How to Identify Bounded Contexts

What is a Bounded Context?

A **Bounded Context** is a boundary within which a particular domain model (and its ubiquitous language) is defined and applicable.

Think of it as a **"sphere of meaning"** — inside this boundary, terms like "Order", "Customer", or "Product" have specific, consistent definitions. (Outside, they might mean something completely different).

STEP 1: Understand the Business — Strategic DDD

Start by learning the **business landscape**. Talk to domain experts. Ask:

- What are the main business capabilities or departments?
- Where do teams have different goals, language, or processes?
- Where do misunderstandings or translation happen between teams?

Example: In an e-commerce company:

- Sales team talks about "Orders" as confirmed purchases.
- Warehouse team talks about "Orders" as pick/pack/ship tasks. → These are likely **two different Bounded Contexts**: Sales and Fulfillment.

STEP 2: Look for "Ubiquitous Language" Clashes

When the **same term means different things**, or **different terms mean the same thing**, you've found a seam for a Bounded Context.

Example:

- Marketing: "Campaign" = promotional event with discounts.
- Finance: "Campaign" = cost center for budgeting. → Split into Marketing Campaigns and Financial Planning contexts.

Or: Two teams use different terms for the same concept:

- "User" vs "Member" vs "Customer" → Might indicate separate contexts, or need for alignment.

STEP 3: Follow Team/Organizational Boundaries (Conway's Law)

"Organizations which design systems... are constrained to produce designs which are copies of the communication structures of these organizations."

Look at:

- How teams are structured.
- Where ownership and responsibilities lie.
- Where handoffs or friction occur between teams.

If two teams never talk — or argue over models — they probably live in different Bounded Contexts.

Tip: Use **Team Topologies** or **Inverse Conway Maneuver** — align software boundaries with team communication paths.

STEP 4: Identify Subdomains (Problem Space)

Break the business problem space into **Subdomains**:

Subdomain Type	Description	Example
Core Domain	The key competitive advantage	Order personalization engine
Supporting	Important but not differentiating	User notifications
Generic	Common, solved problems — buy or outsource	Payment processing, Auth

Each **Subdomain often maps to one or more Bounded Contexts** (solution space).

Note: Subdomain = problem space. Bounded Context = solution space. They're related but not 1:1.

STEP 5: Look for Process or Lifecycle Boundaries

Where does a business process begin and end? Where does responsibility hand off?

Example: "Order" lifecycle:

1. Customer places order → Ordering context
2. Payment processed → Billing context

3. Inventory reserved → Inventory context
4. Shipment scheduled → Fulfillment context

Each stage may belong to a different Bounded Context.

STEP 6: Use Context Mapping (Tactical DDD)

Draw a **Context Map** to visualize:

- Each Bounded Context as a box.
- Relationships between them: Partnership, Customer/Supplier, Conformist, Anti-Corruption Layer, etc.

This reveals integration points and helps avoid model leakage.

Example Context Map Snippet:

[Ordering] ←→ [Billing] (Customer/Supplier)

[Billing] ←→ [LegacyPaymentSystem] (ACL)

[Ordering] ←→ [Inventory] (Partnership)

STEP 7: Validate with Real Use Cases

Test your Bounded Context boundaries:

- Can this context be developed, deployed, and evolved independently?
- Does it have a single, consistent model and language?
- Do all domain rules and invariants fit naturally inside it?
- Is the team responsible for it clearly identified?

If you answer "no" — reconsider the boundary.

Tools & Techniques to Help

Tool/Technique	Purpose
Event Storming	Collaborative workshop to map domain events & boundaries
Domain Storytelling	Visualize workflows to find context seams

User Journey Mapping	See handoffs and transitions in user experience
Impact Mapping / User Stories	Reveal scope and boundaries of features
CRC Cards (Class-Responsibility-Collaboration)	Model responsibilities within a context

Common Mistakes to Avoid

Mistake	Why It's Bad
One Bounded Context for entire system	Leads to "Big Ball of Mud" — no clear boundaries
Too many tiny contexts	Overhead, integration hell
Ignoring team/org boundaries	Creates friction and misalignment
Letting technical layers define boundaries	Bounded Contexts are about domain, not tech
Sharing domain models across contexts	Causes model pollution and coupling

Signs You've Found a Good Bounded Context

- ✓ Has a **clear purpose** aligned with business capability
- ✓ Uses a **consistent ubiquitous language**
- ✓ Can be **owned by one team**
- ✓ Has **explicit interfaces** to communicate with other contexts
- ✓ Can evolve **independently** without breaking others
- ✓ Contains **cohesive domain logic and invariants**

Example: E-Commerce System

Bounded Contexts:

- [Customer Management] → Manages profiles, preferences, authentication

- [Catalog] → Manages products, categories, pricing

- [Ordering] → Handles cart, checkout, order placement

- [Inventory] → Tracks stock levels, reservations

- [Billing] → Manages invoices, payments, refunds
- [Fulfillment] → Manages shipments, tracking, delivery
- [Marketing] → Manages campaigns, promotions, vouchers

Each has its own model:

- Order in Ordering ≠ Order in Fulfillment
- Product in Catalog ≠ Product in Inventory

→ Connected via well-defined contracts (APIs, events, ACLs).