ABORDAGEM DIRIGIDA AO DOMÍNIO APLICADO NA ARQUITETURA DE SISTEMAS WEB

DOMAIN DRIVEN APPROACH APPLIED TO THE WEB SYSTEM ARCHITECTURE

Ramon Santos Vieira
Universidade Salvador/Brasil
ramonsv89@gmail.com

Uedson Reis
Universidade Salvador/Brasil
uedson.reis@pro.unifacs.br

Resumo: **Esta pesquisa propõe especificar um modelo baseado em MDA (Arquitetura Dirigida a Modelos) para o desenvolvimento de sistemas Web. Para atingir este objetivo realizou-se estudo de procedimentos e técnicas que utilizam o MDA, metodologias ágeis e o DDD – Desenvolvimento Orientado a Domínio. Este modelo visa otimizar o desenvolvimento de software reduzindo impactos, como atrasos e altos custos, para o projeto.**

Palavras-Chave: **MDA; DDD; Metodologias Ágeis**

Abstract: **This research proposes to specify a model based on MDA (Model Driven Architecture) for the development of Web systems. To achieve this goal, we carried out the study procedures and techniques using the MDA, agile methodologies and DDD - Domain Driven Development. This model aims optimize the software development reducing impacts, like daley's and high costs, for the project.**

Keywords: **MDA; DDD; Agile Methodologies.**

I. INTRODUÇÃO

A época atual caracteriza-se pela contínua evolução da tecnologia e a exigência, cada vez maior, de se desenvolver grandes sistemas para atender às necessidades de uma sociedade em que o novo rapidamente torna-se obsoleto. Assim, a partir da urgência de agilizar o processo de desenvolvimento de software frente ao rápido crescimento da demanda, foram criadas técnicas e metodologias que visam redução de tempo e custo, aumento da produtividade, manutenibilidade, usabilidade e qualidade do software [2].

Dentre as técnicas, metodologias e padrões existentes na área de desenvolvimento de sistemas, utilizou-se neste artigo o MDA – Arquitetura Dirigida a Modelos (Model-Driven Architecture), o DDD – Desenvolvimento Orientado a Domínio e as metodologias ágeis, dentre elas o Scrum e TDD – Desenvolvimento orientado a testes, que em conjunto embasaram a criação do Modelo de Domínio Orientado a Sprints proposto.

O MDA é uma ferramenta de automação de código que possui uma abordagem neutra em relação à tecnologia utilizada, buscando um alto nível de abstração com a possibilidade de mudança de tecnologia com baixo impacto no desenvolvimento, separando a lógica de negócio, plataforma e tecnologia a ser adotada [10].

O MDA tem como características principais: criar um modelo de alto nível de abstração, que independe da plataforma que será usada, ou seja um modelo que possua portabilidade, reutilização e interoperabilidade com foco em automatizar o máximo possível este processo de desenvolvimento de software [10]. O que justifica este trabalho, uma vez que, através do uso do MDA, pode-se obter uma redução de tempo de desenvolvimento de novas aplicações bem como um aumento da qualidade do software e sua adaptabilidade.

Esta pesquisa teve como objetivo geral propor um modelo baseado no MDA para desenvolvimento de sistemas web. Este se desdobra em objetivos específicos, que são: analisar aplicações do MDA em sistemas de modo geral; identificar melhorias, boas práticas e abordagens que possam ser agregadas e especificar e desenhar o novo modelo baseado no MDA.

Para realização deste trabalho, a metodologia adotada envolveu as atividades de pesquisa sobre as temáticas MDA, DDD, Arquitetura de Sistemas Web, Scrum e TDD. Exemplos de aplicações dos conceitos de MDA e DDD, foram igualmente pesquisados e analisados. O Modelo de Domínio Orientado a Sprints foi assim proposto com base em análises comparativas entre os trabalhos correlatos que foram estudados durante a pesquisa.

Este artigo está estruturado da seguinte forma: a primeira seção consiste na introdução; a segunda apresenta o MDA – Arquitetura Orientada a Modelos; O DDD – Desenvolvimento Orientado a Domínio é explicitado na terceira seção; a quarta traz uma visão

geral sobre Metodologias Ágeis; na quinta seção é exposto o Modelo de Domínio Orientado a Sprints gerado com base no MDA, nos conceitos de DDD e de metodologias ágeis; e, por fim, as considerações finais na sexta seção.

