

Feature Flags as a Migration Engine

De-risking Domain-Driven Decomposition of Monolithic Payment Platforms

ARCHITECTURE

PAYMENTS

PLATFORM ENGINEERING

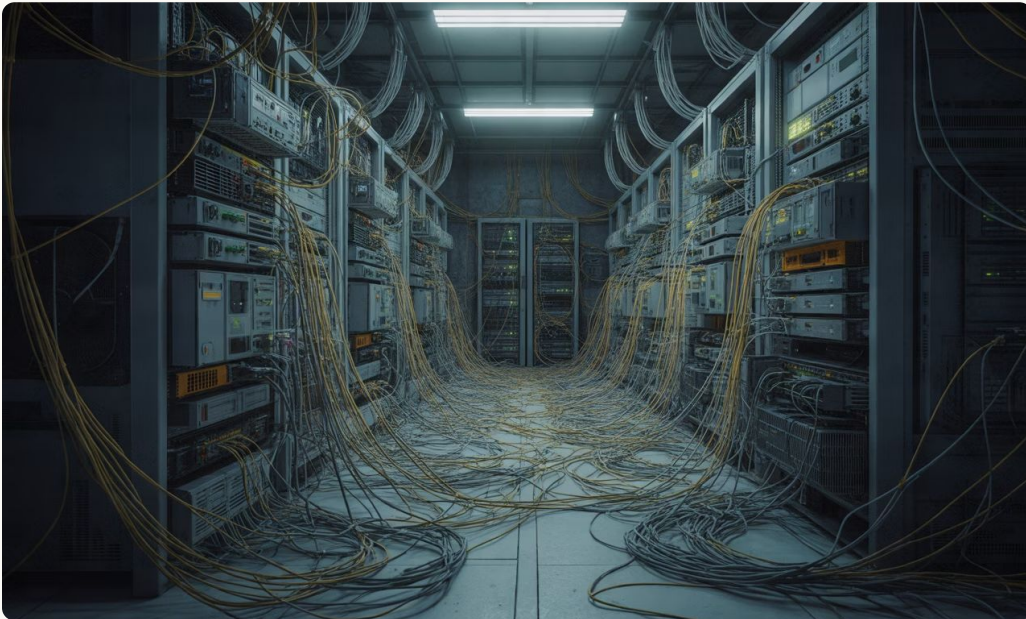
Presented by Jaya Chand Bandi

Director of Software Engineering, MerchantE



The Problem

Monoliths That Outlived Their Design



Enterprise payment platforms were engineered for centralized control, transactional reliability, and operational simplicity. For years, that bet paid off.

But as organizations layer in **digital wallets, new payment rails, loyalty programs, hosted checkout, and regional compliance requirements**, the monolith becomes a bottleneck - not a foundation.



All-or-nothing cutover rewrites carry existential risk for payment-critical systems. There is a safer path.

What This Session Covers

01

Domain-Driven Design Fundamentals

Bounded contexts, ubiquitous language, and context mapping across payment domains

03

Migration Pattern Taxonomy

Strangler Fig, Anti-Corruption Layer, and Branch by Abstraction

02

Feature Flags as a Control Layer

Routing traffic between legacy and new implementations during live decomposition

04

Sequencing & Risk Management

Extraction readiness framework, kill switches, and flag lifecycle discipline

Why Payment Monoliths Are Hard to Break

Transactional Integrity

ACID guarantees span authorization, settlement, and ledger writes in a single DB transaction

Implicit Domain Coupling

Business logic for dispute handling, loyalty accrual, and reporting shares tables and in-process calls

Operational Confidence

Decades of runbooks, alerts, and SLAs are calibrated to a single deployable unit

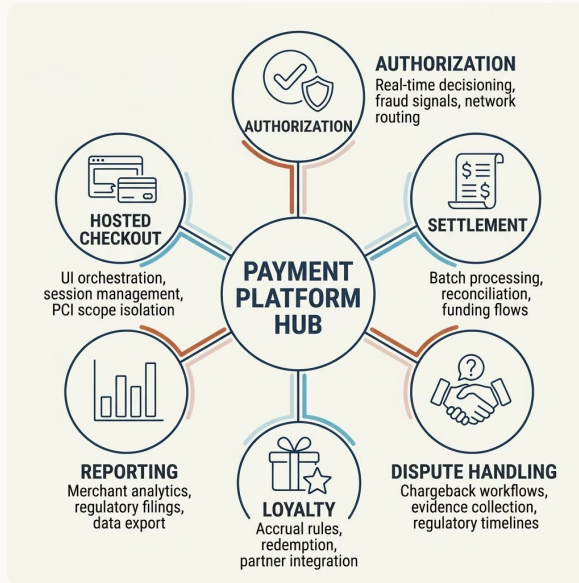


Chapter 1

Domain-Driven Design for Payment Platforms

Defining the boundaries before you break them

Identifying Bounded Contexts in Payments



DDD's bounded context concept maps naturally onto payment platform capabilities. Each context owns its **domain model, ubiquitous language, and data** - and should be independently deployable.

