xingfucoder

GENERAL

[Domain Driven Design] - Extracting Subdomain and Bounded Contexts with an example

Posted **5 years ago** by **xingfucoder**

According of structuring of a project in Laravel and with a more advanced Business Logic, there are two concepts that are difficult to differentiate. The **Bounded Contexts** and **Subdomains**. When organizing a large application, how materialise these two concepts? Can someone give us a real example to discern and see the difference?

In the Laracasts example I would see the Domain as Learning Laravel System. And the subdomain Lessons Management, Accounting, Payment, Forum, but what would be the Bounded Contexts here?

And what will be the relationship with Modules and Namespaces?

If someone could show any other example will be useful.

Thanks in advanced.

...

MDWHEELE

BEST ANSWER

m eele • 5 years ago

BEST ANSWER

I g to keep this short as I wanted to answer, but am strapped for time and don't have a clear picture about what **exactly** you're looking for. One of the **MOST** important thing about domain-driven design questions and answers is that you **MUST** always ask questions within a

context. As with a lot of things, "it depends" comes up a lot.. even more-so in DDD.

First, a subdomain represents a delimited (partitioned) piece of the overall domain. A domain could be something like "Scholarships Management". There will be several subdomains inside of that. You might consider something like:

- Identity and Access Management (generic subdomain)
- Student Application (core)
- Selection Process (core)
- Financing (core)
- Logging and Audit (supporting subdomain)

Subdomains represent partitions of the "problem space". They represent different logical areas of responsibility in an application and they are completely dependent on the domain. Don't go on a witch-hunt expecting to find a bunch of these either. I've seen plenty of people try to split a simple application into 5-6 subdomains. It doesn't make sense. You have to make sense of the situation you're in and really get good at "reading the lay of the land". It takes practice.

Bounded context represents the ubiquitous language of (in a perfect world) a single sub-domain. This is where things get sketchy, but for sake of example... while subdomain is definition of focused problem space, bounded context is the application of a domain model that is formed by the UL as a solution to a particular subdomain. This is a little bit of a general statement, but Bounded Context (as an applied domain model) can be thought of as "the solution" to the problem presented by a sub-domain.

As I mentioned, a bounded context mapping one-to-one with a subdomain isn't always the case, but what's important is that the identification of a bounded context occurs because you discover that the language your business uses changes based on context. For example, when a student starts to complete an application for a scholarship, they think of themselves as a "student". However, when they meet the requirements for some scholarship, they become a "candidate" in the context of selection. The difference is subtle, but important. In addition, personnel in the "finances" context may not even care that a student exists as a concept at all.

So... this got long... back to your question...

> And what will be the relationship with Modules and Namespaces?

contrived example, I might lay my project out like:

```
src/
  ArbitraryBoundedContext/
    Domain/
      // If I have a TON of collab b/w ARs, I'll actually split my domain
      // into a "services" module and "model" module.
      ParticularDomainService.php
      Aggregate/
        AggregateRoot.php
        HappyChildEntityOfAggregate.php   // It happens... keeping it simple here.
        AggregateRepository.php        // Define interface in the domain.
        RelatedValueObject.php
        Events/
          AggregateRootDidSomeMeaningfulThing.php
          AggregateRootDidSomethingTheDomainCaresAbout.php
          // This is an example... but if your events are this long...
          //you're probably doing too much :)

    // Application & UseCases are the same thing really; depends on
    // what you want to do. More command-bussy? Usecases are great
    // to fit into that paradigm. Simpler bounded context? Keep it simple
    // with a service class.

    Application/
      ApplicationServiceWithAFewMethodsAllInOne.php
      SomeFacadeForOtherBoundedContextsToIntegrateWithMe.php
    UseCases/
      Commands/
        DoSomeUseCaseGoodName.php
      TightlyFocusedUseCaseClassThatHandlesCommands.php
    Infrastructure
      Persistance
        ImplementationOfAggregateRepository.php
```

So, the above is just arbitrary and makes sense for *me*. Again, don't go hunting for the **right way** to lay your project out. Get a grasp on the concepts and then agree on a structure that you and your team can swallow and reason about.

