Validation is the process of mapping a set onto its subset.

3 building blocks of good validation technique

{

Always valid domain

All validation rules are part of domain layer

Validation is parsing

}

\*\* Don’t put simplest validation rules to the domain layer e.g null or empty or min and max string checking \*\*

Invariants are the same as input validation

{

Invariants define the domain model

Invariants are the reason validation exists

Invariants are what differentiates valid and invalid domain models

Invariants are what differentiates (Invalid entity) from (valid entity)

}

Data validation is the same as business rules validations. They both represent your application invariants.

The only reason why people segregate them is to rationalize having two ways of handling input validation.

Simple validations are normally tackled with framework e.g. FluentValidation, while complex validations by controllers.

All validations, no matter how simple, are part of the domain layer (come from business requirements). The only distinction here is some validation rule are simpler than the others.

Max length constraints are not purely technical limitations.

===============================================================================

Validation rules are invariants

{

Invariants dictate what is and what isn't a valid domain object

}

===============================================================================

Object creation and validation can't be separated, separation leads to code duplication.

Validation is parsing (Parsing = Validation + Transformation).

All operations that involve transformation and validation should be treated as parsers. and should be implemented as one method.

===============================================================================

1 Invariant = Primitive type

>1 Invariants = Value Object

===============================================================================

===============================================================================

Error messages should not be handled by the domain layer

A pure domain model is a model that doesn't reach out to out-of-process dependencies.

===============================================================================

===============================================================================

DDD Trilemma

{

Encapsulation: All domain logic is in the domain layer

Purity: No out-of-process dependencies

Performance: No unnecessary calls to out-of-process dependencies

}

Domain layer should only be responsible for the domain logic, do not mix it with other responsibilities (e.g injecting out-of-process dependencies)