II. MDA – Arquitetura Dirigida a Modelos

O MDA é um padrão arquitetural empresarial fundamentado em um conjunto de modelos e transformações que tem como foco o desenvolvimento de software. Visa o ganho de produtividade, portabilidade, interoperabilidade, integração e documentação. Da mesma forma, objetiva separar por camadas a regra de negócio e a plataforma, o que possibilita uma maior flexibilidade durante as fases de especificação e de desenvolvimento [9].

Criado em 2001 pelo Object Management Group (OMG), o MDA, possui como principal característica a separação da lógica de negócio, da evolução e da manutenção, tornando assim plausível desenvolver sistemas complexos de forma rápida, robusta e independente de plataforma CCM[1], EJB[2], .NET[3], SOAP[4] [10]. É uma especificação muito empregada por empresas, engenheiros e arquitetos de software, a partir do qual é viabilizado o gerenciamento de projetos complexos em larga escala, com redução do custo e adição de novas tecnologias no decorrer do desenvolvimento do projeto.

De acordo com os padrões estabelecidos pelo OMG, no MDA a linguagem mais utilizada para representar o domínio através de modelos é a UML. Por meio do uso de um diagrama UML é possível gerar a automação de código baseado na representação feita pelo diagrama.

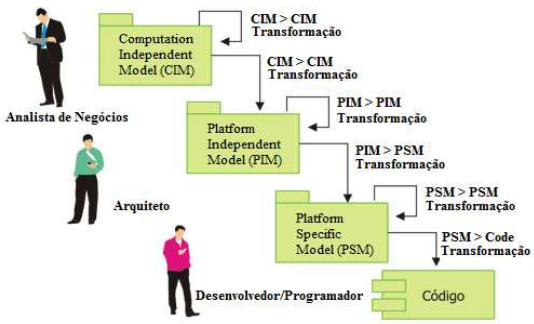


Figura 1 – Ciclo de vida do MDA
Fonte: FREITES [6]

A. CIM – Modelo Computacional Independente *Abbreviations and Acronyms*

O MDA possui um ciclo de vida que é dividido em três etapas ou três camadas, são elas: Computation Independent Model (CIM) onde Analistas de Negócios (Requisitos) são responsáveis pela definição do modelo necessário, Platform Independent Model (PIM) camada na qual os Arquitetos de Software delimitam quais características de domínio, classes e atributos que o modelo deverá possuir e, por fim, a Platform Specific Model (PSM), etapa em que a função dos Analistas e Programadores é criar as regras mais genéricas, esta é a parte de codificação feita em cima do modelo produzido através das transformações, de maneira especifica para a plataforma escolhida [6]. Ver a Figura 1.

O CIM é um modelo computacional que tem como função focar nos requisitos do sistema e na maneira como ele deve funcionar. Sendo assim, ele abstrai todos os detalhes e especificações da estrutura e do processamento para que o modelo seja totalmente independente de tecnologia e plataforma a ser adotada. O CIM possui o papel de desenvolver o domínio de negócio o que origina uma integração junto aos especialistas de domínio, especialistas de tecnologia que serão responsáveis pela implementação do sistema. Gera, desta forma, o padrão de modelo que faz com que os requisitos do CIM sejam rastreados com o intuito de que o PIM e PSM possam dar início ao processo de construção e implementação do software, onde é criada uma alusão de integração geral no processo de desenvolvimento do software [6].

B. PIM – Modelo Independente de Plataforma

PIM é um modelo que tem o papel de descrever as características de domínio, classes e seus atributos. A independência que o MDA traz, permite fazer o mapeamento para uma ou mais plataformas que serão convertidos em modelos PSM, abstraindo todos os detalhes técnicos. Na pós-transformação os serviços serão realizados de maneira especifica para cada tipo de plataforma. A Figura 2 demonstra esta transformação onde o modelo PIM produz PSM distintos [6].

C. PSM – Modelo Especifico de Plataforma

Modelo associado em uma tecnologia que combina especificações de funcionalidades e regras de negócios definidas pelo PIM, com os detalhes necessários para sejam determinados quais os tipos de plataformas serão usadas pelo sistema. Caso todos as particularidades de implementação não sejam delimitados pelo PSM, este passa a ser considerado um modelo abstrato, baseado em modelos implícitos ou explícitos que não possuam as informações necessárias.

