Normalized Perimeter Foundry

NPF

An Architectural Pattern for Secure, Scalable, Assembly-Line Software Systems

Matthew Simpson
Founder & CEO, Orange Sky Software
2026

Abstract

The Normalized Perimeter Foundry (NPF) is a software architectural pattern that synthesizes well-established engineering principles into a coherent, named philosophy for building secure, scalable, and maintainable information systems. NPF establishes a hard perimeter through which all user-submitted data must pass before entering the system — but critically, raw user data never crosses that perimeter. Instead, it is transformed at the boundary into normalized, confidence-scored, unified data structures. Only these system-generated structures flow inward.

Inspired by Henry Ford's assembly line principle, NPF extends this metaphor beyond runtime architecture into development methodology itself — organizing teams and responsibilities as discrete, replaceable stations along a production line, each honoring a clear contract with adjacent stations.

While no individual component of NPF is new, the specific synthesis — hard perimeter, transformation-not-interpretation, unified structures, confidence scoring, assembly-line development organization, and serverless compute alignment — represents a distinct and nameable pattern not previously formalized in the literature.

NPF pairs naturally with Concentric Asynchronous Event-Driven (CAED) architecture for inner-layer processing, and with serverless compute primitives — AWS Lambda, ECS Fargate, and AWS Batch — at the perimeter stations. This combination produces a system whose infrastructure operating expenditure trends toward zero at rest, scaling cost directly with workload rather than reservation.

1. Introduction

Software architecture has cycled through many paradigms over the past several decades: three-tier client-server, service-oriented architecture (SOA), microservices, event-driven architecture, hexagonal/clean architecture, and reactive systems. Each paradigm has offered genuine improvements in specific dimensions — deployment topology, team boundaries, failure isolation, or communication patterns.

Yet beneath each successive paradigm, a persistent truth reasserts itself: information systems always resolve to something that receives input, processes it, and stores it. The labels and topologies change; the fundamental concerns do not.

NPF accepts this reality and works with it rather than against it. It does not attempt to abolish layering. Instead, it imposes a specific, opinionated constraint on what may cross the system perimeter — and organizes both runtime architecture and development methodology around that constraint.

> *"The individual components of NPF are not new. The synthesis is."*

2. The Core Principle

The defining constraint of NPF is stated simply:

> ***No user-submitted data — images, video, audio, documents, or any artifact created outside the system — crosses the perimeter in its raw form. Ever.***

This is not a policy enforced through access controls or compliance procedures. It is a structural guarantee enforced by architecture. The perimeter is the last place raw user data exists. What crosses inward are only system-generated artifacts: normalized, validated, confidence-scored data structures derived from the raw input.

This distinction — transformation rather than forwarding — is the philosophical heart of NPF.

3. Architecture Overview

3.1 The Perimeter Layer (The Foundry Floor)

The perimeter layer is the most critical component of an NPF system. It is not a passive gateway or a simple adapter. It performs substantive work:

- Antivirus and malware scanning of all inbound artifacts
- Format validation and schema enforcement
- Transformation of raw artifacts into unified data structures
- Confidence scoring of each transformed element
- Rejection or flagging of low-confidence transformations

The foundry metaphor is intentional and precise. Raw ore enters the foundry. Finished, standardized components leave. The foundry floor is the only place where the messy, unpredictable raw material exists. Downstream systems never handle ore.

3.2 The Unified Data Structure

All data that crosses the perimeter must conform to a unified schema defined by the system. This schema is the contract between the perimeter and all inner layers. Because inner layers only ever receive data in this unified form, they are:

- **Source-agnostic:** indifferent to whether input was a PDF resume, a video introduction, or an audio file
- **Format-agnostic:** no parsing logic exists inside the system boundary
- **PII-surface-minimized:** the raw artifact, with all its unstructured personal information, never enters the core

The unified structure carries not only the normalized data but also confidence metadata. A skills array extracted from a resume carries a confidence score per element. Low-confidence elements may be

excluded, quarantined, or passed with explicit markers — but they are never silently promoted.

