Domain Driven Design - Clear Your Concepts Before You Start

Mahmud Hasan, 7 Mar 2012

4.88 (60 votes)

When you start a complex application, you always should design your object model first. Domain Driven Design guides you towards that.

Start Developing a New Application

What we traditionally do when we start a business application? We read the spec and find the functionalities. We break down tasks. In most of the cases the goal of the breakdown is to come up with an estimation and plan of works. We do the estimation. We distribute the works among team members. We design the database schema - sometimes by the team leader or sometimes by the respective developer. We start coding.

So? What's wrong with this approach? We have been doing good! Don't we?
The answer is YES and NO! Yes we are doing good in delivering our projects. But NO! We are not doing good in maintaining and extending our projects!

Think about all of the projects you have worked last few years in the traditional approach. Did you ever face any of the issues below?

1. Your project has the same functionality implemented in the same way or different in different places.
2. You have more than one object for the same item.
3. You have objects that have properties that are not actually attributes of that object.
4. You have no or very poor relationship among related items.
5. Looking at your objects it is not possible to understand what actually the whole application is all about.

I am sure you have been facing these issues regularly. But do you know why? The reason is, traditional approach do not guide us designing the system in Up to Bottom fashion. Rather it tempted us to design the system in Bottom-Up fashion. See, when you design a system you need to know what as a whole the application will do? What is the goal the client is trying to achieve? Then, from the top level goal you come up with different smaller functionalities that will eventually allow the users to achieve the top level goal.

But when you design in bottom-up approach, you first design for the granular functionalities, and you have little or no knowledge how this functionality will be used from the Top level and how the Top level functionalities will actually look like.

Have you ever heard that a developer of your team is talking like he does not have the domain knowledge of the whole application? Perhaps yes! I think you can understand the reason. Cause, the design of the application does not represent the domain of the system. And so, developers know only the portions they worked. This is Sad! Isn't it?

So, is traditional approach – "Designing the application starting from database" a throw away concept? Not really! But if you have a complex application to develop, this bottom-up design approach does not dictate you to come up with a proper object oriented design.

What is the solution then?

The solution is **DDD (DOMAIN DRIVEN DESIGN).**

What is DDD?

Domain-driven design is not a technology or a methodology. DDD provides a structure of practices and terminology for making design decisions that focus and accelerate software projects dealing with complicated domains.

- Wikipedia

Concepts to cover in this article:

1. Understanding the Domain.
2. Ubiquitous Language.
3. Contexts and Bounded Contexts.
4. Entities and Value Objects.
5. Aggregates and Aggregate Roots.
6. Persistence Ignorance.
7. Repository.
8. Domain Service.

In this article I will try to avoid becoming too technical, rather I will try to go through different concepts of DDD being close to the real world. I will try not to show any code here. Because I believe if you understand the concept and starts thinking in DDD way, implementation is easy. The toughest part is to tune your thinking process!

Understanding the Domain

A sphere of knowledge, influence, or activity. The subject area to which the user applies a program is the domain of the software.

- Wikipedia

Do you get a feeling what is domain from this definition? Can you tell what is the domain of the project you are working on at this moment? Can you tell what is the domain of the famous website YouTube?

In this article I would like to go through a real world example to give you the feeling how to start analyzing your project driven by your domain. This example may not be related with application development but as the goal is to tune our thinking top to bottom manner, it will be useful. But again, we will go through the technical terms of DDD too!

Let's say you are engaged to design a building. The requirement is:

- You have a defined amount of land
- Your building will have 6 floors.
- Each floor will have 4 apartments.

What is your domain here?

The domain is Building(?). It could be. But note that, if you consider Building as your domain you may miss few granular details for your requirement. The building you are going to design must have design for apartments where people will live. So, a general term "Building" can make us miss few details. So, we may narrow down our domain to "Residential Building".