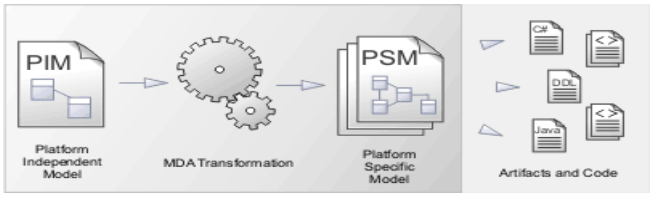


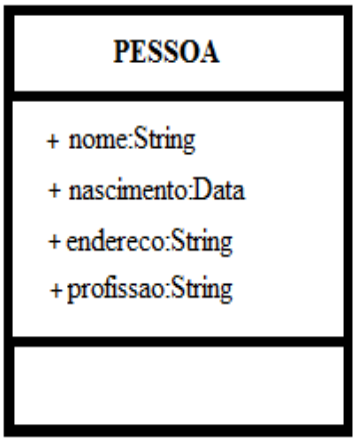
Figura 2 – Transformação: Modelo PIM gerando PSM diferentes.
Fonte: SPARX SYSTEMS PTY LTD. [13]

D. Benefícios MDA

De acordo com Souza [12], são diversos os benefícios do MDA, dentre os quais pode-se destacar portabilidade, produtividade, interoperabilidade, manutenção e teste.

- **Portabilidade:** Possui a facilidade de migração para novas plataformas e ambientes de acordo com as especificações do PIM.
- **Produtividade:** Possibilidade de automatizar diversos processos e tarefas realizadas pelos arquitetos e desenvolvedores, o que gera tempo para centralizar apenas no desenvolvimento da lógica do sistema.
- **Interoperabilidade:** Faz integração entre sistemas legados ou sistemas externos de maneira que é altamente facilitada.
- **Manutenção:** Código do projeto disponível em um formato legível, gerado através da automação dos processos, tendo acesso direto às especificações do sistema, o que simplifica as tarefas dos analistas, desenvolvedores e testadores.
- **Teste:** Os Modelos podem ser testados e validados diretamente aos requisitos, manipulando testes em diferentes infraestruturas. Podem ser utilizados para simular o comportamento do sistema em desenvolvimento.

E. Transformações e Mapeamentos



A transformação de um PIM para PSM ocorre através de meta-modelos[6], esta, que possui a finalidade de transformar PIM fonte em PSM alvo, de maneira que seja gerado um registro do mapeamento entre estes modelos. Estes mapeamentos podem ser feitos por meio do uso de MOF[7], XML[8] ou CWM[9]. Observa-se na Figura 3, um exemplo básico de um PIM.

Figura 3 – Representação do PIM Pessoa.
Fonte: Elaborada pelo autor

A representação do PSM, classe Pessoa feita em Java está explicitado na Figura 4.

```
1. public class Pessoa {
2.       public String nome;
3.       public Data nascimento;
4.       public String endereco;
5.       public String profissao;
6.
7.       public Pessoa() {}
8. }
```

Figura 4 – Representação do PSM, classe Pessoa feita em JAVA.
Fonte: Elaborada pelo autor

Na Figura 5 vê-se o registro da transformação feito utilizando XML.

```
1. <registro>
2.     <PIM>Pessoa</PIM>
3.     <PSM>Pessoa.java</PSM>
4. </registro>
```

Figura 5 – Registro da Transformação feito usando XML.
Fonte: Elaborada pelo autor

III. DDD – Desenvolvimento Orientado a Domínio

Domain-driven Design é uma metodologia de desenvolvimento de software criada por Erick Evans, que aborda uma série de conceitos e técnicas, sempre com foco no domínio do software. Domínio este, que significa a representação, em códigos, de um objeto do mundo real, com seus vários atributos e relacionamentos [4].

De acordo com Cukier [4], o DDD, tem como

[1] CORBA Component Model. Modelo CORBA. Disponível em http://www.omg.org/spec/index.htm.
[2] Enterprise JavaBeans. Componente da plataforma Java Enterprise Edition. Disponível em http://www.oracle.com/technetwork/java/index-jsp-140203.html.
[3] .NET. Framework desenvolvido pela Microsoft, onde todo e qualquer código gerado para .Net é executado em qualquer dispositivo que possua um framework da plataforma. Disponível em http://msdn.microsoft.com/pt-br/vstudio/aa496123.
[4] Simple Object Access Protocol (Protocolo Simples de Acesso a Objetos), protocolo empregado para trocar informações estruturadas entre plataformas diferentes. Disponível em www.w3.org/TR/soap.
[5] Unified Modeling Language (Linguagem de Modelagem Unificada) é mundialmente utilizada para representação de aplicações, comportamento, arquitetura, dados e processos de negócios. Disponível em www.uml.org.

objetivo o uso de boas práticas são elas: o alinhamento do código com o negócio, o isolamento entre domínio, a reutilização, o mínimo de acoplamento e ser independente de tecnologia.