3.3 Inner Layers

Inner layers — business logic, domain processing, state management — operate exclusively on unified structures. They are remarkably clean as a result. There is no format variance, no encoding ambiguity, no PII surface area. The inner system, in a meaningful sense, has never met a user. It has only met system-generated representations of users.

This maps naturally onto a concentric asynchronous event-driven topology: events carry unified structures inward for processing and outward as notifications. Each ring processes what it receives without concern for what the original artifact looked like.

3.4 Data Flow

Stage	What Happens	What Passes
User Submission	Document, image, video, audio received	Raw artifact (stays in perimeter)
AV Scan	Malware and virus scanning	Clean artifact only
Validation	Format and schema checks	Valid artifact only
Normalization	Transform to unified structure	Unified structure + confidence scores
Confidence Gate	Filter or flag low-confidence elements	High-confidence unified structure
Inner System	Business logic, domain processing	System-generated events and state

4. The Assembly Line Development Methodology

4.1 Ford's Principle Applied to Software

Henry Ford observed that the assembly line's greatest contribution was not speed — it was standardization enabling specialization. Each station on the line receives a known input in a known state, performs one operation, and passes a known output to the next station. No station needs to understand the whole car.

NPF applies this principle not only to runtime data flow but to the organization of development itself. Each processing stage in the perimeter maps directly to a discrete development station with:

- A defined input contract
- A single, bounded responsibility
- A defined output contract

• No dependency on what came before or after

4.2 Benefits for Development Teams

Property	Traditional Development	NPF Assembly Line
Onboarding	Requires system-wide context	Station contract is sufficient
Blast radius	Failures propagate unpredictably	Contained within the station
Replaceability	High coupling, risky to swap	Swap any station independently
Parallelization	Coordination overhead	Stations developed concurrently
Quality gates	Cultural, inconsistently applied	Structural, enforced by contract
Testing	Integration-heavy	Station tested in isolation

4.3 Stations Are Not Microservices

It is important to distinguish NPF stations from microservices. A microservice is a deployment boundary. An NPF station is a responsibility boundary. Stations may be deployed as microservices, as modules within a monolith, or as pipeline stages within a single service. The deployment topology is a separate concern. The station contract is the invariant.

5. Comparison to Existing Patterns

NPF draws from and relates to several well-established patterns. Understanding these relationships clarifies what NPF adds.

Pattern	Relationship to NPF	What NPF Adds
3-Tier Architecture	NPF is a 3-tier variant	Hard constraint: raw data never crosses perimeter
Hexagonal / Clean Arch.	Shares ring/layer dependency rule	Foundry metaphor + confidence scoring + data sovereignty
Event-Driven Architecture	NPF inner layers are event-driven	Perimeter normalization before events are emitted
CQRS / Event Sourcing	Compatible and complementary	NPF governs what enters the event stream

Zero Trust Security	Shares distrust of inbound data	Architectural enforcement, not policy enforcement
Pipeline / Filter Pattern	Perimeter is a pipeline	Confidence gating + unified schema as system-wide contract

> *NPF's contribution is not invention of components. It is the synthesis of these components into a coherent philosophy with a specific, opinionated constraint at its center: raw user data never crosses the perimeter.*

6. Implementation Guidance

6.1 Define the Unified Schema First

Before writing any perimeter code, define the unified data structure. This schema is the most important artifact in an NPF system. It is the contract that everything else depends on. Treat it as you would a public API — version it, document it, and change it deliberately.

6.2 Build the Perimeter as a Pipeline

Implement the perimeter as an explicit pipeline of discrete stations. Each station should be:

- A pure function where possible: given input, produce output, no side effects
- Independently testable with its own test suite
- Replaceable: changing the AV engine should not affect the normalization station
- Instrumented: measure throughput, rejection rate, and confidence distribution per station

6.3 Treat Confidence as a First-Class Citizen

Every element in the unified structure should carry a confidence score. Define explicit policies for what confidence threshold is required for an element to flow inward. These policies should be configurable, auditable, and versioned alongside the schema.