Now, when you talk about your work with engineers and also with the people who engaged you to design the building, the term "Residential Building" is more meaningful for everybody concerned. Did you mark very small change in language here? The contractor is telling you to design a building where there will be 4 apartments in each of the 6 floors. Now, if you send an engineer to the site telling him we will need to construct a building there, they might not consider many attributes that a residential building must have. On the other hand if you use the term "Residential Building", most likely he will come with a valid analysis.
This is how we come to an "Ubiquitous Language".

Ubiquitous Language

The concept is simple, that developers and the business should share a common language that both understand to mean the same things, and more importantly, that is set in business terminology, not technical terminology.

More Example of Ubiquitous Language:

Example 1:

Wrong Language:

The length and width ratio of the smaller bed rooms would be 4:3.

Correct Language:

The children's bed room’s length will be 20 ft and width will be 15 ft.
Note that, to the owner of the building “smaller room”, “ratio” - all these things could be very technical terms. Rather it is easier for him to understand children's room, guest room, living room etc. And explicit measurement is more meaningful to him.

Example 2:

Let us see an example from software perspective.

Wrong language:

In search functionality we will consider inflectional and thesaurus feature of sql server to make the search more relevant. Additionally we will also exclude the stop words from the search to make it more accurate.
Note that, your domain expert may not be a technical person and thus he may not understand what you meant by the words “Inflectional”, “Thesaurus”, “Stop word” etc.

Correct language:

In the search functionality we will consider all the synonyms of the search phrase so that it does not exclude relevant results. Moreover we will not differentiate any search word by its number (singular or plural), tense, participle etc so that the result becomes more accurate. Additionally as expected in any search, we will ignore all the noise words that do not have any value in the search. Such noise words could be “am”, “but”, “where”, “about” etc.
Do you see the difference in language here? Really a correct language can make all the involved parties think and understand in the same way.

Let’s come back to our “Residential Building” domain. Look, you can proceed with the residential building design as a single task and address the whole thing together. But will it really be very wise way to do? Note that, if you just consider this a single unit of work you may miss many things. Designing a building is related to so many things. For example: you need to consider ventilation, utility, parking space, community space etc.

Now you see, different other contexts are coming up. This is how the concept “Context” and “Bounded Context” comes up in Domain Driven Design.

Contexts and Bounded Contexts

A Bounded Context can be considered as a miniature application, containing itss own Domain, own code and persistence mechanisms. Within a Bounded Context, there should be logical consistency; each Bounded Context should be independent of any other Bounded Context.
More Example of Bounded Context:

Think of an e-Commerce system. Initially you can tell it is an application of shopping context. But if you look more closely, you will see there are other contexts too. Like: Inventory, Delivery, Accounts etc.

Dividing a large application among different bounded contexts properly will allow you to make your application more modular, will help you to separate different concerns and will make the application easy to manage and enhance. Each of these Bounded Contexts has a specific responsibility, and can operate in a semiautonomous fashion. By splitting these apart it becomes more obvious to find where logic should sit, and you can avoid that BBOM (Big ball of mud) J

What is BBOM?

A Big Ball of Mud is a haphazardly structured, sprawling, sloppy, duct-tape-and-baling-wire, spaghetti-code jungle. These systems show unmistakable signs of unregulated growth, and repeated, expedient repair. Information is shared promiscuously among distant elements of the system, often to the point where nearly all the important information becomes global or duplicated. The overall structure of the system may never have been well defined.

- Brian Foote and Joseph Yoder, Big Ball of Mud. Fourth Conference on Patterns Languages of Programs (PLoP '97/EuroPLoP '97) Monticello, Illinois, September 1997

Our all time objective should be to avoid BBOM

Again with the "Residential Building Domain". So, we could have several bounded contexts:

- Electricity supply
- Car parking
- Apartment
- Etc.

Let's talk about the apartment. The apartment is basically a combination of different rooms. The rooms have different elements inside like windows, doors etc. Now I have 2 questions to you about the windows of the room.

