Sign in

articles　　quick answers　　discussions　　features　　community　　help

Search for articles, questions, tips

Articles » Development Lifecycle » Design and Architecture » Application Design

Domain Driven Design: A "hands on" Example (Part 1 of 3)

Suarte

20 Apr 2016　　CPOL

Rate this: ★★★★★ 5.00 (1 vote)

A hands on example of domain drive design

CodeProject I have received feedback from some readers of my last post, "Domain Driven Design: what is it really about?". Some of them mentioned that it is pretty difficult to get the hang of it, once DDD concepts seem to be very abstract. I must admit: it was not easy for me! Therefore, I believe there is no one who is the owner of the truth. I am still learning… and probably it will never end.

This way, my goal here is just to try helping others to understand some of the core concepts of DDD and how to apply it. Please, do not expect any kind of good practices handbook coming from this post! I will share with you guys part of my knowledge and experience on approaching software modeling problems using the DDD philosophy.

The Problem

I will pick an example from an "e-Commerce" system. You will find a pretty similar one (but not evolved as the one presented here) in the book "Implementing Domain-Driven Design", by Vaughn Vernon (which I recommended in my last post). Vernon's book can be a little bit hard to follow and understand at first, so I recommend that you fight yourself and read the three first chapters. There are a lot of concepts and things that seem to be weird and confusing, but might become clear at the end.

So, let's go to our hypothetical example. You have to build a whole new system in order to support the e-Commerce operation of the company where you work. Your solution must include:

- **Product search, Catalog**: Users on your website must be able to search and see information about products
- **Orders, Payment and Delivery**: Your customers should be able to place orders, pay for it and receive their goodies

- **Inventory**: Your solution must provide inventory control for each product that is available to sell
- **Authorization, Authentication**: Your solution must be able to identify and authorize valid users; the same for customers
- **Customer management**: Your solution must be able to manage customer registries

A Non DDD Approach to the Problem

Given the information above, without applying DDD, we could approach the problem this way:
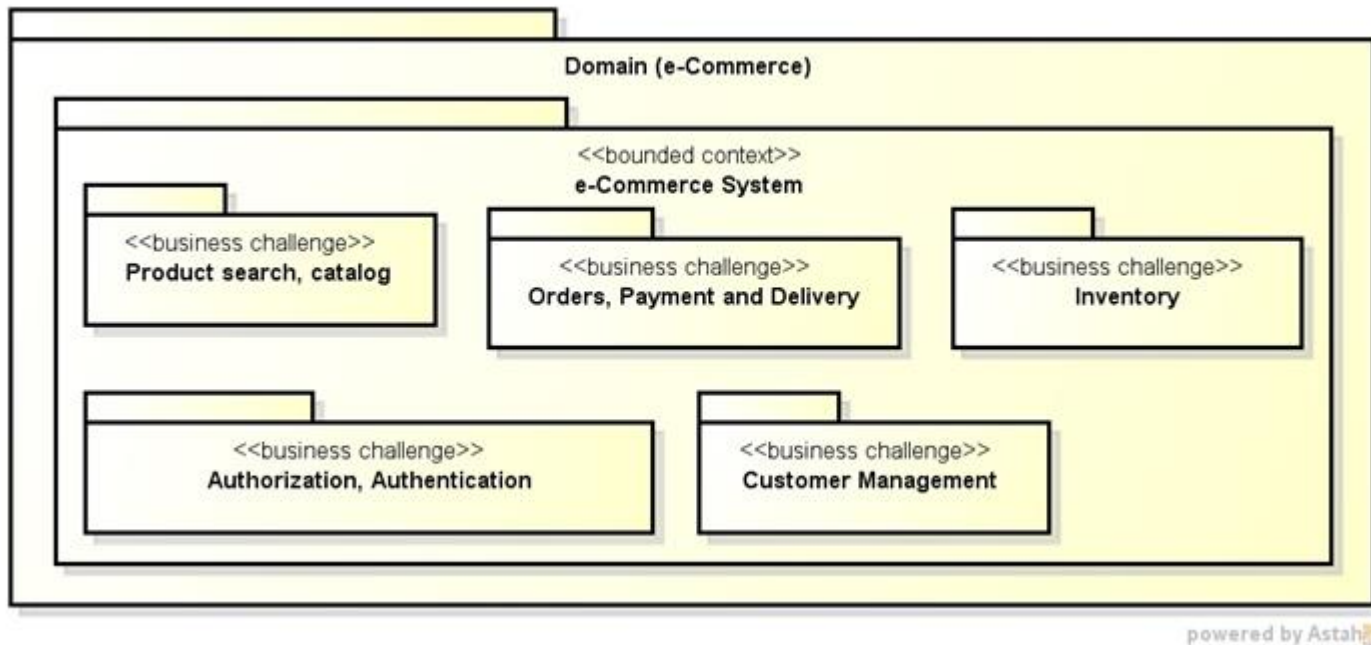
Picture 1: A possible solution (not a Domain Driven Design one)

I have used that package notation but it is not important. It could be anything, such as drawing circles in a white board. The important thing to notice is: this solution is intended to tackle the problem using a single Bounded Context.