This went a lot longer and I'm sure I made some mistakes in reasoning, but there it is. Feel free to ask any follow-ups and I'll try to be helpful!

// edit I left out Laravel!

T onestly the beautiful thing about explicitly declaring an application layer whether it be a big "use-case class" with individual methods to ha different use-cases (similar to a traditional Laravel controller you'd see in a tutorial in the wild) or something more robust like the

command-bus examples on Laracasts. Either way, your code is framework agnostic. You defer technical decisions until the last possible moment (or not; your choice.. and that's the point). So, back to Laravel...

Under the above project structure, I treat my routing layer and controller as almost one "HTTP transport facilitation mechanism". I use the framework to process HTTP messages, convert them into some form of "an intention" on my domain (whether it be a Command class or a method call on an application service) and then I catch exceptions from that execution and respond accordingly. If no exception, I'll most likely render a Blade / Twig template that I give a Presenter from my application. Again, whether I use Blade or Twig for a project doesn't matter. The "Presenter" or "Read Model" is the authoritative source of truth for me. Getting it back to the user as an HTTP response is up to the framework and I'm free to choose path of least confusion! :)

A typical controller method in a project like this, for me is:

1. Grab inputs from framework to try and ...
2. Construct a valid command for my domain that represents a user's intention to do something
3. Execute that command against an application service representing use-case of the bounded context
4. Catch any exceptions and respond accordingly. This might be a `CommandValidationException` that was thrown as soon as I tried to construct the command or it might be an exception from some business rule / invariant that was broken.
5. Fall through to whatever the "success case" should be. Usually, this is either going to be "redirect to some other named route" or "render a page" etc.

Posted **5 years ago** by MDWHEELE (LEVEL 3)

4 Reply

MDWHEELE

mdwheele • 5 years ago

Also, would like to recommend a few resources that have been immeasurably helpful to me learning about all this stuff:

- **dddinphp.org**
- **Domain Driven Design, Eric Evans**
- **Implementing Domain Driven Design, Vaughn Vernon**

The "red book" is a little more "down to earth / practical", but both are excellent reads and highly recommended by folks in the DDD community.

Posted **5 years ago** by MDWHEELE (LEVEL 3)

1 Reply

XINGFUCODER

xingfucoder • 5 years ago

Many thanks **@mdwheele**, I have a bunch of information to read and try to understand better. I have the red book but some concepts are hard to understand :).

Only one question, when you are using the ArbitraryBoundedContext. Could you put an example related with the sample that you explained? This concept is quite abstract, at least for me.

And for this comment, what does ARs mean? Aggregate roots?

```
// If I have a TON of collab b/w ARs, I'll actually split my domain

   // into a "services" module and "model" module.
```

Thanks in advanced.

Posted **5 years ago** by XINGFUCODER (LEVEL 14)

Reply

MDWHEELE

mdwheele • 5 years ago

> nd for this comment, what does ARs mean? Aggregate roots?

Yep! Sorry! And the thing there is that tends to not happen that often. Honestly, having a bunch of domain services orchestrating aggregate roots is probably a sign there's something off in the model because your aggregate roots have a lot outside the consistency boundary going on. It depends!

> Could you put an example related with the sample that you explained?

Do you mean an example from the "Scholarships" domain I proposed?

Posted **5 years ago** by MDWHEELE (LEVEL 3)

Reply

XINGFUCODER

xingfucoder • 5 years ago 

Yeah, but I thought that the Bounded Context rarely was directly linked with the sub-domain.

In this example you refer as Scholarships domain and you set as the Bounded Context.

