The CPSA® Advanced Level Module DDD

Day 1: Foundations - Domain, Model & Ubiquitous Language

iSAQB® Training Course in Domain-Driven Design

22 December 2025

VOLKSWAGEN GROUP INDIA

Overview of Day 1 Learning Goals

LG 1-1

Domain Connections

Explain connections between **domains** , **software** , and **models**

LG 1-2

Ubiquitous Language

Understand role of **ubiquitous language** in domain modeling

LG 1-3

DDD Building Blocks

Explain DDD building blocks (**Entities** , **Value Objects** , **Aggregates**)

LG 1-4

Block Connections

Explain connections between building blocks

LG 1-1: Building Intuition

Communication Breakdown

When Teams Don't Understand Each Other

- ✕ **Misinterpretation** of requirements
- ✕ **Delays** due to clarification cycles
- ✕ **Technical debt** from poor design decisions
- ✕ **Frustration** between business and technical teams

Automotive Impact

Real-World Consequences

- ! **Safety issues** from misunderstood requirements
- ! **Costly recalls** due to software defects
- ! **Delayed releases** for new vehicle features

> "The most disastrous thing that can happen to a software project is to have the wrong people making the key decisions."
>
> *- Eric Evans, Author of Domain-Driven Design*

LG 1-1: Context - Automotive Software Complexity

Software Complexity Metrics

100M+

Lines of Code

150+

ECUs in Premium Vehicles

Multiple Domains

- **Powertrain** - Engine, transmission, battery management
- **Infotainment** - Media, navigation, connectivity
- **Safety** - ADAS, airbags, collision avoidance
- **Connectivity** - OTA updates, V2X communication

Domain Distribution

Software Code Distribution Across Domains

Infotainment	25%
Powertrain	20%
Safety Systems	18%
Connectivity	15%
Body Control	12%
Other Systems	10%

Cross-Team Challenges

- **Integration complexity** between domains
- **Communication barriers** between specialized teams

LG 1-1: Purpose - Why Understanding Connections Matters

Key Benefits

Software Quality

- ✓ **Accurate requirements** translation
- ✓ **Reduced rework** and maintenance costs
- ✓ **Better testability** through clear domain boundaries

Development Efficiency

- ✓ **Faster decision making** with shared understanding
- ✓ **Reduced ambiguity** in requirements

Automotive Impact

System Integration

- ✓ **Clear interfaces** between vehicle subsystems
- ✓ **Reduced integration complexity** between domains
- ✓ **Simplified maintenance** of vehicle systems

> "When the domain model is precise and well-understood by both developers and domain experts, the software becomes an extension of the business thinking."
>
> *- Eric Evans, Author of Domain-Driven Design*

LG 1-1: Key Terminologies - Domain, Model, Software

Domain

Sphere of **knowledge**, **activity**, or **influence** in which software operates

Automotive Examples

- **Powertrain** - Engine, transmission, battery management
- **Infotainment** - Media, navigation, connectivity
- **Safety** - ADAS, airbags, collision avoidance

Model

Abstraction of domain concepts that captures essential structure and behavior

Model Examples

- **Entity Model** - Vehicle with unique VIN and attributes
- **Value Object** - Speed, Temperature, Coordinates
- **Aggregate** - Vehicle with related components

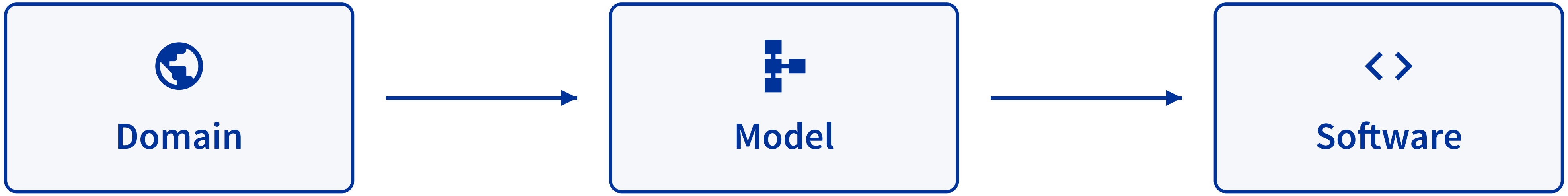
Software

Implementation that serves the domain and reflects the model

Software Examples

- **Powertrain Control Module** - Manages engine operations
- **Infotainment System** - Handles media and navigation
- **Safety Controller** - Processes sensor data

