iSAQB-DDD TRAINING

Day 5: Strategic Design 1

Cutting and Distinguishing Models

- Senior Engineers & Managers
- January 8, 2026
- Module: Strategic Design

Today's Focus

- **Problem Space**
  Understanding business domains
- **Solution Space**
  Designing bounded contexts
- **Cutting Models**
  Strategic separation techniques
- **Distinguishing Models**
  Context mapping strategies

Industry Case Studies

BMW, Toyota, Ford, Mercedes-Benz software architecture implementations

Learning Goals

Strategic Design 1: Cutting and Distinguishing Models

1. **Identify System Symptoms**
   Recognize signs of monolithic or distributed monolith systems
2. **Evaluate Model Coherence**
   Assess cross-team model alignment and consistency
3. **Apply Strategic Thinking**
   Use strategic methods to cut and separate domains
4. **Design Problem/Solution Space**
   Create clear separation between problem and solution domains
5. **Classify Subdomains**
   Distinguish between Core, Supporting, and Generic subdomains
6. **Create Bounded Contexts**
   Design context boundaries with consistent ubiquitous language
7. **Map Context Relationships**
   Visualize and define interactions between bounded contexts

By End of Day

- Analyze existing systems
- Apply cutting techniques
- Design bounded contexts
- Map relationships

Practical Skills

Hands-on exercises with real-world automotive industry examples

What is Cutting and Distinguishing Models?

Strategic Design Phase

The first stage of DDD where we separate the business problem from the technical solution

When to Apply

- Monolithic symptoms: slow deployments, tight coupling
- Distributed monolith: distributed but coupled
- Team scaling: communication overhead, bottlenecks
- Complex domain: business rule complexity

Key Benefits

Maintainability

Isolated changes, reduced coupling

Team Autonomy

Independent teams, clear boundaries

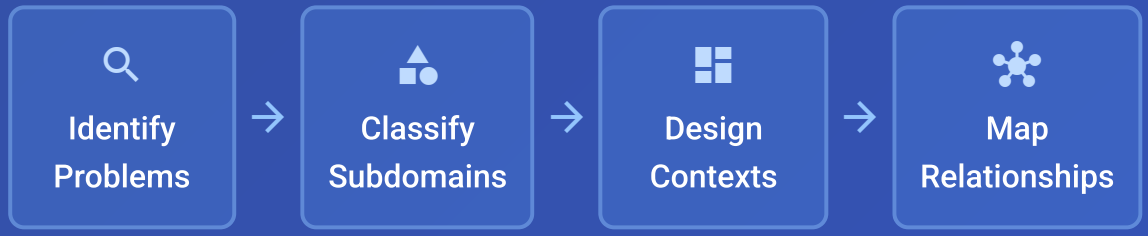
Reduced Complexity

Focused subdomains, simpler code

Business Alignment

Model reflects domain expertise

Strategic Design Process Flow



Problem Space vs. Solution Space

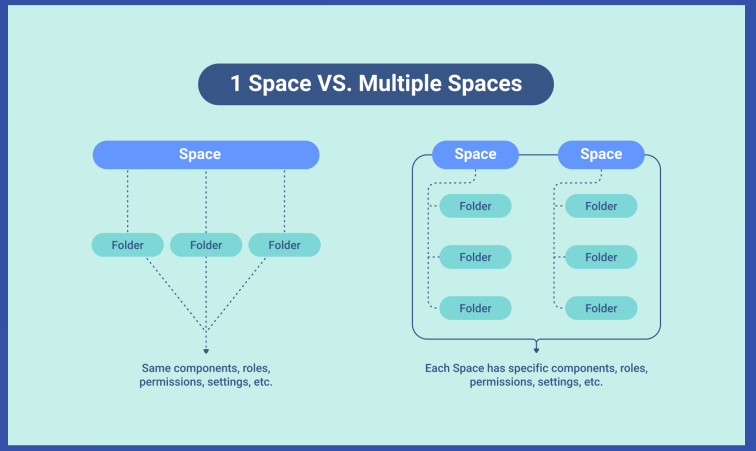
Problem Space

- **What:** Business problem being solved
- **Focus:** User needs and requirements
- **Question:** WHY we build this
- **Key:** Domain knowledge and business rules
- **Example:** "Vehicle needs autonomous driving"

Solution Space

- **What:** Technical implementation details
- **Focus:** Architecture and technology choices
- **Question:** HOW we build this
- **Key:** Code structure and components
- **Example:** "Implement sensor fusion with neural networks"

Visual Comparison



Automotive Domain Example

Problem Space

Enable vehicle to navigate autonomously in urban environments

Solution Space

Design perception, planning, and control microservices using AI/ML

Understanding Domains and Subdomains

Domain

Area of knowledge or activity - broad scope of business operations

Subdomain

Specific, specialized area within a domain - narrower focus

Bounded Context

Boundary of a model with consistent ubiquitous language

Hierarchy Visualization

Vehicle Domain

↓

Powertrain Subdomain

↓

Battery Management BC

Industry Examples

E-commerce Inventory Subdomain	Healthcare Patient Care Subdomain	Finance Trading Subdomain
Inventory Stock Level BC	Patient Care Treatment BC	Trading Order Book BC