- **Alinhamento do código com o negócio:** Na utilização do DDD é necessário buscar o alinhamento dos desenvolvedores com os especialistas de domínio, fato indispensável, quando se aplica este tipo de metodologia.
- **Isolamento entre domínio:** É necessário que exista uma separação do domínio das demais camadas de desenvolvimento.
- **Reutilização:** A aplicação de blocos de código, que possam ser reaproveitados em vários outros trechos, de forma a facilitar a escalabilidade.
- **Mínimo de acoplamento:** Um modelo bem definido, facilita a ocorrência de que partes do sistema interajam sem que exista muita dependência de classes com conceitos distintos.
- **Independência de tecnologia:** Não visa a tecnologia como parte relevante do processo de desenvolvimento de software e sim nas regras de negócio com o objetivo de entender todo o funcionamento do sistema de acordo com os processos a serem executados.

A. *Arquitetura em Camadas*

A arquitetura é composta por quatro partes distintas que são divididas em: Interface de Usuário, Aplicação, Domínio e Infraestrutura [5]. Ao usar DDD, é necessário o isolamento do domínio das demais camadas. Camadas estas que estão ilustradas na Figura 6.

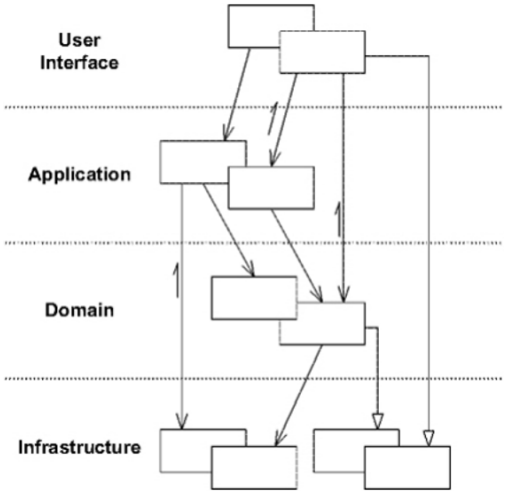


Figura 6 – Representa a Arquitetura em Camadas.

Fonte: EVANS [5]

- **Camada de Interface de Usuário:** Esta é responsável pela exibição de informações do sistema de maneira amigável para com o usuário, de forma que o mesmo possa interagir com o sistema. Ocorre assim a interpretação de comandos feitos pelo usuário.
- **Camada de Aplicação:** Camada responsável por interligar a interface do usuário com as camadas inferiores, esta não possui lógica de negócio.
- **Camada de Domínio:** Esta é a camada principal do DDD. É onde está implementado todos os conceitos, lógica e regras de negócio.
- Camada de Infraestrutura: Responsável pelo suporte às camadas superiores. É nessa camada que ocorre a persistência no banco de dados, conexões com os bancos, envio de mensagens de rede, leitura e gravação em disco.

B. *Linguagem Ubíqua*

Para que se atinja o status de sucesso em um sistema que utiliza o DDD como metodologia em seu desenvolvimento, é necessário que existam interações entre todas as partes envolvidas. Estas partes deverão se comunicar através de uma linguagem técnica que seja compreendida por todos, denominada Linguagem Ubíqua. A Linguagem Ubíqua é um padrão de termos técnicos que serão usados em todo o processo de desenvolvimento do software, desde o levantamento de requisitos até a parte de codificação, onde os envolvidos devem estar a par da linguagem de negócio, ou seja do domínio [8]. Pode-se visualizar na Figura 7 o uso desta linguagem ubíqua.

Este processo é definido pelo Domain Expert[10], ou seja o cliente, o principal conhecedor do domínio, é ele quem irá definir quais atividades e processos deverão ser realizados pelo sistema. Contudo, o cliente pode não conhecer sobre os termos técnicos utilizados em um processo de desenvolvimento de software, por outro lado, assim como o Domain Expert o Analyst[11], pode não conhecer os termos que são utilizados no domínio.