Question1: Can you imagine a window without a room?

Question2: Does a window have any identity without the room it is residing in?

Answering these questions will expose the following concepts of DDD.

1. Entity.
2. Value Object.
3. Aggregates & Aggregate root.

Entity

"This is my Entity, there are many like it, but this one is mine."

The key defining characteristic of an Entity is that it has an Identity – it is unique within the system, and no other Entity, no matter how similar is, the same Entity unless it has the same Identity.

Examples:

1. Your bed room in the apartment.
2. Contract in Facebook.
3. Article in CodeProject.

Value Object

The key defining characteristic of a Value Object is it has no Identity. Ok, perhaps a little simplistic, but the intention of a Value Object is to represent something by its attributes only. Two value objects may have identical attributes, in which case they are identical. They don't however have any value other than by virtue of their attributes. Another aspect common to value objects is that they should probably be immutable, once created they cannot be changed or altered. You can create a new one, and as they have no identity, that is just the same as changing another one.

Example:

1. Windows in the rooms
2. Address of any person in your website.
3. SearchCriteria of your search.

Note: A value object can become an entity depending on the situation. Can you find a scenario like that? If the requirement of the search functionality of your application says that, the search criteria should be saved in the database and the user can do the same search from the list of saved search criteria's. In this scenario SearchCriteria has its own identity and thus it is an entity instead of being a value object.

Now you know what entity is and what value object in DDD is. In domain driven design entities and value objects can exist independently. But in some cases the relation can be such that, an entity or VO has no value without its context.

Example:

1. A window can only be defined if there is a room.
2. An order note can only exist if an order is placed.
3. A question detail can only be there if a question is asked.

Very simple is not it? Believe me, now you know what Aggregate and Aggregate root is in DDD.

Aggregate and Aggregate Root

In the examples given above –

- Room, Order and Question are our aggregate roots.
- On the other hand window, order note and question detail are our aggregates.

"A cluster of associated objects that are treated as a unit with regard to data changes."
All objects of the clusters should be treated as aggregate.
All external access to the cluster is through a single root Entity. This root entity is defined as aggregate root.

Example:

1. A question detail should no way be saved unless the corresponding question is saved.
2. A question detail should no way be retrieved unless the corosponding question is retrieved.

Here Question is the Aggregate root and Question Detail is the aggregate. Aggregates and Aggregates Root are very important concepts of DDD.

So far we have talked about domain, objects/entities, contexts, aggregates etc. What about the Database? Is that something we have missed? Isn't it something should come in the design?

The answer is NO! DDD is a persistence ignorant approach.

Persistence Ignorance

In domain driven design your objective is to create a model of the domain. You need to identify what are the items (objects) you need to accomplish the desired functionalities of your application. You need to identify the relationships among different objects and how they interact among themselves. You need to find if the business goal of your client is achievable using your domain model. Where is the existence of database here? You do not need to know how and where the data of your domain will persist or even if the data do need to persist while you do the model of the domain.

This ignorance about your persistence medium will make your domain model free from any coupling with the persistence layer of the application. This will eventually separate the concerns of the persistence and its communication mechanism from your domain model. In result your application will be free from coupling with any data store and will be very easily unit testable.
But Yes! In a real application you do need to have a database. But your domain model will have no knowledge about that. All it will know is the "Repository" which will eventually manage your application's persistence concern.

Repository

Can you tell me what the meaning of the English word "Repository" is?

Repository commonly refers to a location for storage, often for safety or preservation.
- Wikipedia

As I have already said your domain model will not know any database. What it will know is, there is a repository in the system and that repository will be responsible to store your data and to retrieve your data. It is no way a concern of your domain model how and where data will persist. So, it can be Sql server, oracle, xml, text file or anything else. I hope now you got a sense what a repository means in DDD.

Let's become little more technical.