This way, you will have to use all your knowledge of OOD (Object Oriented Design) in order to construct a Domain Class Model that supports all these business challenges. This task can be very, very hard! It would be necessary to represent many business requirements and behavior in a single Domain Model. Believe me: it can be a pain even for those more experienced professionals.

I will not dig in architecture details about how this system / application could be built. The fact is: we have only one Domain Class Model and we can develop a single application for this. Have you noticed what Bounded Context means here? Having only one Bounded Context means: a single view of the problem = a single Domain Class Model to solve all that business challenges. We are going to talk more about Bounded Contexts further in this post.

Applying DDD

Let's start applying some DDD to this problem. We should start identifying possible Subdomains inside our Domain. What the hell does it mean? Basically, a Domain is the business itself. It is what your company does and, consequently, it is the "problem" you want to solve. As we have to build an e-Commerce System, our Domain is the e-Commerce business. A Domain has its own **strategic challenges** which can be seen as **Subdomains**.

In other words, a Domain can be split in several Subdomains (a small specific view of part of the problem). Each of them can be classified as Core, Support and Generic. Once you have split your Domain in Subdomains, it is a good practice and actually desired that you set a Bounded Context for each Subdomain. I have explained concepts like "Bounded Context" and "Ubiquitous Language" in the post which has led me to this one (see the link on top), but I will try to reinforce it now.

According to Vernon's book (see the link on top again), the following description would be a good definition for "Bounded Context":

"*A set of specific software models, a specific solution expressing its own ubiquitous language.*
It is a desirable goal to align Subdomains one-to-one with Bounded Contexts.
Does not necessarily encompass only the domain model. It often marks of a system, an application and / or a business service."

This way, my sense tells me that a good solution for the problem, applying these DDD concepts, would be:
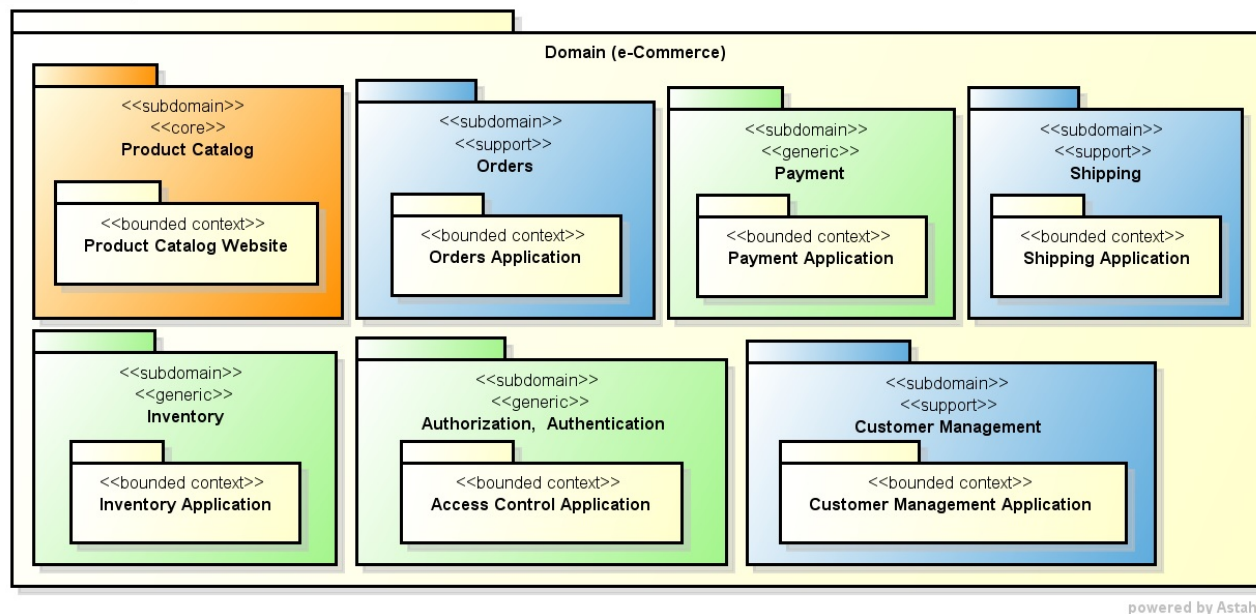
Picture 2: DDD approach to the problem

As it is possible to observe, I took strategic business challenges which are pertinent to the Domain and figured out some Subdomains. For each of them, I set a Bounded Context and classified it accordingly (Core, Support or Generic). Usually, there is only one Core, that represents the main part of the Domain. In this case, I think that the "Product Catalog" Subdomain is the Core because it is what customers will be interacting and, therefore, is from where the revenue will come (customer shopping). Others Subdomains were classified as Support and Generic.

