ISSN 21-76-8498

DESENVOLVIMENTO DE CAMADAS UTILIZANDO DDD

Giovanni Luis Baroni FIRMINO[1]
Ana Paula Ambrósio ZANELATO[2]

RESUMO: O assunto desenvolvido neste artigo será sobre o desenvolvimento em camadas usando um modelo de estrutura chamado DDD ou (Domain Drive Design), que mostrada sua importância, pontos positivos de desenvolver utilizando o DDD.

Palavras-chave: Sistema. Desenvolvimento. Programação. Desenvolvedores. Código-Fonte.

1 INTRODUÇÃO

Atualmente a tecnologia está presente no nosso dia a dia, em todos os dispositivos e aparelhos eletrônicos. Vivemos uma fase que marcou uma época, o crescimento exponencial do uso de equipamentos eletrônicos. Para que os aparelhos funcionem devidamente, é necessário que haja uma programação lógica por trás dos aparelhos, desta forma, o mercado e as tecnologias deste contexto são denominados como desenvolvimento ou programação e estão em constante crescimento. Existem várias formas de se desenvolver os softwares, utilizando diversas linguagens de programação, diversas técnicas e abordagens diferentes para que o desenvolvimento fique mais claro e fácil de ser implementado, uma dessas Abordagens de desenvolvimento de Software é o DDD (**Domain- Drive-Design**) que iremos explorar neste artigo.

Esta abordagem consiste em aprimorar, padronizar e facilitar a maneira com que a programação é realizada. Com o desenvolvimento do software de grande porte e sendo realizado por diversos desenvolvedores, o código-fonte acaba ficando desordenado e repleto de ajustes, caso não seja seguido um padrão, pois cada desenvolvedor tem conhecimento de suas tarefas e utiliza sua própria lógica de

[1] Discente do 4º ano do curso de Sistemas de Informação das Faculdades Integradas "Antonio Eufrásio de Toledo" de Presidente Prudente.
[2] Linguagens e Tecnologias de Programação III – Disciplina Ministrada pela professora Ana Paula Ambrósio Zanelato.

programação. Isso acabou se tornando um grande problema, pois era impossível conseguir dar manutenção nos sistemas, pois o código-fonte era confuso.

Com o código confuso, os erros surgiam no sistema, e por consequência não conseguiam atender seus clientes. Um dos principais objetivos da utilização de softwares deve ser para facilitar a vida de seus usuários de alguma maneira. E caso este objetivo não seja alcançado, o projeto acaba se tornando inviável para a empresa, podendo até mesmo ser cancelado. Portanto, no desenvolvimento do software, é importante compreender as regras de negócio, pois nesta etapa, são captados os requisitos que o sistema deve atender e também estabelecer um padrão de desenvolvimento.

Assim, o uso de técnicas tornou-se indispensável para o desenvolvimento dos sistemas. O DDD utiliza um conceito que está se tornando muito popular devido os benefícios que ele traz para as empresas de desenvolvimento. Ele reúne um conjunto de conceitos, técnicas e princípios cujo o foco da programação está no Domínio (**Domain Model**) ou Modelo de Domínio. Neste contexto o Domínio no qual nos referimos é o que o programa de software modela e o problema no qual ele se propõe a resolver. Com esta mudança, os desenvolvedores que estão acostumados com um estilo de programação, sentem a necessidade de aprender tudo de novo, pois os benefícios a longo prazo são enormes.

2. DESENVOLVIMENTO

2.1 DDD (Domain Driven Design) - Camadas

Domain Driven Design significa Projeto Orientado a Domínio. Ele veio do título do livro escrito por Eric Evans. O livro de Evans é um grande catálogo de **Padrões**, baseados em experiências do autor ao longo de mais de 20 anos desenvolvendo software utilizando técnicas de Orientação a Objetos.

DDD pode ser visto por alguns como a volta da orientação a objetos. Quando se fala em Orientação a Objetos pensa-se logo em classes, heranças, polimorfismo, encapsulamento. Mas a essência da Orientação a Objetos é composta por alguns deles, um deles é o **Alinhamento Código com o Negócio**: “O contato

dos desenvolvedores com os especialistas do domínio é algo essencial quando se faz DDD (o pessoal de métodos ágeis já sabe disso faz tempo)". (CUKIER, s.p,2010).

Existem vários benefícios que a Orientação a objetos traz para os desenvolvedores uma delas é o **Favorecimento de Código**: "Os blocos de construção, que veremos adiante, facilitam aproveitar um mesmo conceito de domínio ou um mesmo código em vários lugares". (CUKIER, s.p,2010).

