ENGINEER-TO-ARCHITECT AND DURABLE THINKING SKILLS

Domain-Driven Design

Level: Advanced • 2 days (expandable to 3) • Virtual, In-person

Overview

In complex business domains, the hardest problems are not technical. They are problems of understanding: the software slowly drifts away from how the business actually works, every conversation needs a translator, and changes that sound simple to the business become archaeology for the team. Domain-Driven Design attacks this directly, by making the domain model the center of the design and keeping the language of the code and the language of the business the same.

This is a hands-on, advanced course. It respects the most important lesson of DDD: strategic design comes before tactical patterns. So the gradient starts with judgment (where DDD pays for itself and where it is overkill), then builds shared understanding of a domain with domain experts, then carves that domain into bounded contexts, and only then descends into the tactical building blocks of aggregates, entities, value objects, and domain events. We go deep on modeling one realistic domain across the whole course rather than surveying every pattern in the book. Every module ends with a lab, and each module builds on the one before.

Who Should Attend

- Experienced developers and technical leads working in genuinely complex business domains
- Architects who need to decompose a large system along domain lines
- Teams adopting DDD who want a shared, practiced understanding rather than vocabulary alone

  Learners still building general design experience should take *Fundamentals of Software Architecture* or *Design Patterns in Practice* first.

Prerequisites

- Several years of professional software development experience
- Strong object-oriented design skills
- Experience with at least one system whose business rules were genuinely complex

What You Will Learn

- Judge where DDD earns its cost and where simpler design is the right call
- Build a ubiquitous language with domain experts and keep it alive in the code
- Identify bounded contexts and map the relationships between them
- Design aggregates that protect invariants and keep transactions small
- Model with entities, value objects, domain events, and repositories
- Fit a domain model into an architecture, from a modular monolith to microservices

Course Outline

Day one: strategic design

- Where DDD Pays Off

  - The core problem: model drift between the business and the code
  - Core domain, supporting domains, and generic subdomains: investing effort where it matters
  - When not to use DDD, and what to do instead
  - Lab: analyze a case-study business, identify its core domain, and rank the subdomains by strategic value
- Ubiquitous Language and Collaborative Modeling
  - Building a shared language with domain experts, and refusing to translate
  - Event storming: mapping a domain as events, commands, and policies on a wall
  - Turning conversations into a candidate model
  - Lab: run an event storming session on the case-study domain and extract its language
- Bounded Contexts and Context Mapping
  - Why one big model always fails, and how bounded contexts fix it
  - Context mapping: partnership, customer-supplier, conformist, and anticorruption layer
  - Aligning contexts with teams and with deployment boundaries
  - Lab: draw the context map for the case-study domain, including the relationships between contexts

Day two: tactical design

- Aggregates, Entities, and Value Objects
  - Entities and identity versus value objects and immutability
  - Aggregates: consistency boundaries, invariants, and why small aggregates win
  - Referencing across aggregates without coupling them
  - Lab: design the aggregates for one bounded context and defend each boundary against invariants
- Domain Events, Services, and Repositories
  - Domain events: recording what happened in the language of the domain
  - Domain services for logic that belongs to no single entity
  - Repositories and keeping persistence out of the model
  - Lab: extend your model with domain events and walk one business process end to end
- The Model in the Architecture
  - Layered and hexagonal architecture: isolating the domain from infrastructure
  - Bounded contexts as service boundaries, and where microservices fit
  - Evolving the model: refactoring toward deeper insight
  - Lab: place your bounded contexts into an architecture and record the decision as an ADR

Extended Version

The three-day version keeps the same gradient and adds depth and a capstone:

- A second, deeper event storming cycle: from big-picture to process-level design
- CQRS and event sourcing: what they solve, what they cost, and when to say no
- Working with legacy systems: anticorruption layers in practice
- A capstone: teams model a fresh domain from a live "domain expert" interview through context map, aggregates, and an architecture decision, then defend it in review