workflows, and AI agents. One field I consider non-optional on every system artifact is human_in_the_loop, a Boolean flag which indicates whether a human must review AI output before it takes effect. That's an architectural decision about accountability and not an implementation detail.

Lifecycle covers operational processes that move models from research to production and involves training runs, evaluation suites, deployment processes, monitoring jobs, and experiments.

Trust covers the controls that constrain and document AI behavior. This includes guardrails that block or flag risky actions, bias assessments, red-team findings (results from people who deliberately try to break the system), alignment controls that keep AI behavior

consistent with its intended purpose, and explainability configurations that document how well the AI can explain its own decisions. Every AI system in production must have at least one linked trust artifact in PRISM/AI. That's a constraint I enforced deliberately, and in my opinion an undocumented AI trust posture is the infrastructure equivalent of running with no firewall rules.

Observability covers production measurement factors including accuracy, latency, cost, drift, override rates, confidence thresholds, drift detectors, alert rules, AI incident records, and trace configurations.

PRISM/AI recognizes five categories of AI-specific architectural risk that are not present in traditional software: model drift (distributional shift between training and production inputs), hallucination or fabrication (confident but factually incorrect generative output), demographic bias (model outcomes that differ unfairly across groups), adversarial input (prompt injection and evasion), and training data leakage (model memorization of sensitive training data). These categories run through both Trust and Observability, and connect to PRISM/CY for threat documentation.

PRISM/A: Application Architecture Framework

PRISM/A adopts Domain-Driven Design (Evans, 2003) as its organizing principle because DDD's bounded context concept gives application architecture a natural unit of ownership that maps cleanly to team structure. A bounded context is a named, interconnected area of the application domain with clear ownership and its own universal language. Services, events, and APIs within and between domains are described explicitly therefore making cross-domain coupling a named architectural decision rather than an undocumented implementation choice that nobody maintains.

PRISM/A organizes application architecture across five dimensions.

Domains define the bounded contexts and each domain has a universal language declaration, a reference to its stakeholder owner, and a classification. Core domains are the business's differentiators and deserve internal investment. Supporting domains enable core capabilities and may rely on package software. Generic domains are commodity functions and should be handled with SaaS tools. This classification guides how much investment each domain should receive. For example, if a team spends significant engineering effort on a domain classified as generic, that mismatch is worth flagging at the landscape level.

Components describes the building blocks within domains, which includes services, user interfaces, background workers, AI models, integration adapters, and third-party platforms along with tech stack, AI system type, and trigger mode fields.

Integrations captures the cross-domain contracts with producer and consumer domain references, coupling type (synchronous, asynchronous, or batch), schema registry status, and versioning scheme. Undocumented cross-domain coupling is the primary source of the 'nobody knows what will break' problem in large application estates and PRISM/A records it.

Quality makes architectural health visible in the landscape with test coverage, SLO compliance, technical debt level, reliability, and security posture which are landscape artifacts alongside functional description. A component with critical technical debt is an architectural risk. Recording it here, not only in a backlog would mean it shows up when someone asks what a given system depends on.

Evolution captures the architectural trajectory and covers Architectural Decision Records, migration programs, roadmap items, and deprecation notices. ADRs reference git commit

identifiers to ensure decisions are traceable from the landscape artifact back to the commit where the decision was made and reviewed.

PRISM/D: Data Architecture Landscape Framework

DAMA-DMBOK addresses data management comprehensively as a discipline. PRISM/D is narrower by design as it is a landscape view of the data architecture which covers who owns what data, how it moves, where it lives, what agreements govern its exchange, and who can access what under what rules. The two are complementary and PRISM/D describes the structural decisions and DAMA-DMBOK covers the management discipline within them.

PRISM/D puts Data Mesh principles (Dehghani, 2022) into operational practice. Domain ownership is the organizing concept where each domain owns its source data, publishes data products for consumers, and is accountable for those products' quality and availability. A data product in PRISM/D is versioned, designed for a specific consumer, and served with a declared schema contract and SLA. The distinction between a data product and a raw dataset matters operationally as it is the difference between something with an owner and something without one.

PRISM/D organizes data architecture across five dimensions.

Domains defines bounded data ownership contexts.

Flows layer covers pipelines, event streams, synchronization processes, and data APIs that move data between domains, platforms, and consumers.

Platforms layer covers storage and compute systems such as data lakes, data warehouses, feature stores, stream platforms, caches, and lakehouses along with data classification ceiling and compliance scope fields.