Sendo assim, um diagrama desenvolvido por um analista pode não ser compreendido pelo cliente, ou até mesmo por outros desenvolvedores. A solução para esse tipo de problema pode ser facilmente atingida através do uso de DDD por meio do emprego da linguagem ubíqua, linguagem esta que é utilizada

[6] Meta-modelos: é a maneira formal de representar modelos de uma linguagem de modelagem.
[7] Meta-Object Facility: padrão criado pelo OMG que define uma linguagem para definição de linguagem de modelagem. Disponível em www.omg.org/mof/.
[8] eXtensible Markup Language: linguagem de marcação que descreve diversos tipos de dados. Disponível em www.w3.org/TR/REC-xml/.
[9] Common Warehouse Metamodel: padrão criado pelo OMG para modelagem relacional, não-relacional ou multidimensionais de metadados. Disponível em www.omg.org/spec/CWM/.
[10] Domain Expert: é o especialista de domínio, pessoa que possui conhecimento sobre o domínio do sistema, o cliente.

na interação das partes envolvidas, no código e nas alterações dos modelos, de forma a garantir a melhor compreensão dos modelos. A Figura 7 representa a utilização da linguagem ubíqua em um cenário de DDD, composto por especialistas de domínio, analistas, desenvolvedores, documentações, especificações, parte de codificação e teste deste código, onde estes processos de desenvolvimento são discutidos e alinhados por todas as partes envolvidas através de uma linguagem padrão que é compreendida por todos.

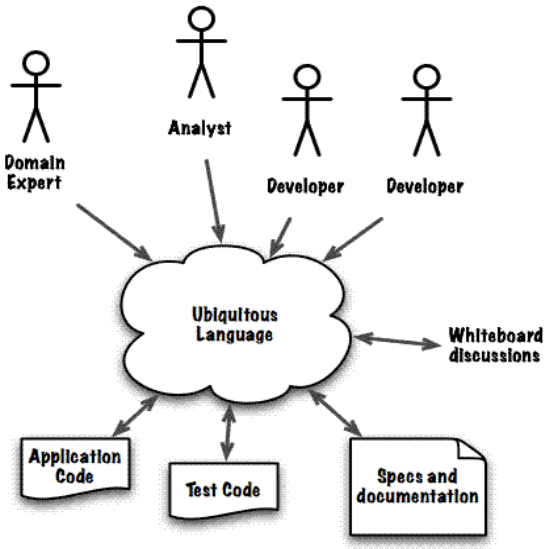


Figura 7 – Uso da Linguagem Ubíqua.
Fonte: BRANDOLINI [3]

IV. Visão Geral sobre Metodologias Ágeis

O mercado atual é caracterizado pela competitividade e prazos restritos infligidos pelos clientes, com isso as empresas que trabalham com desenvolvimento de software vêm buscando métodos mais ágeis para gestão do desenvolvimento de software. A partir de 2001, com a publicação do Manifesto para o Desenvolvimento Ágil, as denominadas metodologias ágeis, que confere novos enfoques para a gestão de projetos de software, tem ocupado um espaço cada vez maior nesse cenário [2].

As metodologias ágeis têm como ideia comum o desenvolvimento baseado mais nas pessoas e suas interações do que em processos rígidos. Dentre essas tecnologias encontra-se o Scrum e o TDD – Desenvolvimento orientado a testes que foram as tecnologias estudadas para a realização deste trabalho.

A. *Scrum*

O Scrum consiste em uma técnica ágil de desenvolvimento de software com princípios compatíveis com o do manifesto ágil. Em linhas gerais, o desenvolvimento de software que utiliza o Scrum ocorre em interações intituladas sprints, ciclos de desenvolvimento do Scrum, caracterizado por ter um curto período onde o time centra em atingir determinada meta [11]. O uso do Scrum, é assim indicado para organizações que possuam entre seus projetos, produtos dinâmicos e com alta taxa de mudança de requisitos [14].

B. *TDD – Desenvolvimento orientado a testes*

Diversas técnicas têm sido utilizadas juntamente com as metodologias ágeis de desenvolvimento. Pode-se destacar dentre estas, o TDD – Desenvolvimento orientado a testes (Test Driven Development), que tornou-se bastante conhecido, após sua adoção em metodologias ágeis. O TDD pode ser considerado um desenvolvimento incremental, onde primeiramente são originados os testes e só posteriormente é escrito o código essencial para passar por eles [7].

Nos últimos anos, segundo Aniche [1], pesquisas como as de Janzen (2005), Maximillien e Williams (2003) e de Edwards (2003) têm sido realizadas com o intuito de averiguar se o TDD verdadeiramente contribui no processo de desenvolvimento de software. Estas investigações apontam para os benefícios do uso do TDD, este além de maximizar o processo de desenvolvimento de software, amplia a qualidade do código, reduz os defeitos e o tempo gasto com depuração, o que acarreta um crescimento na produtividade. Porém, vale ressaltar, que apesar dos efeitos positivos encontrados no uso do TDD, há uma necessidade de se realizar mais experimentos que considerem os diversos elementos de influência em um espaço de desenvolvimento de software [1].