LG 1-1: Concepts - How Domains Relate to Models and Software



Domain

Model

Software

→ Domain → Model

- **Abstraction** of domain concepts
- **Simplification** of complex reality
- **Focus** on essential aspects

→ Model → Software

- **Implementation** of model concepts
- **Translation** to executable code
- **Structure** following model design

Continuous Alignment

- **Feedback** from software refines model
- **Iteration** improves understanding
- **Validation** against domain reality

LG 1-1: Walkthrough - From Business Need to Software Implementation

1

Business Need

- Identify **domain problem**
- Consult **domain experts**
- Define **requirements**

2

Domain Modeling

- Extract **key concepts**
- Create **ubiquitous language**
- Design **domain model**

3

Implementation

- Translate **model to code**
- Apply **design patterns**
- Implement **business logic**

4

Validation

- **Test** against requirements
- **Refine** model as needed
- **Deploy** and monitor

Automotive Example: EV Battery Management

Business Need: **Optimize battery life** → Model: **Battery, ChargeLevel, Temperature** → Implementation: **BatteryManagementService** → Validation: **Range testing**

LG 1-1: Analogies - Blueprints and Maps

Building Blueprint

- **Real-world building** - Physical structure with purpose
- **Blueprint** - Abstract representation of building design
- **Construction** - Physical implementation following blueprint

DDD Connection

Building	DDD
Real-world building	**Domain**
Blueprint	**Model**
Construction	**Software**

Navigation Map

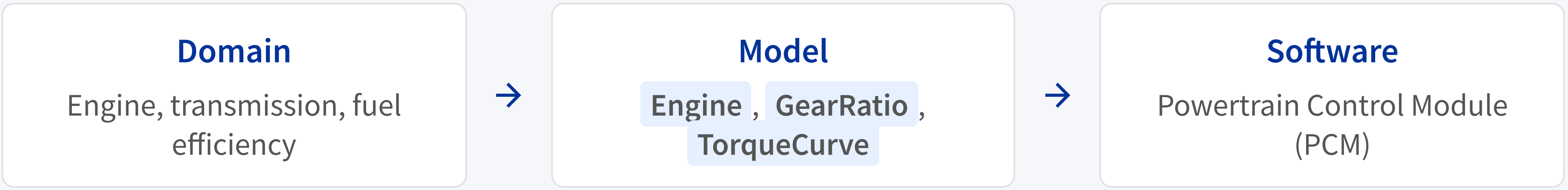
- **Physical territory** - Actual landscape with roads
- **Map** - Simplified representation of territory
- **GPS navigation** - Digital implementation of map

DDD Connection

Navigation	DDD
Physical territory	**Domain**
Map	**Model**
GPS navigation	**Software**

LG 1-1: Connected Examples - Vehicle Control Systems

Powertrain Control System



Domain

Engine, transmission, fuel efficiency

→

Model

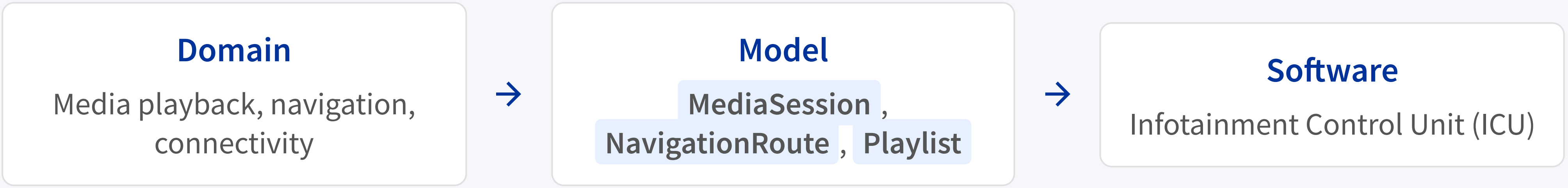
`Engine`, `GearRatio`, `TorqueCurve`

→

Software

Powertrain Control Module (PCM)