Low-confidence elements are not failures — they are signals. A resume with a low-confidence skills extraction is useful information. Route it appropriately rather than discarding it silently.

6.4 Never Let the Inner System Handle Raw Artifacts

This is the invariant that must never be violated. If inner system logic requires access to a raw artifact for any reason, the correct response is to enrich the unified schema — not to pass the artifact. The perimeter extracts more information; it does not open a side channel.

6.5 Organize Teams Around Stations

Assign ownership of each perimeter station to a discrete team or individual. That team is responsible for the station's input contract, its logic, and its output contract. They should be able to deploy their station independently, test it in isolation, and be replaced without disrupting adjacent stations.

This is the assembly line principle applied to engineering organization. It is not Conway's Law by accident — it is Conway's Law by design.

6.6 Audit the Perimeter Relentlessly

Because the perimeter is the only place raw data exists and the only place transformation occurs, it is also the only place where systematic bias, lossy parsing, or extraction error can be introduced. Audit the perimeter continuously. Monitor confidence distributions. Alert on drift. The integrity of everything downstream depends on the integrity of the foundry.

7. Serverless Compute and Infrastructure OpEx

7.1 The Natural Fit

NPF's perimeter stations are discrete, bounded, stateless, and event-triggered by design. These properties are precisely the characteristics that serverless compute platforms are built to exploit. The alignment is not incidental — it is structural.

NPF Station	Serverless Primitive	Rationale
AV Scan	AWS GuardDuty / Lambda	Short-lived, event-triggered, stateless
Validation	AWS Lambda	Lightweight, high-frequency, parallelizable
Normalization	AWS Lambda / ECS Fargate	Variable duration; Fargate for heavy media
Confidence Scoring	AWS Lambda	Compute-bound, isolated, replaceable
Batch / bulk ingestion	AWS Batch	Long-running jobs, managed parallelism

7.2 CAED Inner Layers

The inner system — organized as Concentric Asynchronous Event-Driven (CAED) architecture — reinforces the serverless model. CAED inner rings activate only on events, not on timers or persistent connections. Combined with NPF's perimeter, the result is a system that is:

- **Idle when not processing** — no standing compute costs at rest

- **Instantaneously elastic** — scales from zero to parallel processing on demand
- **Cost-proportional** — infrastructure expenditure mirrors actual workload, not provisioned capacity

This is the serverless promise realized through architectural discipline rather than wishful configuration.

7.3 Slow-to-No Burn Infrastructure OpEx

Traditional architectures provision for peak load and pay for idle capacity continuously. NPF + CAED + serverless inverts this model entirely. At zero submissions, the system burns near zero. At peak load, it scales horizontally across Lambda invocations, Fargate tasks, and Batch jobs — then returns to zero. There is no idle fleet, no reserved instance commitment required for the perimeter processing layer.

The assembly line metaphor holds here too: Ford's line didn't run when there was nothing to build. NPF's foundry doesn't burn when there is nothing to process.

7.4 Primitive Selection Guide

AWS Lambda — the default choice for all stateless, short-duration perimeter stations. AV scanning wrappers, schema validators, lightweight normalizers, and confidence scorers are natural Lambda workloads. Sub-second to low-second execution, event-triggered, scales to thousands of concurrent invocations automatically.

AWS ECS Fargate — appropriate for normalization stations handling large media artifacts: video transcription, audio processing, image analysis. Where Lambda's execution window or memory ceiling is insufficient, Fargate provides containerized processing without server management, still scaling to zero between jobs.

AWS Batch — appropriate for bulk ingestion scenarios: large-volume imports, batch media processing, scheduled confidence re-scoring of existing unified structures. Batch manages job queuing, parallelism, and retry logic, and terminates compute on job completion.

7.5 The OpEx Implication for Product Teams

Infrastructure cost becomes a variable tied to revenue-generating activity rather than a fixed overhead. A system processing no submissions costs effectively nothing to run. A system processing millions of submissions scales its cost proportionally — and that cost is covered by the value being generated.