V. Modelo de Domínio Orientado a Sprints

O MDA, o DDD, os princípios do Scrum, bem como o TDD foram utilizados neste artigo como embasamento na proposta de concepção de um novo modelo. Este novo modelo, denominado Modelo de Domínio Orientado a Sprints, foi pensado para gerenciar projetos na área de software e é dividido em 3 camadas: Domínio, Arquitetura e Sprints, conforme demonstrado na Figura 8.

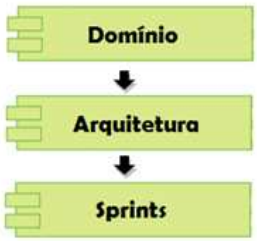


Figura 8 – Camadas do novo modelo
Fonte: Elaborada pelo autor

- **Domínio:** A primeira camada parte da ideia inicial do DDD, que foca inicialmente no domínio. Na primeira interação entre o cliente e os analistas, são levantados todos os requisitos,

[11] Analyst: é o analista, pessoa que possui conhecimentos e técnicas referentes ao desenvolvimento de software.

especificações e documentação do sistema. Com isso, um modelo geral do sistema é desenvolvido embasado nas especificações que foram definidas nesse diálogo cliente-analistas. As próximas interações, devem ser entre todos os envolvimentos no desenvolvimento do sistema, os analistas de requisitos, de negócio, arquitetos, desenvolvedores e testadores, onde será determinado como ocorrerá a modelagem e a divisão do sistema. Esta divisão deve ser subdividida em módulos de forma a facilitar o desenvolvimento de todo o sistema; módulos estes a serem entregues em um determinado prazo e norteados pelos conceitos de Scrum, com a utilização de entregas contínuas para o cliente. A definição de prazos e interfaces são feitos junto com o time de desenvolvimento para que haja tempo hábil para conclusão dos módulos e entrega do projeto por um todo. Esta camada é visualizada na Figura 9.

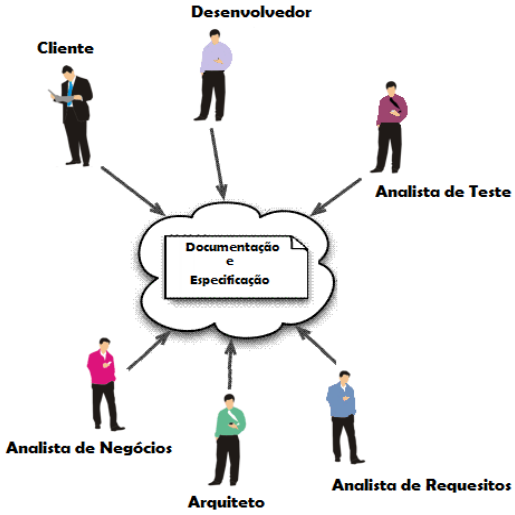


Figura 9 – Interação da camada de domínio
Fonte: Elaborada pelo autor

- **Arquitetura:** Com a modelagem e modularização definidas, inicia-se a segunda camada, considerada como camada da arquitetura do projeto. Nesta etapa é implementado o uso do padrão MDA na transformação dos modelos de PIM para PSM na(as) plataformas que foram expostas no escopo cliente-analistas. A partir desta transformação os modelos que eram incialmente abstratos e independentes, passam a serem específicos, gerando códigos genéricos. Com os modelos PSM originados, as atividades da esquipe são estipuladas com a função de executar e implementar as funcionalidades apoiadas nos modelos que foram originados pelo MDA. A Figura 10 ilustra esta camada.

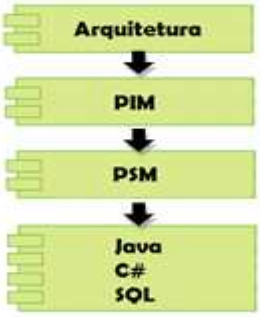


Figura 10 – Fluxo da camada de arquitetura
Fonte: Elaborada pelo autor

Ao fim desse processo de transformação do MDA, o software, teoricamente, ainda não está finalizado. Os módulos obtidos através desta modificação passam para a camada das Sprints, cada modulo passa ser a ter uma Sprint elaborada. A Figura 11 esboça essa transição da camada de arquitetura para a de Sprints.