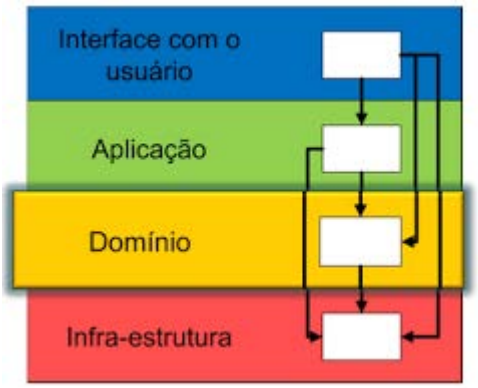
Podemos encontrar um outro benefício é o **Mínimo Acoplamento**: **"**Com um modelo bem feito, organizado, as várias partes de um sistema interagem sem que haja muita dependência entre módulos ou classes de objetos de conceitos distintos." (CUKIER, s.p,2010).

O DDD é independente de Tecnologia, ou seja ele não necessita estar ligado as tecnologias mais atuais.

> **Independência da Tecnologia:** DDD não foca em tecnologia, mas sim em entender as regras de negócio e como elas devem estar refletidas no código e no modelo de domínio. Não que a tecnologia usada não seja importante, mas essa não é uma preocupação de DDD. (CUKIER, s.p,2010).

O DDD depende de um desenvolvimento em camadas. Isso possibilita encapsular as regras de negócio complexas no **domínio**. Define quatro camadas principais, conforme imagem abaixo:

Figura 1: Estrutura DDD.



(CUKIER, s.p,2010).

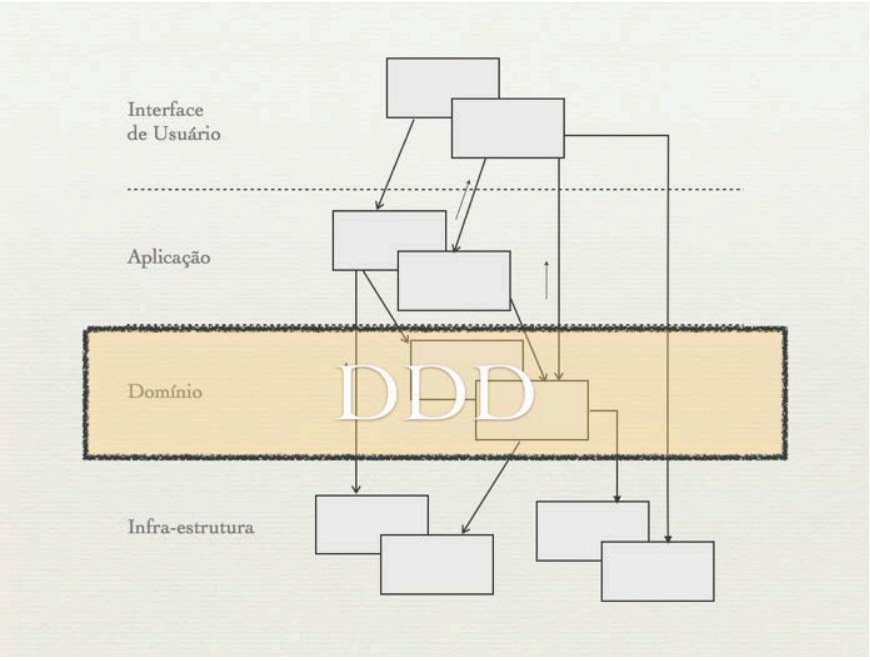
É possível manter o mínimo acoplamento entre as camadas usando recursos como Injeção de Dependência/Inversão de Controle por exemplo. Um dos benefícios é obter independência ou maior flexibilidade de tecnologias.

> Na camada de Interface com o usuário: posso usar Asp.net MVC, Windows Form; Web Forms, Tendo possibilidade de mudança sem afetar/refazer as demais camadas do software.
> Já na camada de Infraestrutura: posso usar Entity Framework, NHibernate, SQL Server, Oracle, também com possibilidade de mudar a tecnologia sem afetar/refazer as demais camadas do software. (Cosme, s.p,2012).

2.1.2 Domínio

Esta camada representa os conceitos, regras e lógicas de negócio, onde encontramos todas as informações que são transmitidas ao sistema sobre como a empresa trabalha, suas regras, suas normas, o que o sistema deve permitir e o que não deve permitir. Todo o foco de DDD está nessa camada.

Figura 2: Estrutura Domínio.



(CUKIER, s.p,2010).

2.1.3 Aplicação

Dentro da modelagem utilizando DDD, encontramos a camada de aplicação ela é responsável para fazer as validações que vem da camada de domínio.

> A camada de aplicação é a camada responsável por tratar trabalhar os objetos da camada de domínio. Nela estão raros e possíveis trabalhos sobre informações da camada de aplicação sem que se execute qualquer tipo de regra de negócio, por mais simples que isso seja. Utilizando exemplos de aplicações Web, estes seriam os seus Servlets ou Commands ou Actions. (ALLEMAND, s.p,2009).

