Domain-Driven Design
A pragmatic approach

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Why DDD? Because it is a way to get rid of:

Late delivery

Stripped down functionality

Not what customers need

Difficult to change

It's hard to fix bugs and it's time consuming

Functionality is scattered over many places

and many other reasons!!

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

About Me

I have more than 24 years of experience in software development, being involved in various projects covering desktop, client/server, web, cloud, and mobile applications using mainly Microsoft tools & technologies, but also Apple and open-source.

I'm a Certified Scrum Master and a certified APMG Agile Project Manager. Since 2003, I'm holding a Ph.D. degree in Industrial Engineering.

You can find more about me on LinkedIn:
https://www.linkedin.com/in/eduardghergu/

Introducing Domain-Driven Design (DDD)

1. What is DDD?

2. Why do we need DDD? Any drawbacks?

3. DDD Building Blocks & Patterns

4. Conclusions. Q & A

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

What is DDD?

It is a design methodology...

… defined by Eric Evans in his seminal book Domain-Driven Design: Tackling Complexity in the Heart of Software (Addison-Wesley Professional, 2003).

… is an approach to software development that enables teams to effectively manage the construction and maintenance of software for complex problem domains.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

What is DDD? - cont.

It is a design methodology...

… that emphasize the need to understand and match the business domain for the application that is going to be implemented.

As software developers, we need to “translate” the real world business domain into a form that can be used to create the application’s code - a domain model.

A domain model is an essential part of our software design. It will also be a communication vehicle between domain experts, and the software development team.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

What is DDD? - cont.

It is a design methodology...

… that favors the usage of Separation of Concern (SoC) design principle.

Usually, a complex problem can be decomposed into smaller problems that can be solved much easier; similarly, a complex business domain can be “divided” into sub-domains with clear roles and purposes easier to be modeled - an application of SoC.

However, not all subdomains are equally important; for the important ones (the core), it is required to invest more time and effort to understand them correctly.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

What is DDD? - summary

Long story short… The main goals are:

- placing the project's primary focus on the business domain and its logic

- basing complex designs on a model of the domain

- initiating a creative collaboration between technical and domain experts to iteratively refine a conceptual model that addresses particular domain problems

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Why do we need DDD?

Most important advantages:

- **Eases Communication** - helps the teams create a common model. The teams from the business side and from the developer's side can then use this model to communicate the business requirements, the data entities, and the process models.

- **Provides Flexibility** - this implies that almost everything in the domain model is based on an object and therefore will be modular and encapsulated, enabling the system to be changed and improved regularly and continuously.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Why do we need DDD?

Most important advantages - cont.:

- **Improved Patterns** - gives software developers the principles and patterns to solve tough problems in software and, at times, in business.

- **Reduced Risk of Misunderstandings** - requirements are always explained from a domain perspective. Conceptualizing the software system in terms of the business domain reduces the risk of misunderstandings between the domain experts and the development team.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Why do we need DDD?

There are also disadvantages:

- Additional Efforts Required - the main disadvantage of introducing DDD in software development is the additional effort and time required to create a substantial model of the business domain before positive effects on the development process become apparent.

- Requires Domain Experts - if not available ‘in-house’, often, they are expensive to hire since they hold valuable knowledge.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Why do we need DDD?

There are also disadvantages - cont:

- **Suitable for Complex Applications** - DDD was designed to simplify complexity. It is a great approach to software development if there is a need to simplify, but for simple applications, using the DDD is not worth the effort.

- **High Encapsulation can be an issue** - a high level of isolation and encapsulation in domain model may present a challenge for business domain experts; that's why is important to avoid the technical details on the design phase.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

DDDs’ Building Blocks

The main DDD concepts (or building blocks) are [3]:

- **Ubiquitous Language**

- **Bounded Context / Context Maps**

- **Domain - A sphere of knowledge (ontology), influence, or activity. The subject area to which the user applies a program is the domain of the software;**

- **Model - A system of abstractions that describes selected aspects of a domain and can be used to solve problems related to that domain;**

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

DDDs’ Building Blocks - cont.

Ubiquitous Language:

To allow fluent sharing of knowledge, DDD calls for the cultivation of a shared, business-oriented language: Ubiquitous Language.

