

# Validation in Domain Driven Design

Asked 2 years, 7 months ago   Active 2 years, 5 months ago   Viewed 4k times



7



Our team is starting a new project following Domain Driven Design (DDD). At the high level, we have an API on the top of our domain which enables a client to perform operations on the domain. One of the question I'm not very clear about is where do we perform validation on a certain property/ attribute in DDD.

Consider this example. Let us say, I have a below data contract/ DTO exposed by my API:



5



```
public class Person
{
    public string Email { get; set; }
    public string Name { get; set; }
}
```

Now, let us say we have business validation which prevents user to enter an invalid email address and restricts user to have name more than 50 characters.

To achieve this, I can see following three approaches:

In **Approach 1** , we do data validation only at the API (either through Data Annotation or Fluent Validation). I do not repeat the validation in my domain. Theoretically, this may mean my domain could go in an invalid state. But, since entry point (API) is being validated it is not possible in a real scenario.

**Join Stack Overflow** to learn, share knowledge, and build your career.

Sign up with email

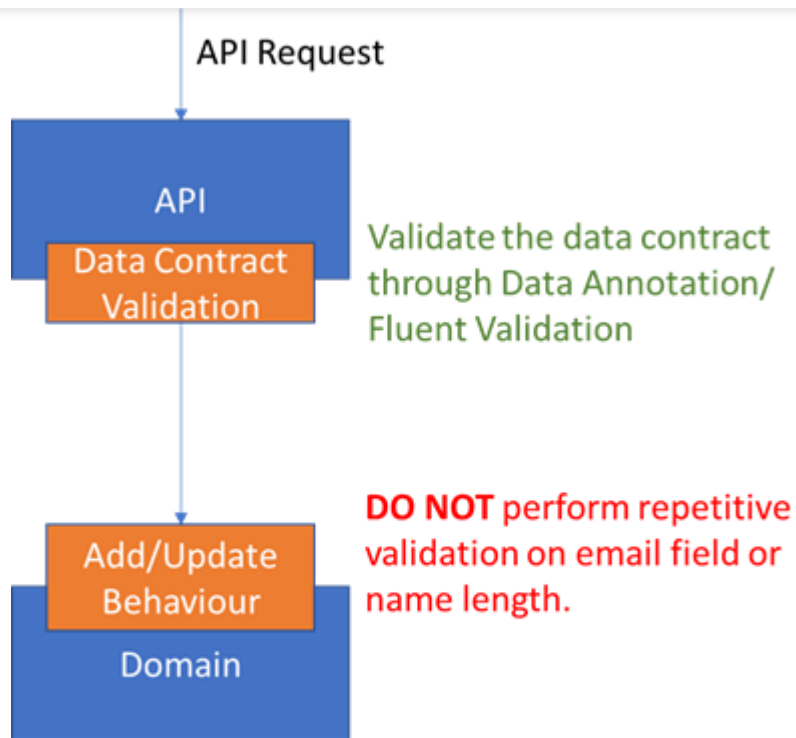


Sign up with Google

Sign up with GitHub

Sign up with Facebook






### Approach 1

In **Approach 2**, we do data validation at both API and in my domain. This approach helps us to completely remove coupling between my domain and API. The API can independently return a Bad request to the client. And since the domain performs the validation again, there is no chance of domain going to an invalid state. In this approach however, we violate DRY principle.