The challenge: in a monolith, these contexts bleed into each other through shared schemas, in-process calls, and implicit contract assumptions baked into business logic over years.

Ubiquitous Language Reveals Hidden Coupling

The Same Word, Different Meanings

In a monolith, the term "**transaction**" might mean an authorization event in the fraud engine, a ledger entry in settlement, a dispute trigger in chargeback handling, and a loyalty accrual event - all referencing the same database row, all with diverging semantics.

Context mapping forces teams to name these differences explicitly, exposing integration surfaces that need intentional boundaries - not accidental sharing.

Context Map Anti-Patterns to Watch

- Shared kernel masking divergent models
- Conformist relationships hiding upstream coupling
- Big Ball of Mud domains with no clear ownership
- Published language violations across team boundaries

Chapter 2

Feature Flags as the Migration Control Plane

Routing, validating, and reversing - without a big bang



Flags Enable Dual-Path Traffic Routing

The core mechanism: a feature flag sits at the domain boundary and routes each request - or each merchant, transaction type, or region - to either the legacy implementation or the newly extracted service.

Percentage Rollout

Gradually shift transaction volume from monolith to new service - 1%, 10%, 50%, 100%

Cohort Targeting

Route specific merchants, card networks, or regions to the new implementation first

Kill Switch

Instantly revert all traffic to the legacy path if the extracted service misbehaves under production load

A Taxonomy of Flag Types in Migration



Release Toggle

Controls whether a new service path is reachable at all. Used to deploy dark - code ships before traffic shifts.



Experiment Toggle

Runs shadow mode comparisons - both paths execute, responses are diffed, legacy wins. Validates behavioral parity before cutover.



Ops Toggle

Runtime kill switches for SRE teams. Decouples incident response from code deployment during production failures.



Permission Toggle

Enables new capabilities for specific merchant tiers or regulatory regions before general availability.

Chapter 3

Migration Pattern Taxonomy

Strangler Fig · Anti-Corruption Layer · Branch by Abstraction

The Strangler Fig Pattern

Phase 1: Intercept

Flag-controlled proxy captures edge traffic.

Phase 3: Retire

Decommission legacy path when flag is 100%.



Phase 2: Divert

Gradually route increasing volume to new service.

Named after the fig that slowly envelops and replaces its host tree, this pattern lets the new service grow around the monolith incrementally.

The flag-controlled proxy intercepts domain calls at a well-defined facade. New implementation grows in parallel; the monolith shrinks as traffic is diverted. **No coordinated cutover required.**

Anti-Corruption Layer & Branch by Abstraction

Anti-Corruption Layer (ACL)

A translation boundary that insulates the new domain model from the legacy data contracts. Flags control whether the ACL is active, enabling the new model to evolve independently while the monolith's schema remains unchanged during transition.

Branch by Abstraction

Introduces an abstraction layer over the component being replaced. The feature flag switches the concrete implementation injected at runtime - legacy or new - without branching in version control. This is the pattern most directly dependent on toggles to stay safe during active development.

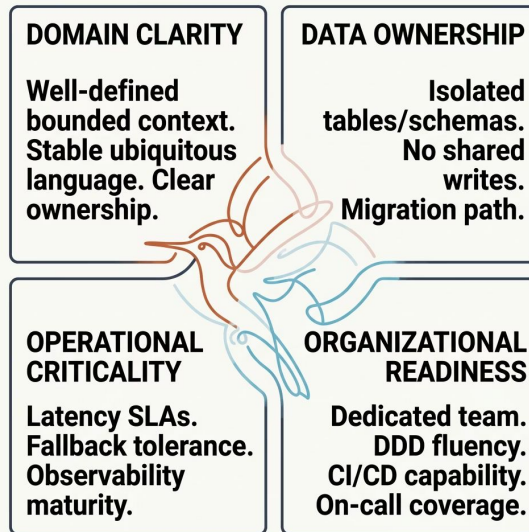
- ❏ All three patterns share a common requirement: the abstraction point must be identified and introduced *before* extraction begins. Flags enforce the seam, not create it.