Subdomain Classification

Core Subdomain

- Competitive advantage
- Unique business value
- Invest heavily here
- Key differentiator

Example: BMW's autonomous driving algorithm, Tesla's battery management system

Supporting Subdomain

- Supports core domain
- Not differentiator
- Can be outsourced
- Still important

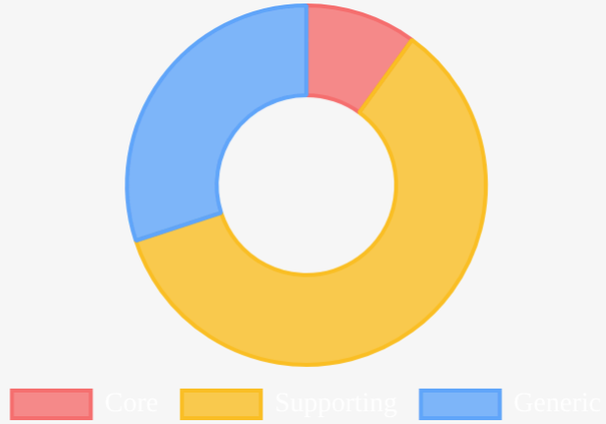
Example: Billing systems, customer support platforms, inventory management

Generic Subdomain

- Commodity solution
- Standard requirements
- Buy or build simply
- Minimal investment

Example: Authentication, logging, file storage, email services

Typical Distribution



Key Insight

- Focus 80% of investment on Core + Supporting
- Identify true Core domains to maximize ROI
- Minimize Generic domain effort with off-the-shelf

Bounded Contexts

Definition

Boundary within which a model is consistent and self-contained, with its own ubiquitous language

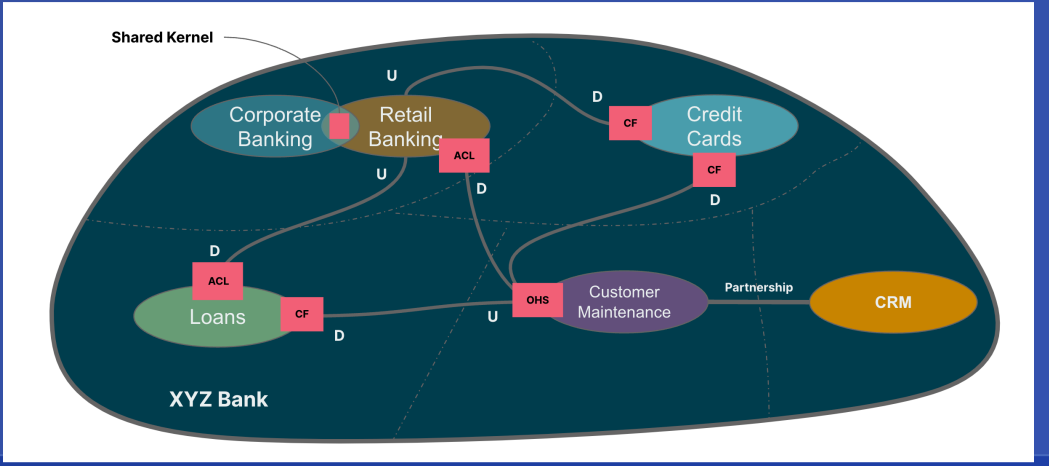
Purpose

- Separate Concerns
- Team Autonomy
- Integration Points

Identification

- Ubiquitous language consistency
- Business process boundaries
- Team responsibilities
- Data ownership

Banking System Example



Automotive Domain Examples

Vehicle Control

Engine control, brake system, transmission management

Infotainment

Navigation, multimedia, user interface

Telematics

Remote diagnostics, OTA updates, fleet management

Security

Access control, data encryption, safety systems

Powertrain

Battery management, motor control, energy distribution

Navigation

Route planning, traffic updates, location services

Cutting Models: Strategies and Techniques

1. **Domain Event Storming**
2. **Subdomain Classification**
3. **Bounded Context Identification**
4. **Ubiquitous Language Alignment**
5. **Context Mapping**

Automotive Domain Example

Event Storm	Subdomain	Contexts
VehicleStart, BatteryLow, NavigationUpdate	Powertrain (Core), Infotainment (Supporting)	BatteryMgmt, EngineCtrl, Multimedia

Do's

- Involve domain experts in storming
- Start with domain events, not UI
- Focus on business value proposition
- Align contexts with team boundaries
- Document ubiquitous language clearly

Don'ts

- Don't cut by technology boundaries
- Avoid monolithic thinking
- Don't ignore cross-cutting concerns
- Never skip the classification step
- Don't assume 1 subdomain = 1 BC

Distinguishing Models: How to Identify Boundaries

Practical Identification Methods

1. Language Analysis

Different terms for same concept? Different meaning for same term?

2. Business Process Boundaries

Where do handoffs occur? Who triggers processes?

3. Data Ownership

Who owns the data? Who creates/updates/deletes?

