# Advantages And Disadvantages

We've discussed most of the essential technical concepts in Domain-Driven Design. To wrap things up:

- Data
  - Value Objects
  - Data Transfer Objects
- Working with data
  - Repositories (or custom query builder in Laravel)
  - Services
  - CRQS
    - Actions
    - View models
  - States and transitions
- Code structure
  - Domains
  - Applications

If you think about them, these are straightforward ideas. Except for states and transitions, there's not even a "pattern" or a class structure behind these concepts. They take an idea (like structuring unstructured data with DTOs) and give us some classes that help us achieve the goal clean. This brings us to the most important advantage:

- **The technical aspects of DDD are easy.** We, developers, make it more complicated than it should be.

But these simple classes and ideas come with disadvantages as well:

- There are a lot of different concepts and jargon associated with DDD. This is why it's often misunderstood, and people make it more complicated than it should be.
- Also, at the code level, we're going to use a lot of different classes. It can be hard to get used to it if you're writing fat controllers and models.
- As a bonus, if you're browsing DDD articles or videos online, they are often written in a