Infotainment System



Domain

Media playback, navigation, connectivity

→

Model

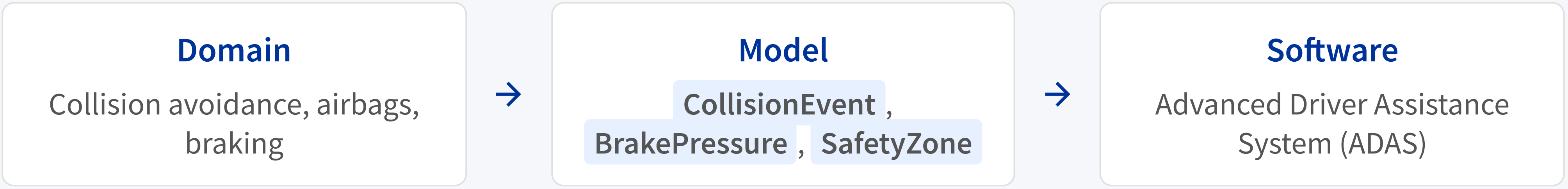
`MediaSession`, `NavigationRoute`, `Playlist`

→

Software

Infotainment Control Unit (ICU)

Safety Systems



Domain

Collision avoidance, airbags, braking

→

Model

`CollisionEvent`, `BrakePressure`, `SafetyZone`

→

Software

Advanced Driver Assistance System (ADAS)

LG 1-1: Case Study - Tesla's Software-Defined Vehicles

Tesla's Approach

Domain-First Strategy

- **Vehicle as computer** on wheels
- **OTA updates** for continuous improvement
- **Centralized architecture** with clear domains

Key Domains

- **Autopilot** - Autonomous driving capabilities
- **Battery Management** - Range optimization
- **Safety Systems** - Collision avoidance

Model-to-Software Connection

Domain Models

- **Vehicle** entity with unique ID and state
- **Value Objects** for speed, battery level, temperature
- **Aggregates** for battery pack, drivetrain

Software Implementation

- **Central computers** running domain-specific software
- **Continuous alignment** between model and code

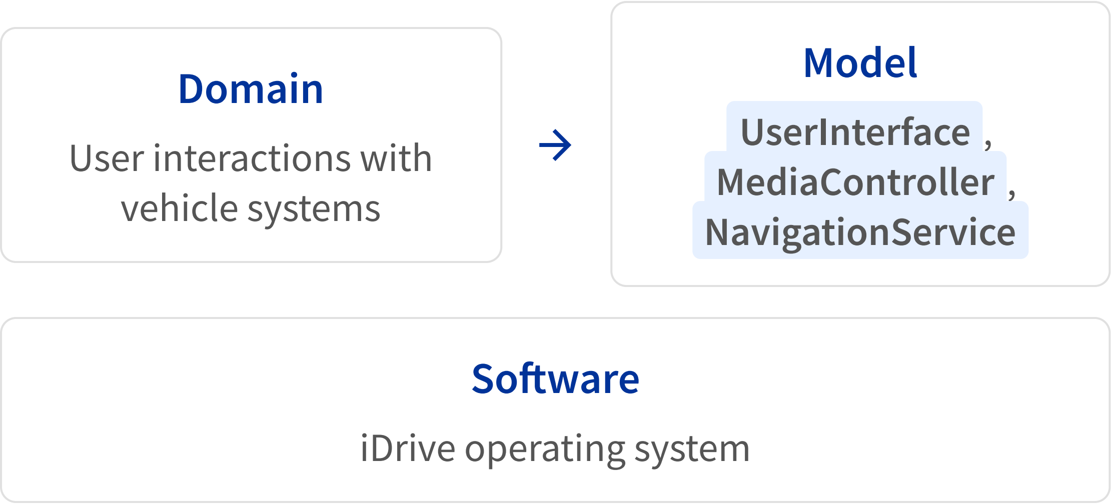
> "Tesla's success comes from treating the vehicle as a software platform first, with hardware as the enabler."
>
> *- Industry Analysis*