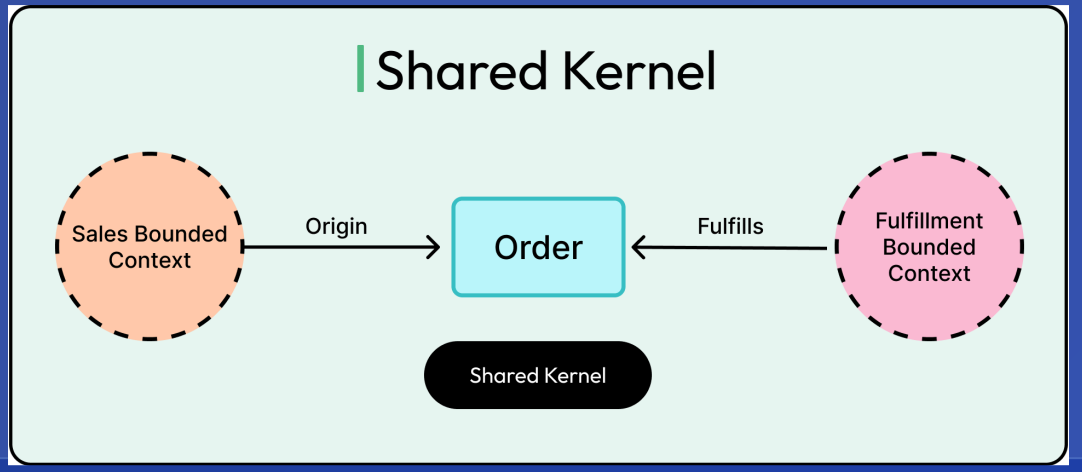
4. Team Responsibilities

Different teams? Different priorities?

5. Technology Constraints

Different tech stack? Different constraints?

Shared Kernel Example



Boundary Decision Matrix

✓ Separate Context

Different language, data ownership, or team

⚠ Shared Kernel

Core shared model with limited coupling

→ Anticorruption

Translation layer between contexts

≠ Separate Models

Same term, different meaning = different BC

Checklist:

✓ Domain terms differ? ✓ Business processes separate? ✓ Data owner differs? ✓ Team boundaries exist? ✓ Tech constraints differ?

The Strategic Design Process

1 Explore & Analyze

- **Domain Event Storming**
  Identify key events
- **Stakeholder Interviews**
  Gather domain knowledge
- **Business Process Analysis**
  Map flows and handoffs

2 Classify & Cut

- **Subdomain Classification**
  Core/Supporting/Generic
- **Bounded Contexts**
  Find natural boundaries
- **Ubiquitous Language**
  Define per context

3 Map & Integrate

- **Context Mapping**
  Visualize relationships
- **Relationship Definition**
  Patterns and integration
- **Integration Patterns**
  API/Event/Messaging

4 Validate & Refine

- **Team Alignment**
  Ensure buy-in
- **Technical Validation**
  Test boundaries
- **Iterative Refinement**
  Continuous improvement

Timeline

- Phase 1: 1-2 weeks
- Phase 2: 1-2 weeks
- Phase 3: 1-2 weeks
- Phase 4: 1-2 weeks

Key Deliverables

- Domain Event Catalog
- Subdomain Classification
- Bounded Context Map
- Ubiquitous Language Glossary

Success Criteria

- Clear context boundaries
- Team ownership defined
- Integration points clear
- Stakeholder alignment

Context Mapping Patterns

Shared Kernel

Common model, limited coupling between contexts

Example:
Shared Customer Data

Customer-Supplier

Downstream context depends on upstream

Example:
Order Service → Inventory

Conformist

Downstream adapts to upstream model

Example:
Legacy System Integration

Anti-Corruption Layer

Protects core from external systems

Example:
SAP Integration Gateway

Open Host Service

Published API for external consumption

Example:
VW Open API Platform

Partnership

Collaborative relationship between teams

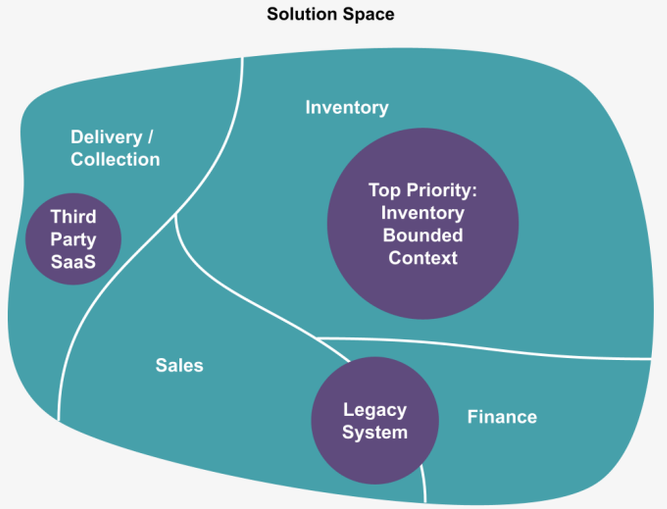
Example:
Joint Development Projects

Separate Ways

Independent systems with minimal coupling

Example:
Mobile App ↔ Web Platform

Context Map Visualization



Key Selection Criteria

- Assess relationship direction and dependency
- Evaluate coupling levels and impact
- Consider team structure and autonomy
- Plan for evolution and change

Pro Tip:
Start simple, evolve patterns as contexts mature and relationships clarify

EventStorming as a Strategic Design Tool

What is EventStorming?