This shifts the infrastructure conversation from "how much does it cost to run?" to "how much does it cost per unit of work?" — a fundamentally more honest and manageable question.

> *Privacy by topology. Cost by consumption. Both enforced by architecture, not by policy.*

8. AI-Native Development Velocity

8.1 The Unexpected Beneficiary

NPF was designed around human engineering principles — Ford's assembly line, discrete station contracts, bounded responsibilities. It turns out these same properties make NPF the optimal architecture for a development team that is not entirely human.

AI engineering agents — non-human developers capable of generating, testing, and iterating on code at machine speed — do not want monoliths. They want exactly what NPF provides: narrow scope, clear contracts, isolated testability, and no requirement to understand the whole system. NPF hands all four to an AI agent by design.

8.2 Why AI Agents Feast on NPF Stations

An AI agent's effectiveness degrades with context size and ambiguity. A monolithic codebase is a poor fit — too much surface area, too many interdependencies, too much implicit knowledge required. An NPF station is the opposite:

- **Narrow scope** — one station, one responsibility
- **Explicit input and output contracts** — the unified schema is the complete specification
- **Isolated testability** — no side effects to reason about, no upstream or downstream state to simulate
- **Self-contained** — the agent never needs context beyond its station walls

The station contract is not just a human communication tool. It is a machine-readable specification precise enough for an AI agent to implement against without ambiguity.

8.3 Parallel Agent Execution

Where a human team builds stations sequentially and coordinates handoffs, an AI engineering team operates differently. Each station can be assigned to a parallel agent simultaneously. Agents generate, test, and iterate within their station boundary at machine speed, independently, without blocking one another.

The assembly line metaphor deepens here. Ford's line ran stations in sequence because physical assembly required it. NPF's development line runs stations in parallel because software contracts permit it. AI agents collapse the sequential dependency that human coordination imposed.

8.4 The Compounding Effect

Dimension	What NPF + AI Delivers
Infrastructure cost	Scales to zero at rest
Development velocity	Scales to near-instant at contract definition
Operational cost	Tied to workload, not headcount or reservation

Quality	Station contracts enforced structurally, not culturally
Parallelism	Unlimited — agents per station, stations per system

8.5 Where Human Judgment Remains Essential

The compounding effect does not eliminate the need for human judgment — it concentrates it upstream. As AI agents absorb station implementation, velocity control moves entirely to system design and contract definition. These are precisely the activities that require human expertise, architectural intuition, and domain knowledge.

NPF, in this light, is not merely a pattern for today's systems. It is a pattern designed — whether intentionally or by structural inevitability — for a development model where humans define contracts and machines fulfill them.

> *The assembly line didn't eliminate skilled labor. It repositioned it where it mattered most. NPF does the same.*

9. Conclusion

The Normalized Perimeter Foundry is a named synthesis of established software engineering principles, organized around a single opinionated constraint: raw user data never crosses the system perimeter. What crosses inward are only system-generated, normalized, confidence-scored unified structures.

This constraint yields structural privacy by topology, not by policy. It yields clean inner layers free of format variance and PII surface area. It yields a development methodology that mirrors Ford's assembly line — discrete stations, clear contracts, independent replaceability, and measurable quality gates at each stage.

When deployed on serverless compute primitives — Lambda, ECS Fargate, and AWS Batch — paired with CAED inner-layer architecture, NPF yields infrastructure whose operating cost trends toward zero at rest. When developed by AI engineering agents operating against station contracts in parallel, NPF yields velocity that compounds with each additional agent.

NPF does not replace existing patterns. It synthesizes them. Its value is not in any individual component, but in the coherent philosophy that connects them — and in giving that philosophy a name precise enough to communicate, document, and build upon.

> *Normalized. Perimeter. Foundry. Each word is load-bearing.*

About the Author

Matthew Simpson is the Founder and CEO of Orange Sky Software, and the originator of the Normalized Perimeter Foundry (NPF) architectural pattern. Echosaw.com and Xojurn.com are Orange Sky Software products built using the NPF pattern. This whitepaper documents the synthesis of principles underlying NPF as developed through practical system design.