Sincerely, I saw all sub-domains as Bounded Context :(

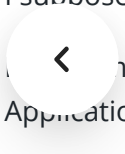
Posted **5 years ago** by XINGFUCODER (LEVEL 14)

Reply

XINGFUCODER

xingfucoder • 5 years ago 

I suppose that a Domain Service should not be only related to an entity, but with several entities in the domain model, correct me if I'm wrong.



...e confusing concepts such as Commands, because if those are related with our Domain Model, why do I need to put within the Application Layer? I would put as a Domain Service.

Posted **5 years ago** by XINGFUCODER (LEVEL 14)

Reply

MDWHEELE

mdwheele • 5 years ago

Yeah, I think the formatting is throwing us off a bit. Uhm, I'm going to put a **small** example to words and kinda go through how I break things down in the domain conceptually and then how I *might* go about building a domain model within a bounded context. We'll stick to just one bounded context to make sure we're on the same page and then maybe integrate a second. Big reply incoming.

In the example project structure above, I meant to communicate that a project might have **many bounded contexts** that will each have their own domain [model]. A domain [model] with sufficient amount of services might warrant a package / namespace / module to **just** for the services, so I **might** mark that off explicitly. But do note that the "Domain" module under a given "BoundedContext" in my example represents the "Domain Model" of a particular "BoundedContext" which represents a solution to the problem a particular "SubDomain" in a "Domain" presents.

The terminology conflates a bit over the boundary between the problem-space and the solution-space, which can be confusing. It's actually a little ironic when you think about it! A common term with different meaning in different contexts :)

Posted **5 years ago** by MDWHEELE (LEVEL 3)

2 Reply

XINGFUCODER

xingfucoder • 5 years ago

Many thanks **@mdwheele** for all your time. If you are busy, you can answer this question after. I don't want to take your time.

Posted **5 years ago** by XINGFUCODER (LEVEL 14)

Reply

XINGFUCODER

xingfucoder • 5 years ago

Ok **@mdwheele**, depends on your target and the complexity, sometimes the Bounded Context could be one of the sub-domain that you explained here (Identity and Access Management, Student Application, ...) or if you are working with all those sub-domains this can be converted as modules or sub-models within the big Bounded Context, is correct?

For example for the Scholarship, the Domain and the Bounded Context would be the same thing, if I'm not wrong, but if you are working within the Selection Process (Domain) the other parts could be another Bounded Contexts that you integrate within your application, using some of the connection method (shared kernels, ...)

Tonight I'm going to have nightmares with the Bounded Context. ahahaha

Posted **5 years ago** by XINGFUCODER (LEVEL 14)

Reply

SP1966

SP1966 • 5 years ago

I would love to see what **@mdwheele** writes when he's not "going to keep it short..."!!

Posted **5 years ago** by SP1966 (LEVEL 4)

Reply

MDWHEELE

ele • 5 years ago

I'm so sorry! Hahaha... I have a hard time being concise in communication a lot of the time. It's something I'm working on. Currently; paring down the other tab... makes Twitter interesting.

Posted **5 years ago** by MDWHEELE (LEVEL 3)

2 Reply

SP1966

SP1966 • 5 years ago

Don't be sorry, it is a great and very useful post!

Posted **5 years ago** by SP1966 (LEVEL 4)

1 Reply

XINGFUCODER

xingfucoder • 5 years ago

I think I need to read this post more slowly to get a better understanding of all concepts. Many thanks **@mdwheele.**

Posted **5 years ago** by XINGFUCODER (LEVEL 14)

Reply

MDWHEELE

mdwheele • 5 years ago

Haha! You're over-thinking it! Free up your mind and try to not think so technically. Inhale! Exhale! No nightmares required. :)

> Ok **@mdwheele**, depends on your target and the complexity, sometimes the Bounded Context could be one of the sub-domain that you explained here (Identity and Access Management, Student Application, ...) or if you are working with all those sub-domains this can be converted as modules or sub-models within the big Bounded Context, is correct?

So, both scenarios you describe can happen! Let's talk about what that means though and what happens when one bounded context overlaps many sub-domains. Forget about folder-names for a second, we'll come back to those.