Collaborative workshop format for exploring complex business domains using sticky notes on a wall. Teams discover events, commands, aggregates, and policies through real-time discussion and visualization.

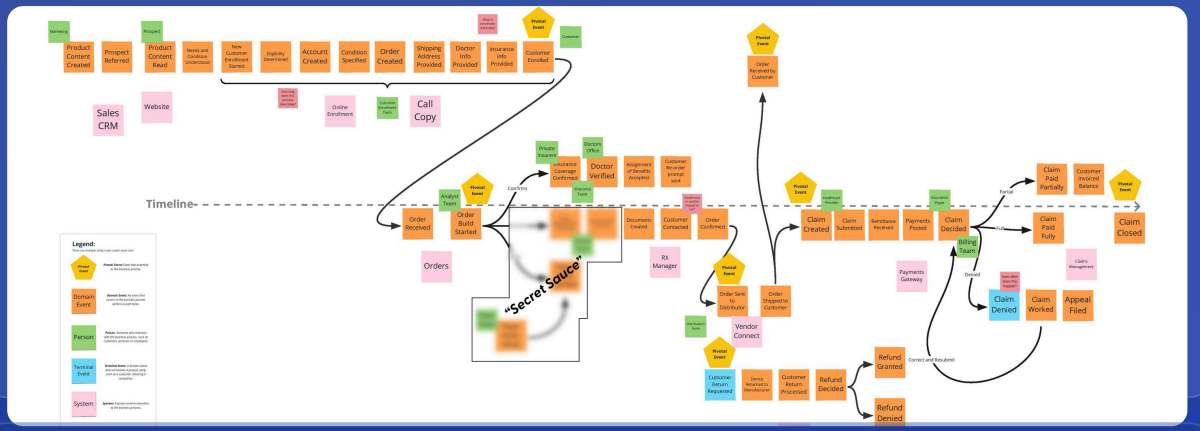
When to Use

- Domain exploration
- Requirements gathering
- Problem space analysis
- Team alignment

Key Elements

- Domain Events
- Commands
- Policies
- Aggregates
- Read Models

EventStorming Example



5-Step Process

1. **Setup**
   Prepare space, gather stakeholders, define scope
2. **Event Mapping**
   Identify domain events chronologically
3. **Command Identification**
   Discover commands that trigger events
4. **Aggregate Definition**
   Group events into aggregates
5. **Policy Discovery**
   Identify policies governing state changes

Automotive Industry Overview: Software Architecture Trends

Software-Defined Vehicles (SDVs)

Key Challenges
100+ million LOC, OTA updates, cybersecurity, autonomous systems

Innovation Impact
60% of car innovation is software-driven

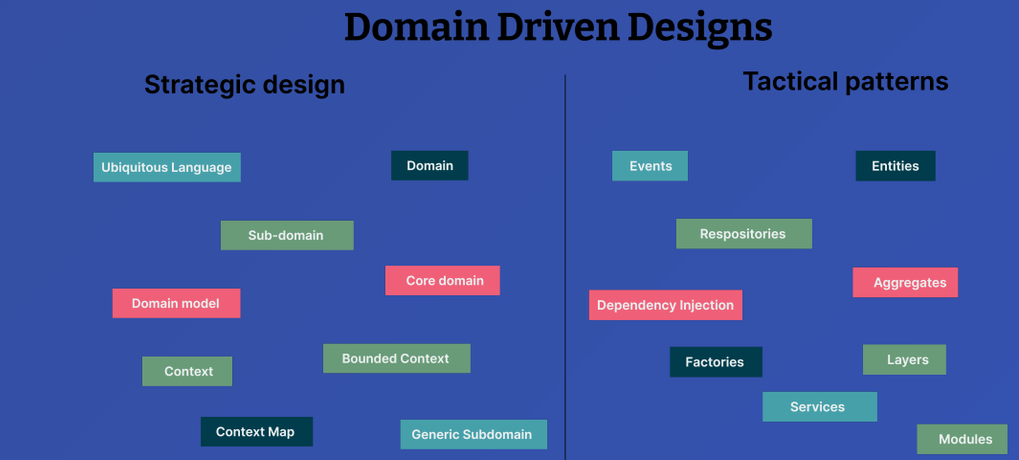
Major Trends

- Centralized Architecture
- Microservices
- Cloud-Native
- DevOps/CI/CD
- API-First Design

Impact Metrics

60%	40%	30%
Software Innovation	Code Reduction	Faster Time-to-Market

Strategic Design Patterns



Industry Insights

- OEMs investing $50B+ annually in software development
- Autonomous driving systems require 500+ sensors and multiple AI models
- VW Group targeting 40% cost reduction with microservices architecture
- Tesla leads with 1000+ software updates per vehicle annually

Competitive Advantage:
Proper DDD implementation can reduce software development time by 30-40%