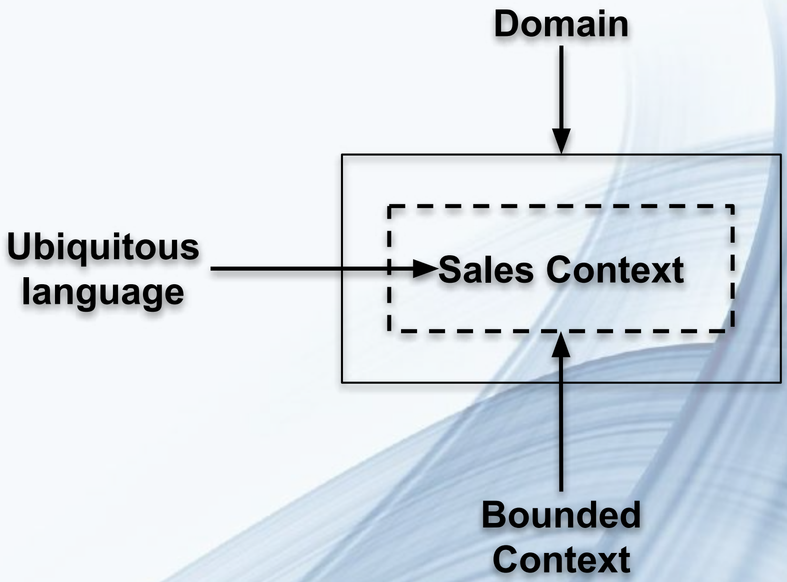
This language should resemble the business domain and its terms, entities, and processes - the communication vehicle mentioned above.

Defining a Ubiquitous Language is not a trivial thing to do because the defined terms can have different meanings in different contexts. To overcome this hurdle, another DDD concept is employed: Bounded Context.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

DDDs' Building Blocks - cont.

Ubiquitous Language:



See ref. [6]

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

DDDs' Building Blocks - cont.

Bounded Context:

As the name suggest, the idea here is to define a boundary inside of which a specific Ubiquitous Language can be defined.

Outside of it, the Ubiquitous Language's terms may have different meanings.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

DDDs' Building Blocks - cont.

Context Maps:

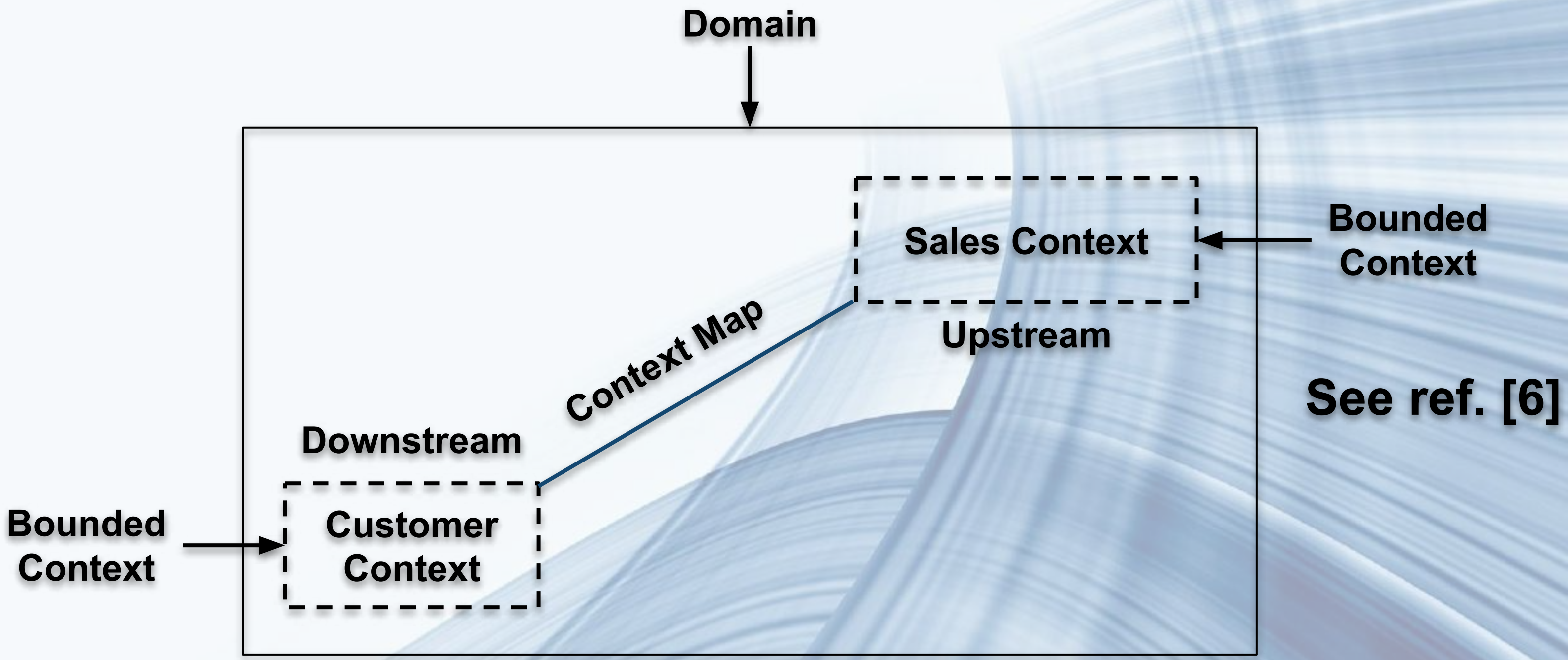
As the application model evolves, the need to create a relationship between the Bounded Contexts will become obvious.