Contracts layer formalizes producer-consumer agreements with schema version, quality thresholds, SLAs, and enforcement level (advisory, enforced, or breaking). Contract state follows a lifecycle which include stages: draft, proposed, agreed, enforced, breached, or retired. A breached contract is an architectural incident and PRISM/D treats it as such by design.

Governance covers PII and SPII registers, access policies, retention rules, lineage maps, and compliance controls. Privacy classifications like PII, SPII, PHI, PCI can appear as fields throughout PRISM/D on domains, flows, platforms, and contracts. Regulatory compliance obligations follow data wherever it flows, not only where it rests at rest, and the landscape reflects that.

PRISM/MVP: Product Validation Landscape Framework

Lean Startup (Ries, 2011) gave the field the build-measure-learn cycle. OKRs (Doerr, 2018) provide outcome-based goal structures. Shape Up offers a bet-based planning model. But none of them provides a structured, version-controlled artifact where hypotheses are named, tracked through validation, and preserved as organizational learning whether they are confirmed or refuted.

PRISM/MVP's organizing principle is that hypotheses are the unit of work. Every capability built is a bet on an underlying hypothesis. PRISM/MVP makes those hypotheses explicit, versioned, and attached to validation evidence through the git history.

Invalidated hypotheses are not deleted. They're marked invalidated with a resolution date and linked as a learning artifact. I made this a hard design principle because deleted hypotheses are invisible learnings that which the organization repeats them.

PRISM/MVP organizes product validation across five dimensions.

Market describes who the organization is building for, using Jobs-to-Be-Done framing (Christensen, Hall, Dillon, & Duncan, 2016): 'When [situation], I want [motivation], so I can [outcome].' Segments are classified as primary (optimized for), secondary (benefiting but not primary focus), or negative (explicitly out of scope). A negative segment is an architectural boundary that should be named, not assumed.

Value captures the problems being solved and the value propositions responding to them.

Bets layer formalizes investment decisions: each bet carries rationale, a linked hypothesis, and a declared opportunity cost.

Hypotheses layer makes assumptions explicit along with type (desirability, viability, feasibility, usability, or ai-behavior), risk level, and validation criteria.

Signals tracks what's being learned through metrics, experiments, results, learnings, and pivots.

The ai-behavior assumption type addresses a category of product assumption that traditional product frameworks don't cover like evaluating whether an AI component will perform accurately enough for users to trust it, whether its outputs are fair across user groups, and whether its failure modes are acceptable. These carry critical risk by default because AI feature failures are often invisible until user trust is already gone. Bets that depend on ai-behavior assumptions link to PRISM/AI trust guardrails for validation evidence which is a traversable chain from product hypothesis to AI architecture control.

Cross-Framework Integration

The PRISM frameworks are designed to be independent and interoperable. You can adopt PRISM/CY without PRISM/AI. PRISM/D stands alone. Each provides value without requiring the others to be in place first.

Interoperability works through soft link fields, which is a field on an artifact holding the ID of a related artifact in another framework. These are navigable references, and not schema-enforced foreign keys. No framework imports another's schema. Adopting one doesn't create a dependency on the tooling or adoption state of another.

PRISM EA's Reality/systems artifacts serve as the anchors of the link graph. Every other framework connects through them. PRISM/I runtime services link to EA systems. PRISM/CY surfaces link to those systems and to PRISM/I control artifacts. PRISM/A components link to EA systems and PRISM/I runtime artifacts. PRISM/AI systems link to PRISM/A components. PRISM/D flows and platforms link to PRISM/I state artifacts. PRISM/MVP bets link to PRISM/A components and EA systems.

Blast radius analysis is about identifying which artifacts are affected when a dependency changes or retires and can be followed across all seven frameworks through this link graph. That traversal is currently manual. Purpose-built tooling would make it automatic; see Section 13 on tooling gaps.

Table 3

Cross-Framework Reference Map

Framework	**Links From**	**Links To**
PRISM/I	EA Reality/systems	EA P/geographies, P/environments
PRISM/CY	EA Reality/systems; PRISM/I Control	EA P/stakeholders
PRISM/AI	EA Reality/capabilities;	PRISM/A Components;

	PRISM/D Domains	PRISM/CY Threats
PRISM/A	EA Reality/systems; PRISM/I Runtime	PRISM/CY Surface; PRISM/AI Systems
PRISM/D	EA Signal/data_contracts; PRISM/I State	PRISM/AI Models (training data)
PRISM/MVP	EA Intent/goals; EA P/stakeholders	PRISM/A Components; PRISM/AI Trust

Note. Links are soft references — no schema enforcement across frameworks. EA = PRISM Enterprise Architecture.