BMW Case Study: Software Architecture and DDD Implementation

Background & Challenges

Journey
Digital transformation since 2018

Monolith
Legacy monolithic architecture

Coordination
Distributed team alignment

OTA Updates
Continuous deployment needs

DDD Solution

1. Domain-driven design adoption across all teams
2. Bounded Contexts: Vehicle Control, Infotainment, Telematics
3. Microservices: 100+ services with event-driven communication
4. Ubiquitous language defined per bounded context

Key Learnings

- Invest heavily in ubiquitous language development
- Continuous refinement of bounded context boundaries
- Enable team autonomy with clear context ownership

Results & Impact

40%	60%	100+
Faster Development Cycles	Fewer Deployment Failures	Microservices Deployed

- Reduced time-to-market
- Improved team collaboration
- Higher code quality
- Better security posture

AWS Partnership

- Virtual ECUs for development and testing
- Automated OTA update pipelines
- Scalable microservices infrastructure
- Event-driven communication on AWS EventBridge

Reference: AWS BMW Case Study

Technical Highlights

- 3 Core Bounded Contexts
- REST + GraphQL APIs
- Kafka Event Streaming
- CI/CD with GitLab

Toyota Case Study: Software-Defined Vehicle Architecture

Background & Challenges

Scale
80M+ vehicles globally

Regulations
Multi-regional compliance

Legacy
Traditional ECU architecture

Modernization
Connected car initiatives

DDD Solution

1. Strategic partitioning: Powertrain, Infotainment, Safety
2. Bounded context per vehicle function module
3. AUTOSAR compliance integrated with DDD principles
4. Service-oriented architecture with message-driven communication

Key Insights

- Balance AUTOSAR compliance with DDD flexibility
- Invest in cross-functional teams with domain expertise
- Implement shared kernel for common automotive patterns

Results & Impact

50%	35%	80%
ECU Reduction	Faster OTA Updates	Software Reuse

- Reduced hardware costs
- Continuous deployment
- Unified platform
- Enhanced safety

Domain Partitioning

- Powertrain Domain: Engine control, transmission, battery management
- Infotainment Domain: Navigation, multimedia, connectivity
- Safety Domain: ADAS, collision avoidance, emergency systems

Research: MIT Thesis on Legacy OEM Transformation

Technical Stack

- AUTOSAR Adaptive
- SOA + REST APIs
- Message Queues
- CI/CD Pipeline

Ford Case Study: Connected Vehicle Platform

Background & Challenges

Cloud Platform
Vehicle-as-a-Service initiative

Legacy Networks
Dealer integration challenges

Real-time Data
High-speed processing needs

Fleet Scale
10M+ vehicles globally

DDD Solution

1. Bounded Contexts: Vehicle Management, Telematics, Dealer Operations
2. Event-driven microservices for real-time vehicle data
3. Cloud-native architecture on Microsoft Azure
4. API Gateway for external partner integrations

Key Practices

- Event-driven architecture for real-time processing
- API-first design for seamless partner integration
- Cloud-native approach for scalability and elasticity

Results & Impact

45%	70%	10M+
Faster Issue Resolution	Fleet Efficiency	Scalable Vehicles

- Reduced downtime
- Predictive maintenance
- Partner ecosystem
- Continuous deployment

Domain Architecture

- Vehicle Management: Fleet tracking, health monitoring, remote diagnostics
- Telematics: Data streaming, analytics, driver behavior insights
- Dealer Operations: Inventory, service scheduling, customer support

Research: Connected Vehicle Architecture Comparison

Technical Stack

- Microsoft Azure
- REST + GraphQL
- Azure Event Hubs
- Azure Functions

Mercedes-Benz Case Study: Software Architecture Transformation

Background & Challenges

MB.OS Initiative
CASE strategy transformation

ECU Complexity
60+ electronic control units

Legacy Systems
Infotainment modernization

ADAS Integration
Autonomous driving complexity

DDD Solution

1. Bounded Contexts: Navigation, Entertainment, Safety, Connectivity
2. Centralized architecture with 3 core domains
3. Microservices: 150+ services with domain ownership
4. Hybrid cloud strategy: Azure + on-prem

Key Successes

- Strong executive sponsorship from board level
- Gradual migration approach with phased rollout
- Focus on developer experience and productivity

Results & Impact

55%	40%	150+
Faster Feature Delivery	Integration Cost Reduction	Microservices

- OTA update capability
- Reduced technical debt
- Improved customer experience
- Enhanced innovation speed

Domain Architecture

- Navigation: Route planning, traffic data, GPS positioning
- Entertainment: Multimedia, streaming, in-car services
- Safety: ADAS, collision avoidance, emergency braking
- Connectivity: V2X, 5G, remote diagnostics

Research: AI Integration Case Study

Technical Stack

- Azure + On-Prem
- REST + gRPC
- Azure Service Bus
- Kubernetes

Tools and Platforms for DDD Strategic Design

EventStorming

- Miro
- Mural
- Stormy
- Domo.robo.to

Context Mapping

- ★ ContextMapper
- Context Canvas
- ★ Bounded Context Canvas

Domain Modeling

- ★ DDD-Toolbox
- Qlerify
- Archi

Integration & API

- Swagger
- AsyncAPI
- OpenAPI

Open Source Tools

- **ContextMapper:** DSL for strategic DDD and context mapping
- **DDD-Toolbox:** Modern web app for DDD practitioners
- **Bounded Context Canvas:** Collaborative tool for designing contexts