For this purpose we will use what's called Context Maps.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

DDDs’ Building Blocks - cont.

Context Maps:



See ref. [6]

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

DDDs’ Building Blocks - cont.

Domain [7]:

DDD revolves around the idea of solving the problems faced by an organization through code. This is achieved by focusing the investment of resources at the heart of the business logic of the application.

The domain is the problem that we’ll need to solve.

The domain is the world of the business you are working with and the problems they want to solve. This will typically involve rules, processes, and existing systems that need to be integrated as part of your solution.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

DDDs’ Building Blocks - cont.

Domain - cont.:

The domain is the ideas, knowledge, and data of the problem you are trying to solve. Most businesses will have terms that have specific meaning within the context of their organization. They will also likely have metrics, goals, and objectives that are unique to their business.

All of the knowledge around the company and how it operates will form the ***domain*** of your Domain-Driven Designed project.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

DDDs’ Building Blocks - cont.

Model [7]:

The Model of a Domain-Driven Designed project is your solution to the problem.

The Model usually represents an aspect of reality or something of interest. The Model is also often a simplification of the bigger picture and so the important aspects of the solution are concentrated on whilst everything else is ignored.

This means your Model should be focused on the knowledge around a specific problem that is simplified and structured to provide a solution.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

DDDs' Patterns

- **Entity** - an object that is not defined by its attributes, but rather by its identity. Its state can change over time.
- **Value Object** - an object that is defined only by its state, and has no identity. They should be treated as immutable.
- **Aggregate** - a collection of entities or value objects that are related to each other through an Aggregate Root object
- **Aggregate Root** - owns an Aggregate (it's the main object) and serves as a gateway for all the changes within the Aggregate

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

DDDs' Patterns - cont.

- **Domain Event** - An immutable domain object that defines an event (something that already happened) about which other parts of the same domain could be interested in reacting to it.
- **Domain Service** - When an operation does not conceptually belong to any object it can be implemented in a service.
- **Repository** - The methods for retrieving domain objects should be delegated to a specialized **Repository** object.
- **Factory** - The methods for creating domain objects should delegate to a specialized Factory object such that alternative implementations may be easily interchanged.

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Code sample - C# [9]

[Customer class]

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Conclusions

- **Patterns: Domain-Driven Design gives software developers the principles and patterns to solve tough problems in software and, at times, in business**

- **Business Logic: Domain-Driven Design creates business logic by explaining requirements from a domain perspective.**

- **Successful History: Domain-Driven Design has a history of success with complex projects, aligning with the experience of software developers and software applications developed.**

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Q & A

For any inquiries or requests:
eduard.ghergu@professional-programmer.com

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

Thank you!
See you at the next webinar!

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com

References

1. https://www.infoq.com/minibooks/domain-driven-design-quickly/
2. https://vladikk.com/2016/04/05/tackling-complexity-ddd/
3. https://en.wikipedia.org/wiki/Domain-driven_design
4. https://www.jamesmichaelhickey.com/clean-architecture/
5. https://blog.knoldus.com/is-shifting-to-domain-driven-design-worth-your-efforts/
6. https://thedomaindrivendesign.io/
7. https://www.culttt.com/2014/11/12/domain-model-domain-driven-design/
8. https://martinfowler.com/bliki/BoundedContext.html
9. https://github.com/zkavtaskin/Domain-Driven-Design-Example

Eduard Ghergu, Eng., PhD.
www.professional-programmer.com