When we reason about the **Domain** of "Scholarships Management", we might identify several **Sub-domains** in talking with the business (IAM, Student Application, Selection, etc). So we have one big problem-space (domain) broken down into a few smaller problem-spaces (sub-domains). But... maybe your team doesn't make this discovery and instead think that "Student Application" and "Selection" are actually the same thing; one bigger sub-domain. We'll continue with that assumption because this is real-life after-all and we shouldn't expect to be right 100% of the time.

Now, we need to model these domains. We model domains with a ... wait for it ... **Domain Model** (By the way, I'm not trying to be condescending here. Just bad humour and I'm trying to consider others coming to thread that may not have as good a grasp on the terminology). A **Domain Model** is a model of concepts from a particular domain or sub-domain that guarantee certain arbitrary business rules or invariants are held consistent. It follows the **Ubiquitous Language** from your **Domain** / **Sub-domains**. This is where **Bounded Context** comes in.

Consider that once you have a **Domain Model**, you're well on your way to architecting a solution to whatever problem you're solving for your business. If it is a simple domain, it will probably have a simple domain model. And if the language is concise and well understood by your organization then it may just be good enough to leave as is. A **Domain Model** operates in a **BoundedContext** and reflects the **Ubiquitous Language** within that context. So, if your **Domain** is simple, you may only have one **Bounded Context**. This is great! It means you get to finish the job early! :)

```
# Project Structure Aside

In the case that you truly only have a domain model in a single bounded context, it may not make sense to have a folder for your bounded context in the project.
You might just be able to get away with your single simple domain model at "src/Domain" and application / infrastructure services also under "src" respectively.
However...
```

metimes find that there will be a "tension" in trying to apply the **Ubiquitous Language** too broadly. **Bounded Context** helps us make exp differences / boundaries where the language changes.

Since we assumed that a our example "Student Application" and "Selection" sub-domains were really one big space, we might hear the following from "the business":

Student: Given I have filled out all requirements for a scholarship application, when I submit my application, then I expect that I will be shown a confirmation of some sort.

Selection Committee Member: Given there are valid student applications for a particular scholarship, when I view **students eligible for** the scholarship, then I expect to see some particular students.

Now, for the sake of argument, we met with the students first and we went ahead and started creating a domain model that fit their needs before seeking a committee member. If we modeled this, the Student might have name and some other identifying information. A student might `attachWorkExperience(...)`. They might `apply()` for a given scholarship period. It's going to do some more stuff, but we'll keep it lean.

We go and have a meeting with the Selection Committee. They give us their story and we're thinking about our student model, all nice-and-shiny. Well, the committee says that given some student applications (and we're thinking [not listening... mind you], "Yes! We can do that because we have `apply()`"), I want to view "students who are eligible for a scholarship". **Crap!**

So we start modeling again and decide we're going to make a student responsible for knowing whether or not they're eligible for a scholarship... then more business rules... then more; and the Student class grows and grows. **Note: ** This is a GROSS over-simplification but **the point is** that you're getting mixed concerns in the Student object. What has happened is that the meaning behind the word "Student" has been conflated. This is actually pretty awesome though because you're seeing that your domain's ubiquitous language is having immediate physical impact on the model. The feedback loop you've created with the business is working!

You feel the friction and you have a break-through:

```
Our students don't care anything about scholarships, they want FREE MONEY! And the selection committee isn't really concerned with a student so much as they are
CANDIDATES for scholarships.  Their business driver is to AWARD scholarships.
```

You grow the language and now you see the beginnings of a separation of your original big domain into two separate sub-domains. With that comes an opportunity to break your conflated domain model into two according to bounded context.