2.1.4 Infraestrutura

Fornece recursos técnicos que darão suporte às camadas superiores. São normalmente as partes de um sistema responsáveis por persistência de dados, conexões com bancos de dados, envio de mensagens por redes, gravação e leitura de discos. Nesta camada é a última camada do modelo representando a comunicação direta com o Banco de Dados.

2.1.5 Interface do usuário

Parte responsável pela exibição de informações do sistema ao usuário e também por interpretar comandos do usuário. É na interface do usuário que encontramos os botões, caixas de texto, está é a única parte visível pelo usuário, pois não se trata somente de código e sim dos objetos que sejam exibidos na tela.

2.2 Linguagem Ubíqua

A linguagem Ubíqua é muito indicada para se trabalhar juntamente com o programador, a Linguagem Ubíqua pode ser compreendida:

> Por **linguagem Ubíqua** (linguagem comum) entende-se que ao trabalhar com **DDD** devemos conversar usando uma mesma língua, em um único modelo, de forma que o mesmo seja compreendido pelo cliente, analista, projetista, desenhista, testador, gerente, etc. nesta linguagem, que seria a linguagem usada no dia a dia.
> (LIVRO, s.p, s.d)

Todos os envolvidos na criação do software devem pelo menos entender um pouco do código que estão desenvolvendo:

> Qualquer pessoa técnica contribuindo para o modelo deve programar, pelo menos tocar no código, independente do papel desempenhado no projeto. Um responsável por mudar o código deve sempre aprender a expressar o modelo através do código. Todo desenvolvedor deve estar envolvido na discussão sobre o modelo e ter contato com os especialistas do domínio.
> (EVANS, s.p, s.d).

2.3 Entidades

Entidades são objetos que possuem um identificador. Esse identificador deve ser único para todo o modelo. Sua principal função é distinguir um objeto de todos os outros do modelo que possuem identidade.

> Entidades são objetos importantes no Domain Model e eles devem ser considerados desde o começo da modelagem do sistema. Também é importante determinar se um objeto precisar ser uma entidade ou não, pois isso implica em certos custos para o sistema discutidos mais adiante nessa mesma seção. (SANTOS, s.p, s.d).

2.4 Objeto de Valor

Existem implicações no desempenho do sistema fazer todos os objetos do domínio, entidades. O processo para se criar um identificador em um objeto não é tão simples. Em muitos casos estamos interessados apenas nos valores dos atributos de um objeto, independentemente de qual objeto seja. Um objeto que é usado para descrever certos aspectos do domínio, mas que não possui um identificador é chamado de Objeto de Valor. Uma das características dos objetos de valores é que eles não são modificados, se precisamos de valores diferentes, criamos outro objeto.

> É recomendável que Objeto de Valor sejam imutáveis, ou seja, que eles sejam criados com os valores passados no construtor e esses valores nunca mais sejam modificados durante sua existência. Quando se precisar de um valor diferente para os atributos, simplesmente se cria outro Objeto de Valor. Isso simplifica bastante o design da aplicação. (SANTOS, s.p, 2008).

Como Values Objects são objetos imutáveis e não possuem identificador, eles podem ser compartilhados entre objetos, até mesmos objetos de outras camadas, pois não se corre o risco que outros objetos alterem o seu valor e gerem uma inconsistência, objetos de valores podem conter outros objetos de valores, bem como referência para Entidades.

2.5 Módulos

Para aplicações complexas, o modelo tende a ficar muito grande, tornando-se difícil de ser compreendido como um todo. Por esse motivo, é preciso reorganizar o modelo em módulos.

Existem várias formas de coesão entre elementos de um módulo. As duas formas principais são a coesão na comunicação e a coesão funcional.

> A coesão na comunicação é atingida quando partes do módulo operam sobre os mesmos dados, já a coesão funcional é alcançada quando todas

> as partes de um módulo trabalham juntas para realizar uma tarefa bem definida. (SANTOS, s.p, 2008).

Essa última é considerada o melhor tipo de coesão, pois separa melhor as funcionalidades de cada módulo. Módulos, como camadas, deveriam ter interfaces bem definidas que seriam acessadas por outros módulos. Em vez de chamar três objetos de um módulo, é melhor acessar uma interface, porque isso reduz o acoplamento.

2.6 Repositórios

O propósito do uso de um Repositório é encapsular toda a lógica de programação necessária para obter uma referência de objeto, Os objetos de domínio não têm que se preocupar com a infraestrutura necessária para obter outro objeto do domínio eles iram apenas pegá-los do repositório, como vantagem o modelo manterá sua claridade e foco, ficando desacoplado da infraestrutura necessária para persistir os objetos. Podemos definir a utilização de um repositório como um lugar onde os objetos se encontram.