Application Across Domains

PRISM was designed to work outside information technology. Whether it actually does at scale is an empirical question and the scenarios below are illustrative, not case studies. They're meant to show where the domain-neutral language holds and where the interesting tensions appear.

Healthcare

A hospital network could use PRISM EA's People and Place layer to model clinical staff roles, patient populations, HIPAA-governed regulatory zones, and operating environments like ICU, general ward, and outpatient. The Reality layer would capture clinical workflows, formularies, and patient record systems. Signal would document HL7 FHIR interfaces between the EHR and pharmacy systems, consent protocols, and patient referral processes. Movement would capture clinician adoption of new EHR modules, change fatigue following organizational restructuring, and the trajectory from paper-based to digital records.

PRISM/D would govern the data domain with HIPAA-scoped PHI registers and retention rules. PRISM/AI would document clinical decision support models with bias assessments across patient demographic groups.

There's a real tension in a healthcare application worth naming: PRISM's authoring philosophy which states that anyone on the team should be able to create and modify landscape artifacts and that runs directly into clinical environments where authoring authority is tightly controlled for regulatory and liability reasons. Whether that tension is resolvable or represents a genuine scope boundary for PRISM isn't clear without field validation.

Manufacturing

A multi-site manufacturer could use the People and Place layer to model factory workers, line supervisors, and supplier relationships across regional sites. The Reality layer would capture production lines as systems, bills of materials as data assets, and lean production workflows as processes. Signal would document EDI advance ship notices with suppliers and manufacturing execution system integration events. Movement would track workforce readiness for process changes and the trajectory toward an optimized target factory layout.

PRISM/I would document OT/IT network convergence and production control systems, which is a particularly interesting application because operational technology carries different failure modes, update cycles, and regulatory contexts than enterprise IT. PRISM/MVP would govern validation of new manufacturing capabilities, with bets linked to feasibility hypotheses about production line retooling.

Limitations and Future Work

Five limitations are worth naming directly.

Empirical Validation

PRISM has not been validated through longitudinal adoption studies. The reasoning behind the source-first, git-native approach is grounded in adoption research (Davis, 1989; Armenakis et al., 1993) and genuine practitioner experience. It hasn't been systematically measured. Until it has, PRISM is a practitioner proposal and a well-reasoned one, but not an empirically validated standard. Future work needs case studies in both IT and non-IT domains with standardized outcome measurement against adoption rates, artifact quality, and governance effectiveness.

Tooling

The framework specifications are complete at the schema and authoring convention level. Purpose-built tooling such as a CLI for schema validation, a landscape renderer, a cross-framework blast radius analyzer, a query interface does not yet exist as a unified suite. Current validation uses third-party JSON Schema validators like ajv and standard YAML-aware editors through prism-framework site demo. The specification is ahead of the tooling. Purpose-built tooling would lower the authoring barrier further and enable cross-framework traversal that manual cross-referencing can't match.

Maturity Models

No PRISM framework currently defines a maturity model for landscapes or individual layers. Organizations adopting architecture frameworks need guidance on what partial adoption means and how to progress toward fuller coverage. A shared maturity model aligned with progressive commitment is planned for the next version of each framework.

Qualitative Dimensions

The Pulse sub-dimension of Movement like adoption signals, sentiment, change readiness, ethics markers and analogous qualitative dimensions in other frameworks rely on

human assessment that cannot be schema-validated. PRISM provides structure and vocabulary for these dimensions but doesn't specify validated measurement tools. The practical result is that different teams will capture them with different rigor. Future work should develop standardized assessment tools that produce PRISM-compatible structured data.

Multi-Landscape Federation

Large organizations may need multiple PRISM landscapes i.e., one per major bounded domain or business unit along with cross-landscape dependencies. No framework currently specifies a federation model for relating multiple landscapes to each other. This is a known gap for conglomerate enterprises and platform-and-domain architectures where one landscape's decisions carry architectural implications for others and PRISM lays foundation to accommodate it.

Conclusion

The problem PRISM was built to solve is specific: architecture work that lives in the same repository as the code it describes, governed by the same tools the engineering team already uses, and authorable by anyone with a text editor and a YAML-aware IDE.

I want to be clear about what that means and what it doesn't. PRISM doesn't argue that existing frameworks are wrong. Zachman's ontological contributions are reflected in PRISM's layer questions. TOGAF's temporal model informed the baseline/transition/target structure. NIST CSF shaped PRISM/CY's Controls dimension. Data Mesh principles shaped PRISM/D. Domain-driven design organized PRISM/A. These are foundations, not targets.

What PRISM argues is that there's a context that’s growing, and practically important where architectural discipline needs to integrate with engineering workflows rather than run

alongside them. Where the team that builds the system also maintains its architectural description. Where 'current architecture' means the main branch, not a SharePoint folder.