**Join Stack Overflow** to learn, share knowledge, and build your career.

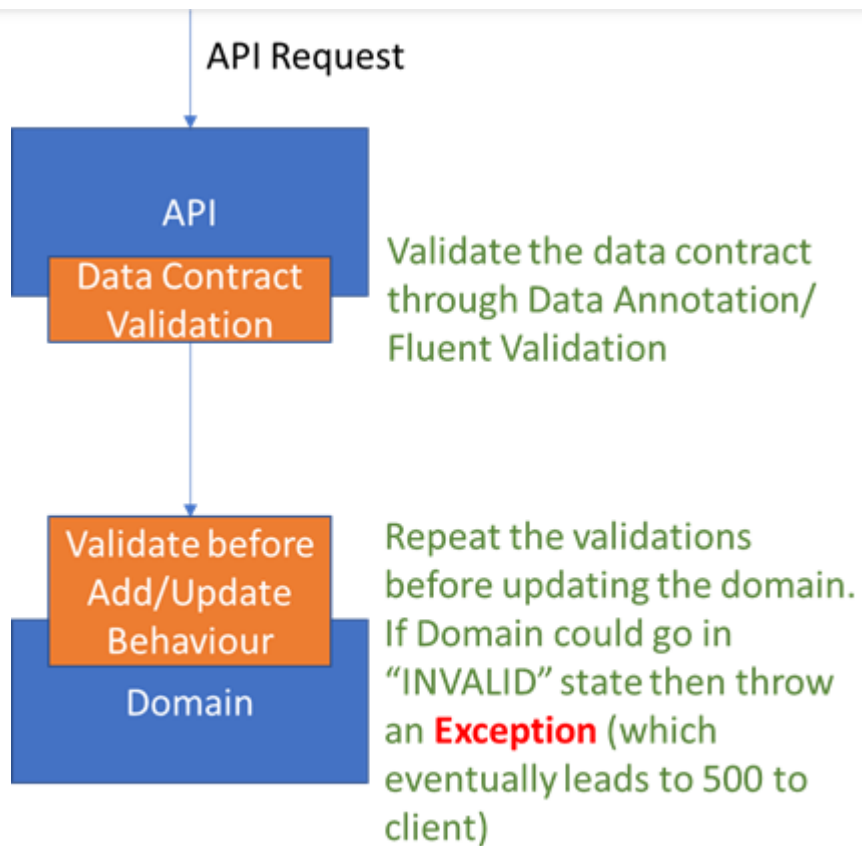
Sign up with email

 Sign up with Google

Sign up with GitHub

Sign up with Facebook





### Approach 2

In **Approach 3**, we do validation only at Domain and do not perform validation on DTO at API level. With this approach, while we are not repeating the validation, the domain cannot throw an exception when API call tries to put it in an invalid state. Rather, we would need to wrap that exception in some `Result` object. This would help the API to send an appropriate response to client (eg. Bad

Join **Stack Overflow** to learn, share knowledge, and build your career.

Sign up with email

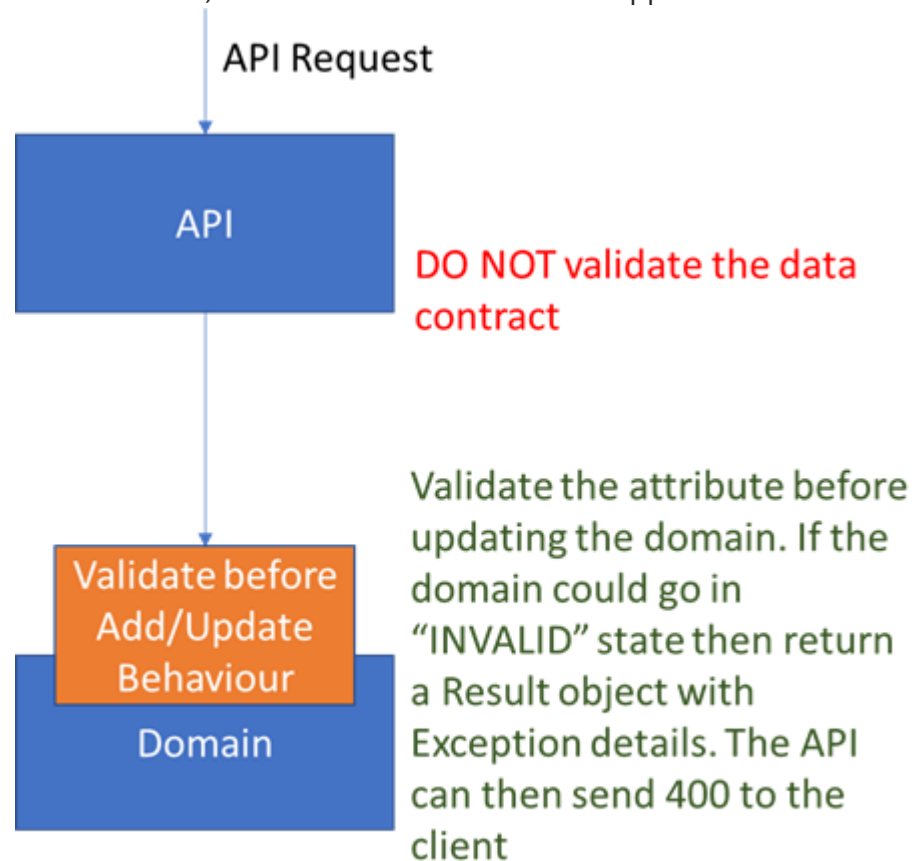
 Sign up with Google

Sign up with GitHub

Sign up with Facebook



request instead of Internal Server error). What I do not like about this approach is that I would prefer to throw a hard exception rather



### Approach 3


than putting a wrapper.