... Back to reality. You've gotta get this thing into SublimeText or phpStorm or whatever... I treat bounded contexts as separate units, each co...te with their own domain model (which we covered a ton above), use-cases (that drive that model), and infrastructure (because ...es). When it comes to how I lay my entire project out, it's separated based on bounded context, so I start there in my `src/` folder.

```
src/
    StudentApplication/
      Domain/
      UseCases/
      Infrastructure
    Selection/
      Domain/
      UseCases/
      Infrastructure
```

Then I keep on modeling. I determine my aggregate roots as primary points of entry. For the student application, it's probably a `Student` model or maybe even a `Profile` housed within the aggregate. The `Student` model (which is an aggregate-root) generates **Domain Events** when important things happen. **Aside:** anytime your business says "When this happens, then this other thing should do this thing", you just learned about a new domain event.

However, **DO KNOW** that this is an entirely iterative process driven by interaction with your domain experts (the business) and a sound BDD/TDD workflow. This is where the "don't go on a witchhunt" comment came from. Start simple and build complexity "just-in-time" and when it makes sense!!

I cut this mental train short because I need to run to the grocery store! But I'll be back later to maybe extend a little and fill in gaps. Also, I'm not an expert on this by any means. This is just my current understanding, so don't take everything I've said for granted. Let's keep the discussion going!

Posted **5 years ago** by MDWHEELE (LEVEL 3)

1 Reply

FAISAL.ARBAIN@GMAIL.COM

faisal.arbain@gmail.com • **5 years ago**

nice example :D

Pr... **ears ago** by FAISAL.ARBAIN@GMAIL.COM (LEVEL 3)

Reply

XINGFUCODER

xingfucoder • 5 years ago

Great example, I will read slowly and comment here **@mdwheele.**

@faisal_arbain, would be great your sharing knowledge about it.

Posted **5 years ago** by XINGFUCODER (LEVEL 14)

Reply

XINGFUCODER

xingfucoder • 5 years ago

OMG! Many many thanks, I'm beginning to understand some concepts. Great explanation.

Posted **5 years ago** by XINGFUCODER (LEVEL 14)

Reply

FAISAL.ARBAIN@GMAIL.COM

faisal.arbain@gmail.com • 5 years ago

@codeatbusiness I also still learning in DDD. But this discussion is really great. its provide me rooms to validate my understanding :D

@mdwheele please correct me if im wrong:

- `Domain` here is **Scholarship management**. since the *problem space* is too large, we split them into several `sub-domain`.
  udentApplication's `sub-domain` we have **student** `DomainModel` that was modelled specifically/good enough for **StudentApplication**'s
  `-domain`

- in **Selection**'s `sub-domain` the DomainModel is not the **student** anymore but **Candidate**. is it?

I think im lost..where is the `Bounded-context` and `UL`.

Posted **5 years ago** by FAISAL.ARBAIN@GMAIL.COM (LEVEL 3)

1 Reply

XINGFUCODER

xingfucoder • 5 years ago

Yes, **@faisal_arbain**, I think this thread is great for get more knowledge about DDD.

In your questions, I think you are right and with the last, I think the **Student** entity can be used by the two Bounded Context or sub-domains by using the shared kernel, correct me if I'm wrong, because this entity will have different contextual behaviour depends on the context it works, and although it will be using in the Infraestructure Layer the same storage source, the attributes used and their behaviour differs between those entities in each Bounded Context but the kernel entities don't change.

Posted **5 years ago** by XINGFUCODER (LEVEL 14)

Reply

FAISAL.ARBAIN@GMAIL.COM

faisal.arbain@gmail.com • 5 years ago

@codeatbusiness if i understand correctly, (eloquent) Student entity can be shared, but `Domain Model` for each `Sub-domain` is separated. I not sure myself actually.

but in red book there is a part where the *forum sub-domain* fetch user from `IAM` but then convert the user into a VO called `Moderator` where moderator will contains its own business logic that only makes sense for the forum.

f t understanding, I think that **Selection** `sub-domain` not interested in whole Student information but selective information as a **Candidate**. r myself.