The frameworks are version 1.0. The intent is to refine them through adoption, not to deliver a final answer. Several gaps are named in Section 13: empirical validation hasn't happened yet, the tooling lags behind the specification, maturity models don't exist yet. These are real limits, not rhetorical humility. The framework family is published as open-source artifacts including specifications, JSON Schemas, and authoring templates and very specifically to invite the community feedback that will close them.

Architecture frameworks that can't evolve aren't frameworks, they're just artifacts. PRISM is designed to be updated, on its own is a complete enterprise architecture framework that is lightweight, scalable, measured like Zachman framework.

References

Armenakis, A. A., Harris, S. G., & Mossholder, K. W. (1993). Creating readiness for organizational change. Human Relations, 46(6), 681–703. https://doi.org/10.1177/001872679304600601

Christensen, C. M., Hall, T., Dillon, K., & Duncan, D. S. (2016). Know your customers' 'jobs to be done.' Harvard Business Review, 94(9), 54–62.

DAMA International. (2017). DAMA-DMBOK: Data management body of knowledge (2nd ed.). Technics Publications.

Davis, F. D. (1989). Perceived usefulness, perceived ease of use, and user acceptance of information technology. MIS Quarterly, 13(3), 319–340. https://doi.org/10.2307/249008

Dehghani, Z. (2022). Data mesh: Delivering data-driven value at scale. O'Reilly Media.

Doerr, J. (2018). Measure what matters: How Google, Bono, and the Gates Foundation rock the world with OKRs. Portfolio/Penguin.

European Parliament. (2024). Regulation (EU) 2024/1689 of the European Parliament and of the Council laying down harmonised rules on artificial intelligence (Artificial Intelligence Act). Official Journal of the European Union, L 2024/1689.

Evans, E. (2003). Domain-driven design: Tackling complexity in the heart of software. Addison-Wesley.

International Organization for Standardization. (2022). ISO/IEC 27001:2022 Information security, cybersecurity and privacy protection — Information security management systems — Requirements. ISO.

International Organization for Standardization. (2023). ISO/IEC 42001:2023 Information technology — Artificial intelligence — Management system. ISO.

National Institute of Standards and Technology. (2018). Framework for improving critical infrastructure cybersecurity, version 1.1. https://doi.org/10.6028/NIST.CSWP.04162018

National Institute of Standards and Technology. (2023). Artificial intelligence risk management framework (NIST AI 100-1). https://doi.org/10.6028/NIST.AI.100-1

Ries, E. (2011). The lean startup: How today's entrepreneurs use continuous innovation to create radically successful businesses. Crown Business.

Ross, J. W., Weill, P., & Robertson, D. C. (2006). Enterprise architecture as strategy: Creating a foundation for business execution. Harvard Business School Press.

The Open Group. (2018). The TOGAF® standard, version 9.2. The Open Group.

The Open Group. (2019). ArchiMate® 3.1 specification. The Open Group.

Zachman, J. A. (1987). A framework for information systems architecture. IBM Systems Journal, 26(3), 276–292. https://doi.org/10.1147/sj.263.0276

Appendix A

PRISM Framework Family — Member Frameworks at a Glance

Table A1

PRISM Framework Family: Member Frameworks

Framework	Full Name	Dimensions	Primary Focus
PRISM	Enterprise Architecture	P, R, I, S, M (5 layers)	Enterprise-wide architectural landscape; human adoption built in
PRISM/I	Infrastructure Landscape	Who, Where, Runtime, State, Control, Operations (6)	Infrastructure governance — vendor-agnostic, ownership-tracked
PRISM/CY	Cybersecurity Landscape	Identity, Surface, Controls, Threats, Posture (5)	Security posture as a navigable, versioned landscape
PRISM/AI	AI Architecture Landscape	Models, Systems, Lifecycle, Trust, Observability (5)	AI systems as architectural assets; trust and drift governance
PRISM/A	Application Architecture	Domains, Components,	DDD-organized application

		Integrations, Quality, Evolution (5)	landscape; quality as a landscape dimension
PRISM/D	Data Architecture Landscape	Domains, Flows, Platforms, Contracts, Governance (5)	Data mesh operationalized; regulated data as a classification throughout
PRISM/MVP	Product Validation Landscape	Market, Value, Bets, Hypotheses, Signals (5)	Hypothesis-driven validation with versioned learning history

Note. All frameworks share YAML/JSON Schema authoring conventions, git-native governance, and the baseline/transition/target temporal model. Version 1.0 for all.