#### Ask

- Which approach makes the most sense and why?
- Where is the line between a business validation and a business rule? (Assuming the business rule exists in domain).
- Is there anything obvious which I'm missing here?

**Join Stack Overflow** to learn, share knowledge, and build your career.

Sign up with email

 Sign up with Google

Sign up with GitHub

Sign up with Facebook



[c#](#) [design-patterns](#) [domain-driven-design](#)[Share](#) [Improve this question](#) [Follow](#)

edited Oct 18 '18 at 23:40

asked Oct 18 '18 at 21:41

[Ankit Vijay](#)

2,742 2 26 40

input validation and in the domain business rules enforcement/validation doesn't violate DRY principle, since they fulfill two different tasks. Input validation is just a rough check that the schema fits in and data looks good before you even get as deep as to calling a domain service or even the aggregates themselves. For example some value of the input may be mandatory or optional depending on another value. You usually wouldn't check that at the input validation (WebApi here) but only in the domain if the content of the value looks okay and within the range – [Tseng](#) Oct 19 '18 at 10:01

- 1 In DDD you usually enforce values by treating them as value objects, not as literals. i.e. Email wouldn't be `string` in your domain but a value object named `Email` where the validation is enforced in the constructor. Once `Email` object is constructed its assumed to always be correct, because a violation should throw an exception and prevent the object to be created in the first place – [Tseng](#) Oct 19 '18 at 10:02

Hi @Tseng, I have seen people using `Email` as value object. But, do you think it is really necessary? Looking at @valueofunreason response, can we not categorize them just a "precondition"? – [Ankit Vijay](#) Oct 19 '18 at 21:31

Well, if email is really an email and not a random sequence of characters, then it needs to be a value type (even as going so far making it struct, if that's not a constraint of your persistence). It makes designing of your domain services and aggregates a lot easier, since the second you get an instance of `Email` you know its valid. And with `Mail` as value type is implicitly a precondition too, its invariant is enforced in the constructor. I'd disagree treating `Email` as just an identifier and it may not just be enough to check that at the input level – [Tseng](#) Oct 20 '18 at 8:55

For example, an email you may treat the RFC compliance at input level (is it well formatted? If not, just spot there). But within the domain you may have other validations on it. Is the domain required? Can it contain an IP instead of an domain. What if you want exclude specific domains, such as disposable mail hosts like `@yopmail.com` ? You can't and shouldn't checked blocked mails at validation level only if its part of your business rules to exclude such dubious sources – [Tseng](#) Oct 20 '18 at 8:58

## 4 Answers

[Active](#)[Oldest](#)[Votes](#)

I have gone through the answers provided by the experts here and after much deliberation within the team we have decided to

**Join Stack Overflow** to learn, share knowledge, and build your career.

[Sign up with email](#)[Sign up with Google](#)[Sign up with GitHub](#)[Sign up with Facebook](#)



but sit in between but our domain logic would not change. The API validations include but not limited to field length checks, regex checks etc.

- The domain validations are limited to business rules only. The field level validations like length check, regex etc which do not mean anything to business are kept out of business rule. If there is a business rule violation we simple throw a `DomainException` without caring about how consumer would handle it. In case of an API response, it would simple mean returning status code `500` back to the client.
- Since, we are now treating API validations and business rules as a separate concern, it does mean that there are chances of repetition of some validation rule. But we are OK with that as it makes things much simpler and easy to reason about. Since they are loosely coupled we can easily change API validation or business rule independently without impacting the other.
- There are few border line cases like `email` where one could have argued to have `valueObject` in our Domain. But we take this on case-by-case basis. For example, we do not see any value to have a separate `valueObject` for `email` field in our Domain at the moment. Hence, it is validated (regex check) only at the API level. If in future if business comes up with additional rules like restricting to a specific domain, then we may revisit our decision.

The above principles in mind have helped us to keep things simple and easy to reason about.