LG 1-1: Reverse Engineering - BMW's iDrive System

iDrive System Analysis

Domain Identification

- **Infotainment** - Media playback, navigation
- **Connectivity** - Phone integration, OTA updates
- **User Interface** - Touch, voice, gesture controls



Domain

User interactions with vehicle systems

→

Model

UserInterface, MediaController, NavigationService

Software

iDrive operating system

DDD Implementation

Model Elements

- **UserSession** - Tracks current user and preferences
- **Coordinates**, **MediaMetadata** - Value objects
- **NavigationAggregate** - Route with waypoints

Integration Patterns

- **Anticorruption Layer** between vehicle systems
- **Bounded Contexts** for different vehicle functions

BMW's iDrive demonstrates clear separation of concerns with distinct domains for infotainment, connectivity, and vehicle controls

LG 1-1: Brainstorming Puzzles - Identifying Domains

1 Electric Vehicle Charging Network

A system that manages EV charging stations, handles payments, tracks vehicle battery levels, and provides navigation to charging points.

Identify the domains

Charging · Payment · Battery · Navigation

2 Fleet Management System

A platform for managing a car-sharing service with vehicle tracking, driver assignment, maintenance scheduling, and customer booking.

Identify the domains

Fleet · Driver · Maintenance · Booking · Tracking

3 Connected Car Platform

A system that provides over-the-air updates, collects vehicle telemetry, enables remote features, and integrates with smart home devices.

Identify the domains

OTA Updates · Telemetry · Remote Control · Smart Home

LG 1-1: Scenarios and Solutions - When to Prioritize Domain Understanding

Complex Business Rules

Systems with intricate business logic that directly impacts core functionality

Approach

- ✓ **Deep domain modeling** before implementation
- ✓ **Collaborative workshops** with domain experts
- ✓ **Iterative refinement** of domain model

Cross-Team Collaboration

Multiple teams working on interconnected vehicle subsystems

Approach

- ✓ **Clear domain boundaries** between teams
- ✓ **Shared vocabulary** across teams
- ✓ **Integration patterns** between domains

Evolving Requirements

Systems with frequent changes to business rules and functionality

Approach

- ✓ **Flexible domain model** that adapts to change
- ✓ **Strategic design** to isolate volatile areas
- ✓ **Continuous alignment** with domain experts

LG 1-1: Self-Study Resources

Books

Domain-Driven Design
Eric Evans - **Foundational text** on DDD principles

Implementing DDD
Vaughn Vernon - **Practical guide** with code examples

Patterns, Principles
Martin Fowler - **Analysis patterns** for domain modeling

Articles

Domain Modeling Made Functional
Scott Wlaschin - **Functional approach** to domain modeling

Strategic DDD
Alberto Brandolini - **Bounded contexts** and context mapping

Modeling in DDD
Udi Dahan - **Domain events** and message-based systems

Online Resources

DDD Community
Active community with **forums, events** and resources

DDD-Crew
Free resources, patterns, and **practical examples**

EventStorming
Tools and techniques for **collaborative modeling**

Video Courses

DDD Fundamentals
Julie Lerman - **Entity Framework** with DDD

Advanced DDD
Vladimir Khorikov - **Practical patterns** and refactoring

DDD in Practice
Jimmy Bogard - **Real-world applications** and patterns

LG 1-2: Introduction - Building Intuition

⚠ Communication Challenges

Lost in Translation

- ✕ **Different vocabularies** between teams
- ✕ **Misinterpreted requirements** due to terminology
- ✕ **Implementation gaps** from misunderstood concepts
- ✕ **Endless clarification** cycles

Automotive Impact

Real-World Consequences

- ❗ **Feature delays** from misunderstood requirements
- ❗ **Integration failures** between vehicle systems
- ❗ **User experience issues** from inconsistent terminology

> "The most important thing about communication is hearing what isn't said."
>
> *- Peter Drucker*

LG 1-2: Context - Communication Challenges in Software Development

Communication Barriers

Terminology Gaps

- **Different terms** for same concept
- **Technical jargon** vs. business language
- **Implicit assumptions** about terminology

Team Silos

- **Specialized vocabularies** within teams
- **Lack of shared context** between domains
- **Inconsistent terminology** across projects

Automotive Impact

Development Consequences

- **Feature delays** from misunderstood requirements
- **Integration failures** between vehicle systems
- **Inconsistent user experience** across interfaces

> "The same feature implemented by different teams often uses different terminology, creating confusion for both developers and users."

