## Assignment 1: Domain-Driven Design (DDD)

**Illustrating the steps in DDD and a real-world use case**

---

### 1. Introduction to Domain-Driven Design (DDD)

Domain-Driven Design (DDD) is a software development approach that focuses on **understanding the business domain** first and building the software model around it. DDD encourages close collaboration between developers and domain experts, ensuring the system reflects real business rules, workflows, and terminology.

The main goal is to create software that accurately represents the **core business logic** and is easy to maintain, scale, and evolve.

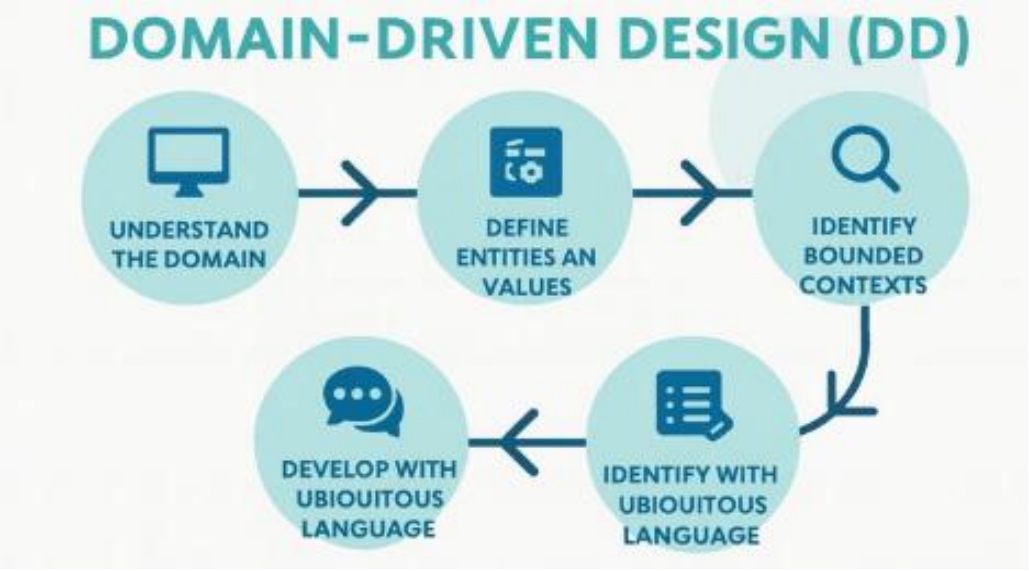
---

### 2. Why DDD Is Used

- Solves complex business problems
- Encourages clear communication between developers and business teams
- Produces clean, modular, domain-centric architecture
- Helps break large systems into manageable parts
- Increases flexibility for future changes

---

### 3. Steps in the Domain-Driven Design (DDD) Process




**Step 1: Understand the Domain**

Developers meet with domain experts to understand how the business works.
Activities include:

- Understanding workflows
- Identifying business problems
- Asking questions about rules and constraints

**Outcome:** Clear knowledge of what the business needs.

**Step 2: Identify and Define Entities & Value Objects**

Developers classify objects into:

- **Entities (unique identifiers)**
- **Value objects (no identity)**

**Example:**

- Entity → Customer, Product, Order
- Value Object → Address, Price

This step builds the foundation of the domain model.

**Step 3: Define the Ubiquitous Language**

A common vocabulary is created so that communication remains clear.
All teams agree on using the same terms.

**Example:**
"If a product is added to the cart, it becomes a Cart Item."

**Step 4: Identify Bounded Contexts**

The entire system is broken into small contexts (modules) with clear boundaries.

**Examples:**

- **Customer Context** – handles customer data
- **Order Context** – handles order creation
- **Inventory Context** – manages stock levels

Each context has its own logic and cannot interfere with others.

**Step 5: Model the Domain (Design Interactions)**

Each bounded context gets a detailed model:

- Entities

- Value objects
- Events
- Aggregates
- Repositories

This ensures every context works independently and communicates only through defined interfaces.

**Step 6: Implement the Model in Code**

Developers use the domain model to build actual software modules.
Each module strictly follows the rules defined in its bounded context.

**Step 7: Continuous Refinement**

DDD is iterative.
As domain experts give more insights, the model is updated and improved.

---

**4. Real-World Use Case: E-Commerce Platform (Shopping Website)**

To understand DDD practically, here is a real-world example.

**Domain: Online Shopping System**

**Domain Experts: Shop owners, logistics managers, payment team**

**1. Customer Context**

Handles:

- Customer registration
- Login
- Address management

**Entity:** Customer
**Value Object:** Address

**2. Order Context**

Handles:

- Add to cart
- Order creation
- Order tracking

**Entity:** Order
**Value Object:** Cart Item, Product Price

**3. Inventory Context**

Handles:

- Stock updates
- Warehouse quantities

**Entity:** Inventory Item
**Value Object:** Quantity

**4. Payment Context**

Handles:

- Payment processing
- Refunds
- Payment status

**Entity:** Transaction
**Value Object:** Payment Details