Support Subdomains are like auxiliary ones. In practice, you set a Bounded Context to it and create a specific application. It will work as a support application for the Core Domain application, however, support applications will have its own model.

Generic Subdomains are like support ones, but they have a strong particularity: they are so generic solutions that they could be used not only in the Domain it was created, but they could be used by others Domains too. It is completely reasonably that your "Access Control Application", when well designed, can be reused to support other Domains which not the e-Commerce, for instance. That is why they are called Generic.

It is not difficult to realize that I have just organized my ideas about how I intend to approach the problem, and I have used DDD concepts in order to accomplish that. So, I ended up realizing that I would get a bunch of applications to develop. Perhaps, for some of these applications (like the Inventory one), an off-the-shelf software application would fit well. Also, it is perceptible that and integration between all these applications would be necessary.

If you imagine a real life scenario in a big e-Commerce company, it is possible that more than one team is allocated to support this operation. Possibly one team per Bounded Context.

DDD offers a way to classify and treat relationships between Bounded Contexts. If you have heard terms like "Partnership", "Shared Kernel", "Customer-Supplier", "Conformist", "Anti-corruption Layer", "Open Host Service", "Published Language", "Separate Ways" and "Big Ball of Mud" (my favorite one), you probably know what I am talking about. The details of these concepts are out of scope of this "hands on" example, but all of them can be found in the recommended book above (see the link on top).

Conclusion

We have just seen a problem description, a very simple and traditional way to approach it and a Domain Driven Design approach too. In this manner, we figured out that we will have several applications, very specialized applications, and that it is going to be necessary to put in some effort in order to integrate them. In the next post, I will explore more technical examples regarding the Domain Class Model of some of these applications. For each Bounded Context, we should build a Domain Class Model. It is there that your OOD (Object Oriented Design) knowledge shines. :)

Related Posts

- Domain Driven Design: what is it really about?
- Domain Driven Design: a "hands on" example (part 2 of 3)

License

This article, along with any associated source code and files, is licensed under The Code Project Open License (CPOL)
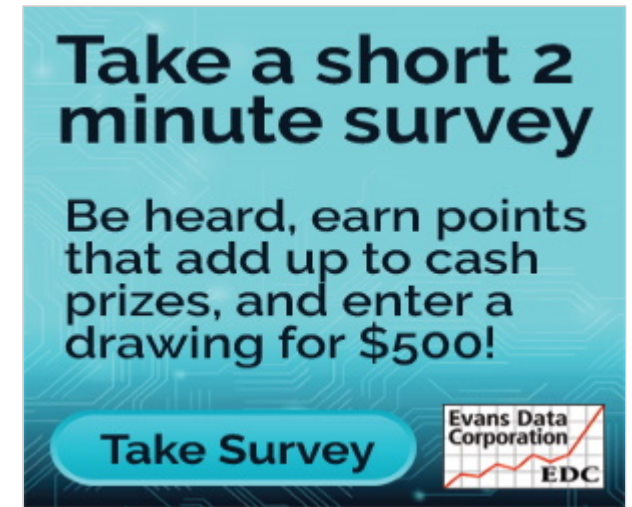
Share

About the Author

Suarte

Technical Lead PSafe Technology

Brazil 🇧🇷

A software development lover with more than 10 years of experience on programming, the IT field and computer science as well.

Comments and Discussions

You must **Sign In** to use this message board.

Search Comments

Spacing   Relaxed ▼   Layout   Normal ▼   Per page   25 ▼   Update

First   Prev   Next

Part2 📌   sri_mncl   **20-Oct-17 5:22**

Re: Part2 📌   qrlodhi   18-Apr-18 20:19

Last Visit: 30-Mar-20 4:10   Last Update: 1-Apr-20 13:48   Refresh   1

General   News   Suggestion   Question   Bug   Answer   Joke   Praise   Rant   Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

Permalink
Advertise
Privacy
Cookies
Terms of Use

Layout: fixed | fluid

Web04 2.8.200331.1