> Um repositório age como um lugar de armazenamento para objetos acessíveis globalmente, através de uma interface global bem conhecida. Um Repositório pode também incluir um padrão de estratégia. Ele pode acessar um armazenamento persistente baseado em uma estratégia específica, permitindo a ele usar um local diferente de armazenamento para diferentes tipos de objetos. (SANTOS, s.p, 2008).

É aconselhado que para cada tipo de objeto que precise ser acessado globalmente, se crie um repositório que providencie a ilusão de acessar objetos em coleções, como se tivessem em memória.

> Um repositório deve conter métodos com todas as operações que devem ser realizadas com objetos persistidos, por exemplo, incluir, apagar, buscar pelo identificador, entre outras. Um repositório pode ter detalhes sobre o

> acesso à infraestrutura, mas sua interface deve ser simples. (SANTOS, s.p, 2008).

Um repositório pode manter objetos em cache localmente, mas o mais frequente é precisar obter esses objetos de algum meio persistente.

2.7 Serviços

Uns dos princípios da programação orientada a objetos é o encapsulamento, no qual objetos devem ser responsáveis por manipular seus próprios dados e manter a consistência deles. Mas, às vezes, existem algumas ações no domínio que não pertencem a nenhum objeto do domínio em particular, essas ações são importantes e não podem ser ignoradas.

> É necessário criar alguma coisa para conter essas operações. Usando uma linguagem orientada a objetos, como não se pode deixar ações soltas, é necessário criar objetos que possuam essas operações. Esses objetos que possuem operações que não dizem respeito a um único objeto no domínio são chamados no DDD de **serviços**. (SANTOS, s.p, 2008).

Se a incorporação de uma funcionalidade em uma entidade ou objeto de valor criar uma confusão se essa funcionalidade pertence mesmo a o objeto em questão, é uma boa indicação que provavelmente essa funcionalidade deveria estar incorporada a um serviço, podemos citar três características básicas de um serviço.

> A operação contida em um serviço faz referência a um conceito do domínio que naturalmente não pertence a uma entidade ou a um objeto de valor.
> A operação necessariamente refere-se a outro objeto no domínio.
> A operação não guarda o estado dela. (SANTOS, s.p, 2008).

3 CONCLUSÃO

DDD é uma técnica de desenvolvimento muito bem consolidada, e muitas empresas grandes e também pequenas utilizam esse método, como dito anteriormente existem muitas vantagens em se usar o DDD, tornando o desenvolvimento mais fácil e prático após a estruturação do sistema. Com certeza existirão técnicas muitos melhores futuramente, mas essa persistirá, e tornará o trabalho dos desenvolvedores muito mais gratificante.

REFERÊNCIAS BIBLIOGRÁFICAS

CAMADAS e o DDD. **Thiago Holder**.Mar.2010 Disponível em:<http://thiagoholder.wordpress.com/2010/03/21/camadas-e-o-ddd/>Acesso em: 7 maio 2014.

DOMAIN Driven Design(DDD). **Renan Cosme**. Nov. 2012 Disponível em:<http://ehbomcompartilhar.blogspot.com.br/2012/11/domain-driven-design-ddd.html>Acesso em: 7 maio 2014.

LIVRO - Resumo sobre Domain-Driven Design de Eric Evans. **José Carlos Macoratti**. Disponível em:<http://www.macoratti.net/11/05/ddd_liv1.htm> Acesso em: 7 maio 2014.

DDD – Introdução a Domain Driven Design. **Daniel Cukier**. Jul.2010. Disponível em:http://www.agileandart.com/2010/07/16/ddd-introducao-a-domain-driven-design/> Acesso em: 7 maio 2014.

ANALÍSE da Utilização de Padrões no Desenvolvimento de Softwares em Camadas. **Jadson José dos Santos**. 2008. Disponível em:<http://jadsonjs.files.wordpress.com/2008/03/artigo-padroes-de-projeto-padroes-do-modelo.pdf> Acesso em: 7 maio 2014.

DDD, TDD, BDD, Afinal o que são essas siglas. **Eduardo Pires**. Jun.2012. Disponível em:<http://eduardopires.net.br/2012/06/ddd-tdd-bdd/> Acesso em: 7 maio 2014.

TÉCNICAS de Desenvolvimento em .NET. **Cadu**. Disponível em: <http://www.devmedia.com.br/tecnicas-de-desenvolvimento-em-net-revista-net-magazine-94/23666> Acesso em: 8 maio 2014.

DDD – Camadas e Comunicação. **Rodrigo Allemand**. Feb.2009. Disponível em: <http://blog.rodrigoallemand.com.br/?p=120> Acesso em: 8 maio 2014.