Share Improve this answer Follow

answered Nov 29 '18 at 22:10



Ankit Vijay

2,742 2 26 40

this seems to be what Microsoft calls [Two-step validation](#). @Ankit Vijay I'm wondering if, after this time, you still feed this the right approach? – [r.pedrosa](#) Sep 29 '20 at 19:03



3



Now, let us say we have business rule which prevents user to enter an invalid email address and restricts user to have name more than 50 characters.

Important things to notice in this example

**Join Stack Overflow** to learn, share knowledge, and build your career.

Sign up with email



Sign up with Google

Sign up with GitHub

Sign up with Facebook



- As far as the domain model is concerned, this data is opaque; you probably aren't ever going to manipulate it, or vary any computations based on the contents. As far as your business rules are concerned, they are just hash values that you are copying so that you can pass them along to something else (printing the name on an envelope, or sending an email).

Semantically, both of these values are basically just flavors of "identifier".

That being the case, the domain model doesn't care about the validation at all, except for such problems like not running out of memory. Your *data model* may care, if you have fixed length columns or something like that.

So this could very easily be one of the places where you care at the message boundary, but not within the domain itself.

But it's not a good proxy for the general question of where validation might live.

Contrast this case with something like a deposit amount -- it's a number, and you would reasonably expect to add/subtract it from other numbers, compare it to other numbers, and so on. There, you might look at something like `Integer.MAX`, and reasonably conclude that an attack/data entry error is so much more likely than a genuine use case that you will eliminate that option altogether.

Validation at the message boundary is primarily driven by the question: can you trust the source? If there is any doubt, then there is no doubt. (Deogun and Johnson on Domain Driven Security is a good starting point).

Largely, validation at the message boundary comes down to establishing that the sequence of bytes that you have received is actually compliant with the message schema; which can of course include limits on the range of allowed values. (Example: HTTP Responses include status codes, but you aren't required to pretend that a response with status code 777 is meant to improve your afternoon).

And so it's a perfectly reasonable thing to declare that the name field in a message be no more than 50 characters, and that the email address field in the message conforms to the definition of [addr\\_spec in RFC 5322](#).


And then at the boundary you make sure the bytes you get actually satisfy the message constraints, and pass it long if it does.

But within the domain model? if you don't need to make assumptions about the data, then done. "The application said these are bytes? Good enough for me!"

---

**Join Stack Overflow** to learn, share knowledge, and build your career.

Sign up with email

 Sign up with Google

Sign up with GitHub

Sign up with Facebook



(Again, contrast with `amount` -- the domain model has a lot of interest in detecting violations before it starts moving money around indiscriminately).

Share Improve this answer Follow

answered Oct 18 '18 at 23:07



VoiceOfUnreason

40.1k 3 34 73

Hi @voiceofunreason, thanks for the detailed explanation. One of the most important key-take away I take from here is "Precondition". – Ankit Vijay  
Oct 19 '18 at 21:21



Theoretically, this may mean my domain could go in an invalid state.

2



I don't think a Name longer than 50 characters would mean an invalid domain state... The domain would still be perfectly functioning.



You have to differentiate between input validation (does the data fit in the technical slots?) and domain invariants. Some things you can validate at API level and don't matter to the domain, others have to be checked by loading domain data and thus aren't easily accessible to outer layers.

They are really two (or more) different sets of rules, more largely disconnected than you'd think.

*TL;DR - there's no hard and fast answer. Try to characterize things deeper than just "business validation" and choose Approach 1, 2 or 3 wisely **depending on type of rule**.*

Share Improve this answer Follow

edited Oct 19 '18 at 7:56

answered Oct 19 '18 at 7:50



guillaume31

12.7k 28 43

**Join Stack Overflow** to learn, share knowledge, and build your career.

Sign up with email



Sign up with Google

Sign up with GitHub

Sign up with Facebook







that cannot be validated without accessing domain.

NOTE: I follow the approach of validating the operations, not the data. A given data can be valid for performing an operation but not other.

Where is the line between a business validation and a business rule? (Assuming the business rule exists in domain).

I think you are misunderstanding/mixing concepts. A concept is "business rules", and another concept is "validation". Validation is the process we do for checking whether the business rules are satisfied or not.

Share Improve this answer Follow

answered Oct 19 '18 at 13:00



[choquero70](#)

3,292 2 24 40

**Join Stack Overflow** to learn, share knowledge, and build your career.

Sign up with email



Sign up with Google

Sign up with GitHub

Sign up with Facebook