Chapter 4

Sequencing Extraction: A Practical Framework

Not all domains are ready to move at the same time

Extraction Readiness Scoring



Score each candidate domain across four dimensions before committing to extraction. **High scores across all four** indicate a safe first extraction candidate - often a reporting or loyalty domain rather than the authorization core.

Start where you can afford to learn. Authorization is the last domain you extract, not the first.

Recommended Extraction Sequence

1

Authorization & Settlement Core

Highest criticality - extract last, after
pattern mastery is proven

2

Dispute Handling

Regulatory timelines and evidence
workflows - complex but bounded

3

Hosted Checkout

PCI scope isolation accelerates
extraction justification

4

Loyalty & Rewards

Often the cleanest bounded context - good early candidate

5

Reporting & Analytics

Read-heavy, tolerant of eventual consistency - ideal first
extraction

Chapter 5

Common Pitfalls & Organizational Friction

Where migrations stall - and how to recover

Four Failure Modes to Anticipate

False Decomposition

Services that share a database are distributed monoliths in disguise. Data ownership must precede service extraction - not follow it.

1

2

Over-Fragmentation

Nano-services with anemic domain models multiply operational overhead without delivering bounded context clarity. Conway's Law cuts both ways.

3

Flag Sprawl & Debt

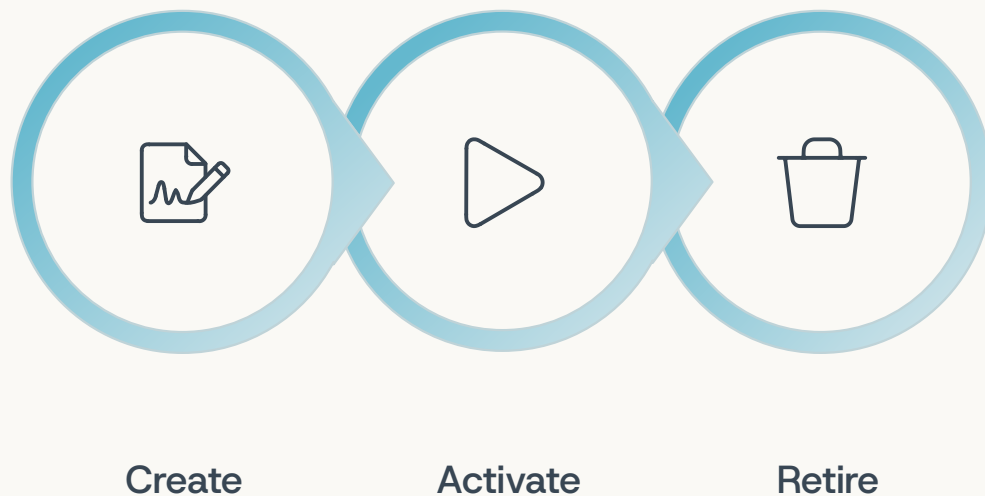
Flags without expiry dates become permanent conditionals. Every toggle needs an owner, a removal ticket, and a lifecycle policy enforced at the platform level.

4

Conway's Law Misalignment

Teams structured around technology layers - not business domains - will reproduce the monolith's coupling regardless of service boundaries drawn on an architecture diagram.

Flag Lifecycle Discipline



The Flag Is Not the Finish Line

Reaching 100% rollout is not completion. The legacy code path must be deleted and the flag removed from the codebase. Permanent flags are technical debt with a runtime cost.

- Register every flag with an owner and TTL at creation
- Automate staleness detection in CI pipelines
- Track flag count as an architectural health metric

Key Takeaways

Flags Enable Reversible Migration

Kill switches and percentage rollouts eliminate all-or-nothing cutover risk for payment-critical domains

DDD Defines the Seams, Flags Enforce Them

Bounded contexts and ubiquitous language must be established before extraction begins - flags operationalize the boundary, not create it

Sequence by Readiness, Not Ambition

Extract reporting and loyalty first.
Earn the right to touch authorization through demonstrated pattern mastery

Manage Flag Debt Aggressively

Every toggle needs an owner, a TTL, and a removal ticket. Flag sprawl is the hidden cost of a successful migration program



Thank You

Questions & Discussion

Feel free to connect, reach out, or continue the conversation after the session.