Tool Comparison Matrix

Tool	Free Tier	Collaborative	OS
Miro	✓	✓	✗
ContextMapper	✓	–	✓
DDD-Toolbox	✓	✓	✓
Qlerify	✓	✓	✗
Bounded Context Canvas	✓	✓	✓

Collaboration

- Slack
- Teams
- Discord

Documentation

- Notion
- Confluence
- GitBook

Context Mapper Tool - Detailed Guide

Overview

- DSL and tools for strategic DDD and context mapping
- Open-source project with active community support
- Supports Bounded Context, CML, and multiple exports

Key Features

- CML DSL Language
- Context Mapping
- BC Design
- ACL Generation
- PlantUML Export
- Mermaid Export

Getting Started

1. Install CLI or use web app at contextmapper.org
2. Define Bounded Contexts using CML syntax
3. Specify relationships (Customer-Supplier, etc.)
4. Generate artifacts: diagrams, code, documentation

CML Example

```
BoundedContext VehicleControl {
  type = CORE_DOMAIN
}
BoundedContext Infotainment {
  type = SUPPORTING
}
CustomerSupplier(VehicleControl -> Infotainment)
```

Build Pipeline Integration

Maven Integration

Add context-maven-plugin to pom.xml

```
<plugin>
  <groupId>org.contextmapper</groupId>
  <artifactId>context-maven-plugin</artifactId>
</plugin>
```

Gradle Integration

Apply context-gradle plugin

```
plugins {
  id 'org.contextmapper' version 'latest'
```

Key Benefits

- Version Control Support
- Automated Documentation
- Multiple Export Formats
- IDE Integration

Resources & References

- contextmapper.org - Official Website
- Bounded Context Canvas - GitHub
- Free DDD Learning Resources

Practical Exercise 1: Subdomain Classification Workshop

Exercise Objective

Practice identifying and classifying subdomains using real-world scenario from automotive industry

Task List

1. List all business capabilities
2. Group into subdomains
3. Classify as Core/Supporting/Generic
4. Justify your classification

Scenario

- Automotive e-commerce platform
- Vehicle sales and inventory management
- Customer journey: browsing, purchasing, delivery

Sample Solution

- **Vehicle Inventory** - Core
- **Order Management** - Supporting
- **Payment Processing** - Generic
- **User Management** - Generic
- **Vehicle Recommendations** - Core
- **Customer Support** - Supporting

Time Allocation

60
Minutes

Materials

- Miro board
- Classification template

Discussion Questions

- Why is this core subdomain?
- Could this be a generic subdomain?
- What competitive advantage does it provide?

Practical Exercise 2: Bounded Context Mapping

Exercise Objective

Define bounded contexts and their relationships using connected vehicle platform scenario

Scenario (Ford Case Study)

- Connected vehicle platform
- Real-time vehicle data processing
- Dealer network integration
- Scalable to 10M+ vehicles

Task List

1. Identify bounded contexts
2. Define ubiquitous language
3. Choose mapping patterns
4. Draw context map

Duration

90

Minutes

Tools

- ContextMapper
- Miro
- BC Canvas

Sample Bounded Contexts

- **Vehicle Management** - Core
- **Telematics** - Supporting
- **Dealer Operations** - Supporting
- **Payment Processing** - Generic
- **User Management** - Generic

Context Mapping Patterns

- **Vehicle Management → Telematics**
  Customer-Supplier Pattern
- **Vehicle Management → Dealer Operations**
  Partnership Pattern
- **All Contexts → Payment Processing**
  Customer-Supplier Pattern

Key Considerations

- Define ubiquitous language per context
- Choose appropriate pattern for each relationship
- Consider data ownership and team boundaries

Mini-Project: Automotive Domain Strategic Design

Project Overview

End-to-end DDD strategic design for an automotive e-commerce platform

Team & Effort

Team Size
4-6

Duration
2-3 wks

Deliverables

- Context map document
- CML files
- Presentation deck

Tools & Tech Stack

- ContextMapper
- Miro (EventStorming)
- Git (Version Control)

Project Phases

1 Domain Discovery

Days 1-2

- ✓ EventStorming workshop
- ✓ Stakeholder interviews
- ✓ Business process mapping

2 Subdomain Classification

Days 3-4

- ✓ Identify core/supporting/generic
- ✓ Create classification matrix
- ✓ Define investment priorities