Posted **5 years ago** by FAISAL.ARBAIN@GMAIL.COM (LEVEL 3)

Reply

XINGFUCODER

xingfucoder • 5 years ago

Yeah, you are right. I suppose that the moderator will be an Eloquent User or another Entity of the shared Kernel, but can be decorated with some kind of different context functionality.

Posted **5 years ago** by XINGFUCODER (LEVEL 14)

Reply

MDWHEELE

mdwheele • 5 years ago

@faisal_arbain @codeatbusiness Here's something to munch on that might help clear up confusion:

When we're writing software, we're "creating Domain Models within a Bounded Context". When you dig deeper and really start to grok the entirety of DDD, you eventually come to the understanding that it make no sense for a "Sub-domain" to be represented directly in the file-system. You don't code a Sub Domain, you code a Domain Model within a Bounded Context.

Domain and Sub-domain are ideas that reflect the state of the situation (or problem-space) as we understand it. The folks that operate in these domains / sub-domains have their own terminology for getting work done.

Bounded Context and Domain Models represent the implementation (or solution-space). Bounded Contexts simply make explicit the fact that different domains use different terminology. They may even have different interpretations of the same thing. Think about "student" versus "candidate". They both represent (or are projections) of the same thing, "a person". There may even be other terms for "a person". For instance (opening another can of worms), a **User** may authenticate to a system in the ~~bounded~~ context of Identity & Access Management. We just have three different viewpoints on the same unique thing.

- "Programming a domain" doesn't make sense because the domain already exists!
- "Programming a domain model" makes sense because we are creating a model BASED on our domain for the purposes of abstraction solving a particular problem.

- We break domains into smaller sub-domains when we discover differences in how parts of our organization (our domain) reason about similar concepts in different ways or use the same words, but mean different things.

Posted **5 years ago** by MDWHEELE (LEVEL 3)

1 Reply

MDWHEELE

mdwheele • 5 years ago

Regarding Eloquent (Active Record) and DDD!!! Be careful.

A significant technical strategy of DDD is to firmly separate the domain model from infrastructural concerns like persistence, presentation, etc. Active Record inherently couples a particular object to persistence.

THIS IS NOT TO SAY ACTIVE RECORD IS BAD BECAUSE IT IS NOT

There's just a trade-off. I've seen plenty of examples where Active Record solved the problem very nicely and the expressiveness of the Domain Model was not damaged. However, as you start getting more involved and applications begin to grow, it's something to be aware of and watch like a hawk. Unfortunately, people get passionate and make inflammatory claims regarding Active Record. It's just a tool; no different from a table-saw. A table-saw is very useful within the context of cutting wood, but fails to be as useful when I hurt myself on it...

One of my favorite things about the technical side of DDD is that I don't have to worry about frameworks, databases, templates, any of that stuff until the last possible moment (also happens to usually be when we have the most information to make the RIGHT decision).

Posted **5 years ago** by MDWHEELE (LEVEL 3)

Reply

XINGFUCODER

xingfucoder • 5 years ago

@mdwheeles, many thanks. Your explanations are very useful.

I [illegible]t as answered but if you and **@faisal_arbain** want to continue with this, we can return to this thread or open any similar to comment [illegible]DD and its applications.

I'm understanding some concepts righ now.

Posted **5 years ago** by XINGFUCODER (LEVEL 14)

Reply

FAISAL.ARBAIN@GMAIL.COM

faisal.arbain@gmail.com • 5 years ago

> "creating Domain Models within a Bounded Context"

@mdwheele can you clarify on `Bounded-context` in this discussion example. I not really sure whether the `domain` (Scholarship management) is considered as bounded context or it is sub-domain (student application and selection) .

Posted **5 years ago** by FAISAL.ARBAIN@GMAIL.COM (LEVEL 3)

Reply

XINGFUCODER

xingfucoder • 5 years ago

Yes **@mdwheel**, I know about Active Record and DDD, I think for this reason is very usefull the use of the Repository Pattern to avoid the storage dependency.