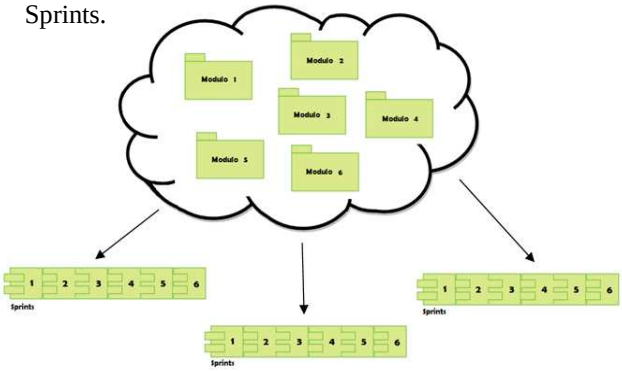


Figura 11 – Transição da camada de arquitetura para a de Sprints
Fonte: Elaborada pelo autor

- **Sprints:** A terceira camada, definida como a camada dos Sprints, consiste no processo de desenvolvimento dos módulos que é subdividido em 6 etapas. Estas etapas são: análise e projeto, tarefas, cronograma, TDD, implementação e entrega da Sprint. Ver figura 12.

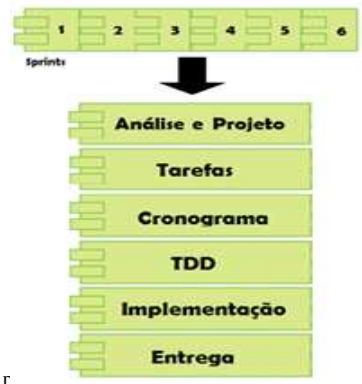


Figura 12 – Subdivisão do módulo
Fonte: Elaborada pelo autor

- **Análise e Projeto:** Através da análise e levantamento dos requisitos, junto com a especificação já definida para cada funcionalidade do sistema, as sprints começam a ter copo para o início do desenvolvimento do projeto e as demais etapas.

- **Tarefas:** As tarefas da equipe são divididas por nível de complexidade pelos membros da mesma. Essas tarefas são postas em um quadro que é fixado em local onde todos têm acesso. Os estados das tarefas (a fazer, andamento ou concluído) também devem ser acrescentado no quadro, assim, ao olhar o quadro, a equipe é capaz de perceber facilmente como está o andamento do trabalho.

- **Cronograma:** Etapa onde ocorre o planejamento de prazos e datas para execução de testes, implementação, codificação, interações e entrega do pacote.

- **TDD:** Nesta etapa, são criados testes, de forma a possibilitar avaliação da modelagem antes de escrever o código. Com isso, torna-se possível escrever código funcional limpo e bem testado.

- **Implementação:** Após a criação dos testes, a etapa de implementação entra em execução, para o desenvolvimento das funcionalidades de cada módulo.

- **Entrega:** Com a conclusão das etapas anteriores, é gerado um pacote por módulo, que obedece o prazo de entrega estabelecido no cronograma. Ao final da entrega de todos os pacotes, um novo pacote geral é apresentado. O pacote geral contém todos os outros implementados e sua entrega gera a conclusão e sucesso do projeto.

Assim, a partir destas três camadas denominadas Domínio, Arquitetura e Sprints, um o Modelo de Domínio Orientado a Sprints é criado. Modelo este cuja redução de erros, aumento da qualidade, redução de tempo de desenvolvimento e do custo do projeto almeja-se fazer parte da proposta.

Ao utilizar o Scrum neste modelo, espera-se obter redução de custo e tempo, uma vez que nesta metodologia a prioridade de entrega são os requisitos de maior valor de negócio ou aqueles que apresentam maior risco estratégico para a empresa, ou seja, será desenvolvido, testado e entregue primeiro, tudo aquilo que for mais importante para o negócio do cliente. Isso permite que seja agregado mais valor ao negócio, em um tempo menor, reduzindo os custos de operação e ampliando o retorno sobre os investimentos do projeto. A flexibilidade às mudanças é outro atrativo no Scrum. Se por exemplo, o cliente mudar, o mercado modificar ou mesmo o que antes era tão importante hoje deixou de fazer sentido, no Scrum, altera-se o planejamento e reprioriza-se o backlog. Outro aspecto importante do uso do Scrum é que nesta metodologia o produto vai sendo construído e entregue à medida que o projeto avança e novas funcionalidades vão sendo surgindo no mercado, o que admite que produtos sejam lançados e validados no mercado rapidamente [11].