3 Bounded Context Design

Days 5-6

- ✓ Define bounded contexts
- ✓ Ubiquitous language
- ✓ Context map creation

4 Integration Planning

Days 7-8

- ✓ Choose mapping patterns
- ✓ Define API contracts
- ✓ Plan implementation

Common Patterns and Anti-Patterns in Strategic Design

Common Patterns

1. Shared Kernel

Common model across contexts with limited coupling

⚠ Careful with scope - can become anti-pattern if too large

2. Anti-Corruption Layer

Protects core from external system influences

⚠ Adds complexity - use when integration risk is high

3. Open Host Service

Published APIs for external consumption

⚠ Requires robust version management and documentation

4. Conformist

Downstream adapts to upstream model

⚠ Reduces autonomy - use when upstream is stable and aligned

Common Anti-Patterns

1. Distributed Monolith

Distributed architecture but tightly coupled

❗ No team autonomy, single point of failure, deployment issues

Solution: Strong bounded contexts, autonomous teams, clear ownership

2. Big Ball of Mud

No clear boundaries, everything depends on everything

❗ Unmaintainable, fragile, slow development cycles

Solution: Identify contexts, establish clear boundaries, refactor gradually

3. Microservice Hell

Too many services creating orchestration complexity

❗ Operational overhead, distributed transactions, debugging nightmares

Solution: Coarse-grained services, domain events, embrace eventual consistency

4. Generic Over-engineering

Investing heavily in commodity generic subdomains

❗ Wasted resources, opportunity cost, slow innovation

Solution: Use off-the-shelf solutions, focus on core domains

Best Practices for Cutting and Distinguishing Models

Strategic Thinking

1. **Problem Space First**
   Start with business challenges, not technology constraints
2. **Domain Experts**
   Involve business stakeholders early and continuously
3. **Business Value**
   Focus on competitive advantage, not technical elegance
4. **Iterative Refinement**
   Embrace continuous evolution of boundaries

Priority: MUST-HAVE
Foundation for all DDD initiatives

Implementation

1. **EventStorming Workshops**
   Use collaborative discovery for domain exploration
2. **Ubiquitous Language**
   Document and maintain consistent terminology per context
3. **Focused Cohesion**
   Keep bounded contexts cohesive and single-purpose
4. **Plan Evolution**
   Design for inevitable changes and growth

Priority: SHOULD-HAVE
Critical for successful execution

Team Considerations

1. **Team-Context Alignment**
   Align bounded contexts with team boundaries
2. **Team Autonomy**
   Enable independent development and deployment
3. **Knowledge Sharing**
   Share learnings and patterns across contexts
4. **Regular Retrospectives**
   Continuous improvement of boundaries and practices

Priority: NICE-TO-HAVE
Enhances team effectiveness and morale

Metrics and Evaluation

Evaluation Categories

Domain-Level Metrics

- Subdomain Clarity
- Bounded Context Cohesion
- Cross-Context Coupling
- Team Autonomy

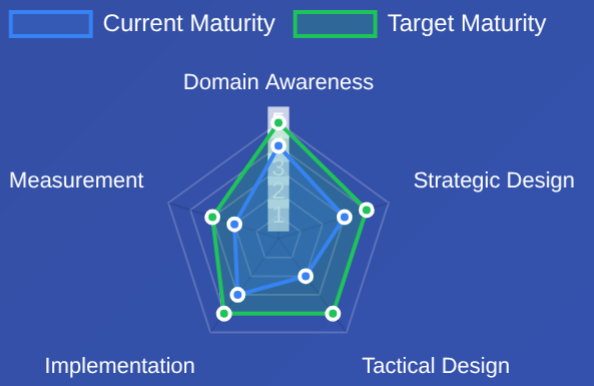
Team Metrics

- Cycle Time
- Lead Time
- Deployment Frequency
- Change Failure Rate

Business Metrics

- Time-to-Market
- Feature Delivery Speed
- Cost of Change
- Customer Satisfaction

DDD Maturity Model



Target Metrics

Cross-Context Coupling	< 20%
Team Autonomy	> 80%
Cycle Time	< 2 weeks
Subdomain Clarity	> 85%

Challenges and Solutions

Common Pitfalls

1. Analysis Paralysis
Overthinking, excessive theory, slow decision-making

2. Team Resistance
Fear of change, ownership concerns, pushback

3. Legacy Integration
Tight coupling, monolith dependencies, technical debt

4. Context Creep
Blurred boundaries, context expansion over time

5. Testing Complexity
Complex dependencies, end-to-end testing challenges

Practical Solutions

Time-Boxed Workshops
Focus on MVP, iterate frequently, limit analysis phase

Team Involvement
Involve teams early, demonstrate benefits, provide training

Anti-Corruption Layer
Strangler pattern, gradual migration, protect core domain

Regular Reviews
Context guardians, automated validation, continuous alignment

Contract Testing
Consumer-driven contracts, mock services, integration testing

Connecting to Tactical Design

Bridge Between Layers

Strategic Design

WHAT - Domain Modeling, Bounded Contexts, Context Mapping

↓

Tactical Design

HOW - Aggregates, Entities, Value Objects, Repositories

Key Transitions

- → Bounded Contexts → Aggregates
- → Ubiquitous Language → Domain Models
- → Context Maps → Integration Patterns
- → Subdomain Classification → Tactical Patterns

Practical Guidance

1. **Start with Strategic**
   Begin by understanding the domain and identifying bounded contexts
2. **Validate with Tactical**
   Apply tactical patterns to implement strategic decisions
3. **Iterate Continuously**
   Refine boundaries and models based on feedback and learning

Remember

Strategic design provides the "what" and "why", while tactical design provides the "how". Both layers are essential for successful DDD implementation.

