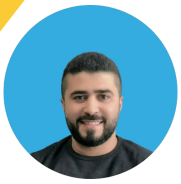
Domain Driven Design IN C#

Part 1

With Example of Bank System



Keivan Damirchi

Include:

- What is DDD?
- Why use DDD?
- When to use DDD?
- Key items in DDD
- Pros and Cons of DDD
- A Scenario with Implementation in C#

What Is DDD?

Domain Driven Design (DDD) is an approach to software development that focuses on designing software systems around the **business domain** and the **behaviors** and **interactions** of the business entities.

DDD is based on the idea that a deep understanding of the business problem being solved is essential for creating effective software solutions. By creating a common language between business experts and developers, and modeling the domain in the software design, developers can gain a deeper understanding of the problem and create a more effective solution.

The goal of DDD is to create software that is more maintainable, extensible, and well-organized. DDD provides a set of patterns and best practices for designing and implementing software using this approach. These patterns include concepts like entities, value objects, aggregates, repositories, domain events, and domain services.

Note
A domain in software engineering field is business on which application is intended to build.

Why Use DDD?

DDD enables the development team to build software that accurately reflects the business domain and its complexities.

It also helps to improve communication between business experts and software developers by creating a **shared language** that they can use to discuss the system. By modeling the software around the domain, DDD also helps to create a more maintainable and flexible codebase.

Note
It is not the customer's job to know what they want.

When To Use **DDD?**

DDD is most beneficial when building complex systems that involve multiple business domains, complex business logic, or significant changes to the requirements over time. It's also useful when the business domain is not well understood, and the development team needs to collaborate closely with the business experts to ensure the software meets their needs.

Note
When we are developing software our focus should not be primarily on technology, rather it should be primarily on business.

Keys Items In DDD?

Ubiquitous Language

Bounded Contexts

Aggregates

Entities and Value Objects

Domain Events

Repositories

Pros and Cons of DDD

Pros

- Improved understanding and communication between business experts and development team.
- More maintainable and flexible codebase.
- Clearer separation of concerns between different parts of the system.
- More testable code.
- Improved code quality.

Cons

- DDD can add complexity to the system and require more time to develop.
- It can be challenging to find a balance between modeling the domain accurately and building a practical software system.
- It requires a significant investment in time and effort to learn and apply DDD effectively.