LG 1-2: Purpose - Why Ubiquitous Language Matters

Key Benefits

Shared Understanding

- ✓ **Common vocabulary** across teams
- ✓ **Precise communication** between experts and developers
- ✓ **Reduced ambiguity** in requirements

Implementation Benefits

- ✓ **Consistent naming** in code and documentation
- ✓ **Better domain modeling** through precise terminology
- ✓ **Simplified maintenance** with clear concepts

Automotive Impact

System Integration

- ✓ **Clear interfaces** between vehicle subsystems
- ✓ **Consistent terminology** across vehicle systems
- ✓ **Simplified OTA updates** with shared vocabulary

> **"When a team shares a common language, the software becomes an extension of the domain experts' thinking."**
>
> *- Eric Evans, Author of Domain-Driven Design*

LG 1-2: Key Terminologies - Ubiquitous Language, Bounded Context

Ubiquitous Language

Shared vocabulary between domain experts and developers

Automotive Examples

- **Powertrain** - Torque, GearRatio, ThrottlePosition
- **Infotainment** - Playlist, MediaSource, NavigationRoute

Bounded Context

Explicit boundary where a specific domain model applies

Context Examples

- **Safety Context** - Different meaning for "Brake" vs. "Parking"
- **Powertrain Context** - Specific terminology for engine components

LG 1-2: Concepts - Creating and Evolving Ubiquitous Language

Evolution Process

Iterative Refinement

- **Gather terminology** from domain experts
- **Refine meanings** through discussion
- **Model concepts** with precise language
- **Apply consistently** in code and documentation

Best Practices

Language Development

- Collaborative Creation
- Document Glossary
- Consistent Usage
- Regular Refinement

LG 1-2: Walkthrough - Developing a Shared Vocabulary

1

Gather Terms

- **Interview** domain experts
- **Document** existing terminology
- **Identify** key concepts

2

Refine Meanings

- **Discuss** with stakeholders
- **Resolve** ambiguous terms
- **Consolidate** similar concepts

3

Model with Language

- **Apply terms** to domain model
- **Use consistently** in implementation
- **Create glossary** for reference

Automotive Example: Powertrain Domain

Gather: Engine terms from mechanics → **Refine:** Define "Torque" precisely → **Model:** Use "TorqueCurve" in code → **Document:** Add to glossary

LG 1-2: Analogies - Common Language Examples

Medical Field

- ✓ **Precise terminology** for body parts, symptoms, treatments
- ✓ **Universal understanding** between doctors, nurses, technicians
- ✓ **Reduced errors** from miscommunication

Legal Field

- ✓ **Specific language** for contracts, clauses, precedents
- ✓ **Shared vocabulary** across legal teams
- ✓ **Clear documentation** with precise terms

Scientific Research

- ✓ **Standardized terminology** for methods, results, conclusions
- ✓ **Peer review** with common language
- ✓ **Knowledge transfer** through precise communication

DDD Connection

Just as specialized fields develop precise terminology, **ubiquitous language** creates shared understanding between domain experts and developers in software

LG 1-2: Connected Examples - Automotive Terminology

Powertrain

Torque

Rotational force, not "power" or "strength"

GearRatio

Precise ratio, not "gear setting"

ThrottlePosition

Position value, not "acceleration"

Infotainment

MediaSource

Source type, not "player" or "format"

Playlist

Ordered collection, not "song list"

NavigationRoute

Complete path, not "directions" or "map"

Safety Systems

SafetyZone

Defined area, not "safe space"

AlertLevel

Severity level, not "warning" or "alarm"