Recap of Learning Goals

Learning Goals Checklist

- Identify symptoms of monolithic or distributed monolith systems
- Evaluate cross-team model coherence
- Apply strategic thinking to cut domains
- Design problem/solution space separation
- Classify subdomains (core, supporting, generic)
- Create bounded contexts
- Map context relationships

✓ 100% Coverage Achieved

Self-Assessment

Understanding Rating 4.5/5

Key Strengths

Strategic Thinking | Domain Analysis | Context Mapping

Next Steps

- → Review key concepts and case studies
- → Practice with mini-project exercises
- → Complete self-study resources
- → Prepare for Day 6: Context Mapping

Action Items: Reflect on automotive case studies, experiment with ContextMapper tool, practice subdomain classification

Self-Study Resources

Recommended Reading

Domain-Driven Design
by Eric Evans

Implementing Domain-Driven Design
by Vaughn Vernon

Patterns, Principles, and Practices of DDD
by Scott Millett

Online Articles

- Nick Tune's Strategic DDD articles
- Microsoft Learn DDD documentation
- ThoughtWorks DDD guides

Video Learning

- DDD Crew YouTube channel
- Context Mapper tutorials
- EventStorming workshops

Community & Support

- Reddit r/DomainDrivenDesign
- StackOverflow DDD tag
- DDD Slack community

Connection to Day 6: Context Mapping

Day 6 Preview

Strategic Design 2: Context Mapping

Deep dive into context mapping patterns and integration strategies

Key Topics

- Detailed context mapping patterns
- Integration strategies
- Relationship management
- Evolution strategies
- Conflict resolution

Preparation Checklist

- [x] Review today's context mapping section
- [x] Practice with ContextMapper tool
- [x] Read automotive context mapping case study
- [x] Prepare questions for Day 6

Bridge to Next Day

Context mapping builds directly on strategic design foundations established today.

Recommended Preparation

2 hours of focused review and practice

Additional Resources

Expert Insights

Nick Tune

Strategic DDD blog, microservices and DDD patterns

Martin Fowler

DDD patterns, practices, and architectural guidance

Alberto Brandolini

EventStorming creator, collaborative domain exploration

Eric Evans

DDD founder, strategic design pioneer

Case Studies

- BMW microservices transformation
- Toyota connected vehicle architecture
- Ford cloud platform integration
- Mercedes software architecture transformation

Research & Discussions

- MIT automotive software transformation research
- IEEE autonomous vehicle architecture papers
- Connected vehicle architecture comparison study
- Reddit r/DomainDrivenDesign community
- StackOverflow strategic design discussions

Summary

Key Takeaways

1. Strategic Design 1 is about cutting and distinguishing models
2. Problem space ≠ Solution space
3. Subdomain classification guides investment decisions
4. Bounded contexts define model boundaries
5. Context mapping relationships are crucial
6. EventStorming is a powerful exploration tool
7. DDD is being adopted by automotive leaders

Remember

- Start with problem space
- Involve domain experts
- Iterate continuously
- Embrace team autonomy

Progress Tracker

- ✓ Day 5: Strategic Design 1 Complete
- → Day 6: Context Mapping (Next)

Preparation: 2 hours review recommended

Next Steps

Immediate Actions

1. **Review all Day 5 slides**
   Revisit concepts, case studies, and exercises
2. **Complete mini-project**
   Apply DDD to automotive domain scenario
3. **Practice with ContextMapper**
   Create bounded contexts and relationships
4. **Prepare for Day 6**
   Draft questions on context mapping

Timeline: Within 24 hours
Estimated time: 2-3 hours total

Short-term (1-2 weeks)

- Apply DDD to current project
- Organize EventStorming workshop
- Create context map with team
- Share learnings with team

Long-term (1-3 months)

- Attend DDD conference or meetup
- Write and share a case study
- Mentor colleagues on DDD
- Contribute to open-source tools

Resource Pack Index

Quick Reference

Tools
ContextMapper, DDD-Toolbox, Miro, Context Canvas

Case Studies
BMW, Toyota, Ford, Mercedes-Benz

Exercises
Subdomain Classification, Bounded Context Mapping, Mini-Project

Resources
Books, Articles, Videos, Communities

Download Links

- GitHub repo for templates
- ContextMapper official site
- DDD Crew resources
- Miro community templates

Full Resource Directory

Strategic Tools
EventStorming, ContextMapper

Modeling Tools
DDD-Toolbox, Context Canvas

Documentation
Books, Articles, Papers

Learning
Videos, Tutorials, Courses

Support Channels

Email Questions
ddd-training@example.com

Slack Community
#ddd-training workspace

Twitter Updates
@DDDTraining

Thank You & Questions

Thank You!

Thank you for participating in Day 5 of the iSAQB-DDD training!

We've covered the fundamentals of Strategic Design 1: Cutting and Distinguishing Models.

Remember

DDD is not about technology - it's about understanding and modeling the domain.

Contact Information

Training Coordinator
ddd-training@isaqb.org

Resources Website
www.isaqb.org/training/ddd

Next Session
Day 6: Context Mapping

Q&A Session

- Open floor for questions
- Discussion on mini-project
- Sharing of insights