I always declare my domain entities interfaces within the domain, and leave in the Infraestructure Layer the implementations classes, correct me if I 'm wrong.

Posted **5 years ago** by XINGFUCODER (LEVEL 14)

Reply

FAISAL.ARBAIN@GMAIL.COM

faisal.arbain@gmail.com • 5 years ago

eel understood. Yes, usually i just code and assume i will retrieve `Candidate` from somewhere but not focusing on that yet. because i not y w what information i require from persistent.

in infrastructure layer than I code my repository.

How usually is your process/approach?

Posted **5 years ago** by FAISAL.ARBAIN@GMAIL.COM (LEVEL 3)

Reply

MDWHEELE

mdwheele • 5 years ago

Go check out dddinphp.org if you haven't already! There is a TON of information there *specific* to DDD applied in PHP projects.

This stuff is hard and nobody should expect to "get it" without some level of investment. I was thinking about how to better communicate some of this stuff and the fact is that you can't explain a lot of the reasons for different concepts in DDD without more complex domains. DDD as a modeling framework gives us a bunch of tools for working with both simple and complex domains.

I think if I had to "start over" and forget everything I knew about DDD before starting... I would recommend focusing **heavily** on understanding the more strategic modeling processes (Domain, Sub-domain, Bounded Context, Context Map) and **then** go back and try to write something **assuming** I was working with one giant domain and having to force myself into the friction to realize that ... "Hey, these folks are using the same words but they mean different things... maybe that means I'm working with multiple sub-domains here."

I mean, that's how the real world works, especially when you're learning something new. You get it wrong and then make it right; refactoring continuously.

Also, I'd warn newcomers to be on constant guard against directions of development of a solution that include the phrase: "Oh! We can **JUST** do ____ or ____". You don't take short-cuts in modeling if you're investing in DDD. It undermines the entire purpose. If a particular part of your application doesn't warrant this investment of modeling resources, **don't feel like you need to apply DDD everywhere**. It's a pattern. It's a tool; to codify business knowledge and expertise. As an example, an "IdentityAccess" bounded context could really be a set of thin remote facades on top of something like Laravel's authentication system or Sentry or whatever. Apply DDD in the areas of your system where you derive competitive advantage. Or, in the case that you're given resources to do otherwise, feel free to apply it wherever you want! :)

In addition to all this, I'll sum up by saying that DDD is a **VERY EFFICIENT** means to improving skills as a software developer overall. Think about the physics of it. Your brain is a network of physical connections between neurons that are constantly being built and rebuilt. It's a muscle that be worked and your "understanding" of any concept is really just the "right connections being made". Different thought processes hit different parts of the brain. DDD as a continuous process is hitting parts of your brain that:

1. Give you understanding of SOLID principles.
2. Sound object-oriented programming techniques.
3. Communication skills
4. Modeling abstract problems / concepts.

It's good stuff, but it's hard work. Keep at it!

Posted **5 years ago** by MDWHEELE (LEVEL 3)

1 Reply

FAISAL.ARBAIN@GMAIL.COM

faisal.arbain@gmail.com • 5 years ago

@mdwheele noted. Thanks for your time and advice :)

Posted **5 years ago** by FAISAL.ARBAIN@GMAIL.COM (LEVEL 3)

Reply

XINGFUCODER

xingfucoder • 5 years ago

Good night, returning to this thread.

Only one one more question. What option is better, shared Database between Bounded Contexts or a single Database for each bounded context?

For example, if we are to manage product data in two Bounded Contexts: - OrdersManagement BC - Inventory BC

What is the better way to store information of the Products for these two Bounded Contexts? If I use two Databases could be duplicating some data, but if I use one big database, when I manage the Products fromt he OrderManagement or the Inventory I will be using different data attributes and the schema will be also different.

I ome help.

Thanks in advanced.

Posted **5 years ago** by XINGFUCODER (LEVEL 14)

Reply

Load Newer Replies

Please sign in or create an account to participate in this conversation.