CollisionEvent

Specific occurrence, not "crash" or "impact"

LG 1-2: Case Study - Mercedes-Benz's MBUX System

MBUX Approach

Consistent Language

- **Unified terminology** across all interfaces
- **User-centric vocabulary** for all features
- **Shared understanding** between UX and engineering

Key Language Elements

- **"Hey Mercedes"** - Voice activation
- **"Comfort Mode"** - Unified setting concept
- **"MBUX Interior Assist"** - Gesture control

DDD Implementation

Model Elements

- **UserSession** - Tracks user preferences and state
- **MediaMetadata**, **NavigationPoint** - Value objects
- **InfotainmentAggregate** - Manages media and navigation

Integration Benefits

- **Clear boundaries** between vehicle systems
- **Consistent terminology** across vehicle functions

> "MBUX demonstrates how a consistent, user-centric language creates a unified experience across complex vehicle systems."
>
> *- Automotive UX Analysis*

LG 1-2: Reverse Engineering - Audi's MMI Interface

MMI System Analysis

Language Consistency

- **Media Control** - Consistent terminology across audio sources
- **Navigation** - Standardized route and destination terms
- **Connectivity** - Unified phone integration vocabulary

DDD Implementation

Model Elements

- **UserInterfaceSession** - Tracks interaction state
- **MediaMetadata**, **Coordinates** - Value objects
- **InfotainmentAggregate** - Manages connected features

Integration Patterns

- **Bounded Contexts** for different MMI modules
- **Consistent terminology** across MMI interfaces

Audi's MMI demonstrates clear separation of concerns with consistent terminology across infotainment, navigation, and vehicle controls

LG 1-2: Brainstorming Puzzles - Creating Ubiquitous Language

1 EV Charging System

A system with multiple teams working on charging, payment, and battery management. Each team uses different terms for "charging session."

Create Ubiquitous Language

ChargingSession | PaymentTransaction | BatteryState

2 Vehicle Safety Features

Safety engineers use "brake assist" while UX designers use "emergency stop" for the same feature.

Resolve Terminology

AutomaticEmergencyBraking | CollisionAvoidance | ProximityDetection

3 Infotainment Controls

Users interact with media, navigation, and climate control through different interfaces with inconsistent terminology.

Unify Interface Language

UserInteraction | ControlMode | SystemState

LG 1-2: Scenarios and Solutions - When Terminology Conflicts Arise

Cross-Team Conflicts

Different teams using inconsistent terminology for same concepts

Solution Approach

- **Glossary creation** with clear definitions
- **Regular alignment** meetings between teams
- **Bounded contexts** f different domains

Evolving Requirements

Business terminology changing during development

Solu Appr

- **Versi gloss** chang
- **Livin docu** that e langu
- **Dom work** termi refine

Technical vs. Business

Developers using technical terms that business users don't understand

Solution Approach

- **Translation layer** betwee technical and business term
- **User-facing documentatic** with business terminology
- **Code comments** mapping technical to business term

LG 1-2: Self-Study Resources

Books

- **Domain-Driven Design**
  Eric Evans - **Ubiquitous Language** chapter
- **Implementing DDD**
  Vaughn Vernon - **Bounded Contexts** examples
- **DDD Distilled**
  Jimmy Nilsson - **Strategic Design** and language

Articles

- **Ubiquitous Language in Practice**
  Alberto Brandolini - **Practical examples** and patterns
- **Strategic Domain-Driven Design**
  Eric Evans - **Language patterns** and context mapping
- **EventStorming Guide**
  DDD-Crew - **Collaborative modeling** techniques

Online Resources

- **DDD Community**
  Forums, discussions, and **language examples**
- **Glossary Tools**
  Software for **creating and managing** ubiquitous language
- **Workshop Templates**
  Guides for **language creation** workshops

LG 1-3 & 1-4: Introduction - Building Intuition

Why Building Blocks Matter

Software Challenges

- ✕ **Poor structure** leads to maintenance issues
- ✕ **Inconsistent design** increases complexity
- ✕ **Weak boundaries** cause integration problems