Repository Mediates between the domain and data mapping using a collection-like interface for accessing domain objects. It is more like a facade to your data store that pretend like a collection of your domain.

`Repository Is Not A Data Access Layer.`

Note that repository doesn't talk in terms of "data", it talks in terms of Aggregate Roots. You can tell your repository to add an Aggregate Root into its collection, or you can ask it for a particular Aggregate Root. When you remember that Aggregate Roots may comprise one or many Entities and Value Objects, this makes it fairly different to a traditional DAL that returns you back a set of rows from your database tables.

Implementation Strategy of Repository:

As I said, Repository is a design pattern that is used in DDD to handle the persistence concern. The detail of this pattern is out of the scope of this article. However, here I am trying tell in minimum how we may achieve a repository implementation.

1. 1st of all you will have an interface - `IRepository` that should be generic.
2. You will have an abstract implementation of the `IRepository` interface.
3. You will have interface `INhRepository` for your persistence mechanism (i.e. `Nhibernate`) this will inherit from `IReposiroty`.
4. You will have implementation of `INhReposiroty` in a class like "`NhReposirory`".
5. Finally you may have a generic implementation of the repository that will have default implementations of all the common methods of the repository.
6. Like `NHGenericRepository` that inherits from `NhRepository` and implments `IGenericNhReposirtory`.
7. You will specific repository for you Aggregate Roots, that will be extended from `NHGenericRepository`.
8. Your application will use service locator to find which repository the application will use.

Domain Service

Domain service is another important concept of DDD. If Entities and Value Objects are the "things" in your domain, the services are a way of dealing with actions, operations and activities.

Shouldn't Logic Be on the Entities Directly?

Yes, it really should. We should be modeling our Entities with the logic that relates to them and their children. But, there are occasions when we need to deal with complex operations or external responsibilities or maybe we need to expose the actions of the aggregate roots to the external world. This is why creating a domain service for different aggregate root is a good idea. You can consider the domain services as façade layer of the business logics and operations of your domain.

End Words

In this article I have tried to introduce the basic concepts and terminologies of Domain Driven Design with examples of real world. The goal was to make you feel comfortable with DDD world. But really developing applications with DDD is a big challenge. The more you love and practice DDD concepts while you design your object model, the more accuracy you will gain in your design. As I said before the most important thing is, you must think in Domain Driven Way. If you don't you will hugely suffer when your application is a real complex one.

Great DDD Resources

http://www.dddcommunity.org/

http://thinkddd.com/

http://www.infoq.com/minibooks/domain-driven-design-quickly

http://thinkddd.com/assets/2/Domain_Driven_Design_-_Step_by_Step.pdf

http://www.dddcommunity.org/library/young_2010

http://www.dddcommunity.org/library/evans_2010

http://blog.fossmo.net/post/Command-and-Query-Responsibility-Segregation-(CQRS).aspx

http://msdn.microsoft.com/en-us/library/ff649690.aspx

License

Share

About the Author

Mahmud Hasan

Software Developer (Senior)
Netherlands

Software Engineer | Software Architect | System Designer | System Analyst | Team Leader | Consultant (.Net)

9 Years of Experience in the Industry.

Currently working as System Designer at CIMSOLUTIONS, Netherlands

View My Profile in LinkedIn

You may also be interested in...

Getting Started with the Intel® Edison Board on Windows

Red Hat JBoss BPM Suite 6 Compared to Pegasystems Pega 7 BPM

Scan Anything, Anywhere, Any Time

Beyond RDBMS: A Guide To NoSQL Databases

Red Hat JBoss BPM Suite 6 compared to Appian 7 BPM Suite

Why NoSQL?

Comments and Discussions

19 messages have been posted for this article Visit **http://www.codeproject.com/Articles/339725/Domain-Driven-Design-Clear-Your-Concepts-Before-Yo** to post and view comments on this article, or click **here** to get a print view with messages.