Automotive Impact

Real-World Consequences

- ! **Feature delays** from unclear boundaries
- ! **Integration failures** between vehicle systems
- ! **Safety issues** from inconsistent design

> "Building blocks provide the vocabulary and structure for expressing domain concepts in software."
>
> *- Eric Evans, Author of Domain-Driven Design*

LG 1-3 & 1-4: Key Terminologies - DDD Building Blocks

Entity

Object with **distinct identity** that tracks state over time

Automotive Example: Vehicle with unique VIN

Value Object

Immutable object defined by attributes, not identity

Automotive Example: Speed, Temperature, Coordinates

Aggregate

Cluster of related objects treated as a **single unit**

Automotive Example: Vehicle with Components

Service

Stateless operations that don't naturally fit entities

Automotive Example: DiagnosticService, PaymentProcessor

Repository

Mediates between domain and **data mapping** layers

Automotive Example: VehicleRepository, CustomerRepository

Factory

Encapsulates **complex object creation** logic

Automotive Example: VehicleFactory, OrderFactory

LG 1-3 & 1-4: Concepts - DDD Building Blocks and Their Relationships

Key Relationships

Building Block Connections

- **Aggregates** contain Entities and Value Objects
- **Repositories** manage Aggregates
- **Factories** create complex Entities and Aggregates
- **Services** operate across multiple Entities

Design Principles

Aggregate Rules

- ✓ **Consistency boundary** - Ensures invariants
- ✓ **Single root** - One entry point to aggregate
- ✓ **Local persistence** - One repository per aggregate

Automotive Example

Vehicle Aggregate contains Engine Entity and Speed Value Object, managed by VehicleRepository

LG 1-3 & 1-4: Walkthrough - Implementing DDD Building Blocks

1

Identify Building Blocks

- **Entities** - Objects with identity
- **Value Objects** - Immutable attributes
- **Aggregates** - Related object clusters

2

Design Relationships

- **Repositories** - Manage aggregate lifecycle
- **Factories** - Create complex objects
- **Services** - Stateless operations

3

Implement in Code

- **Define invariants** in aggregate roots
- **Enforce boundaries** with repositories
- **Refactor** to improve structure

Automotive Example

Vehicle Aggregate with Engine Entity, managed by VehicleRepository, created by VehicleFactory

LG 1-3 & 1-4: Analogies - Building with LEGO

LEGO Building Blocks

- **Entities** - Unique LEGO bricks with special identifiers
- **Value Objects** - Standard bricks with fixed attributes
- **Aggregates** - LEGO models built from multiple bricks

Building Principles

- **Repositories** - LEGO storage containers for models
- **Factories** - Instruction manuals for complex models
- **Services** - Special tools for specific tasks

DDD Connection

Just as LEGO provides standardized building blocks for creating complex structures, **DDD building blocks** provide standardized components for creating complex software

LG 1-3 & 1-4: Connected Examples - Automotive DDD Implementation

Powertrain Domain

Entity

`Engine` with unique ID and state

Value Object

`TorqueCurve` - Immutable performance data

Aggregate

`Powertrain` - Contains Engine and Transmission

Repository

`PowertrainRepository` - Manages lifecycle

Infotainment System

Entity

`MediaSession` - Tracks user interactions

Value Object

`MediaMetadata` - Immutable track info

Service

`MediaStreamingService` - Handles playback

Factory

`SessionFactory` - Creates media sessions

Safety Systems

Entity

`SafetyEvent` - Records incidents

Value Object

`SafetyZone` - Defined risk area

Aggregate

`SafetySystem` - Manages safety components

Service

`CollisionDetectionService` - Analyzes sensor data

LG 1-3 & 1-4: Summary & Resources

Key Takeaways

Building Blocks

- ✓ **Entities** have identity and track state
- ✓ **Value Objects** are immutable
- ✓ **Aggregates** enforce consistency boundaries

Relationships

- ✓ **Repositories** manage aggregate lifecycle
- ✓ **Factories** create complex objects
- ✓ **Services** handle cross-entity operations

Resources

Books

- DDD Patterns
- Implementing DDD

Online Tools

- Modeling Tools
- Code Generators