O emprego do TDD foi pensado uma vez que este auxilia na escrita de um código com qualidade e legível, contribui na redução de defeitos e facilita a correção deste caso eles surjam, além de evitar retrabalho da equipe, o que ao final reduz custos e colabora no aumento da chance de sucesso. Isso se deve ao fato do TDD proporcionar o pensar antes de programar o que direciona para a análise de problemas para o teste passar. Primeiro se escreve os testes, antes de implementar o sistema [7].

Sendo assim, com o uso do Scrum e do TDD neste modelo, espera-se que apresente benefícios referente ao aumento de qualidade e redução de tempo e custo no clico de desenvolvimento de softwares.

VI. Conclusão

Este artigo, definiu o Modelo de Domínio Orientado a Sprints de desenvolvimento de software a partir do estudo do MDA, DDD, metodologias ágeis (Scrum e TDD). Este modelo, além de tornar possível uma maior interação com o cliente no decorrer do processo de desenvolvimento, sugere trazer benefícios referente ao aumento de qualidade e redução de tempo e custo no clico de desenvolvimento de softwares. Outro aspecto positivo do modelo proposto é que, através do foco no negócio, da proximidade da equipe com o cliente e da arquitetura que possibilita uma automação de código, o processo parece se tornar mais ágil e eficaz para ser adotado na temática de desenvolvimento de sistemas e de softwares.

Apesar da pesquisa sugerir que o Modelo de Domínio Orientado a Sprints auxilie na qualidade e ao mesmo tempo contribua na redução de tempo e custo, é indicado um aprofundamento maior sobre a temática. Da mesma forma, faz-se necessário o desenvolvimento prático deste modelo de forma a testar e validar este.

Este artigo abarca assim alguns pontos significativos sobre o tema estudado, podendo servir como um caminho para futuros trabalhos que

contribuirão para a criação de novos modelos embasados no MDA, no DDD e nas metodologias ágeis. Destarte, ao invés do fim, este trabalho aponta para o início de novos caminhos e possibilidades no âmbito da geração de software.

REFERÊNCIAS

[1] ANICHE, Mauricio. TDD realmente ajuda? São Paulo, 16 abr. 2010. Disponível em: <http://www.aniche.com.br/2010/04/tdd-realmente-ajuda/>. Acessado em: 10 abr. 2014.

[2] BERTHOLDO, Leonardo; BARBAN, Lidia R. C. F. Adaptação do Scrum ao Modelo Incremental. 2010. 14f. Monografia – Faculdade de Tecnologia, Universidade Estadual de Campinas, Limeira, 2010. Disponível em: <http://www.ft.unicamp.br/liag/Gerenciamento/monografias/Monografia_ModeloIncremental_SCRUM.pdf>. Acessado em: 12 maio 2014.

[3] BRANDOLINI, Alberto. Strategic Domain Driven Design with Context Mapping. 2009. Disponível em: <http://www.infoq.com/articles/ddd-contextmapping>. Acessado em: 19 abr 2014.

[4] CUKIER, Daniel. DDD – Introdução a Domain Driven Design. 2010. Disponível em: <http://www.agileandart.com/2010/07/16/ddd-introducao-a-domain-driven-design/>. Acessado em 15 abr. 2014.

[5] EVANS, Eric. Domain-Driven Design: Tackling Complexity in the Heart of Software. EUA: Addison-Wesley, 1ª edição, 2003.

[6] FREITES, Rafael González. Proyectos y procesos de software. Azua, [s.d.]. Disponível em: <http://www.monografias.com/trabajos96/proyectos-y-procesos-software/proyectos-y-procesos-software.shtml>. Acessado em: 19 abr 2014.

[7] GASPARETO, Otávio. Test Driven Development. Rio Grande do Sul, 2006. Disponível em: <http://www.inf.ufrgs.br/~cesantin/TDD-Otavio.pdf>. Acessado em: 14 maio 2014.

[8] HOLDER, Thiago. Uma introdução ao Desenho direcionado ao domínio. 2010. Disponível em: <http://thiagoholder.wordpress.com/2010/03/20/uma-introducao-ao-desenho-direcionado-ao-dominio/>. Acessado em 22 abr. 2014.

[9] KLEPPE, A., WARMER, J., BAST, W. MDA Explained: The Model Driven Architecture™: Practice and Promise. EUA: Addison-Wesley, 2003.

[10] OMG. MDA Guide Version 1.0.1. Document Number: omg/2003-06-01. 12 jun. 2003 Disponível em: <http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf>. Acessado em: 2 abr 2014.

[11] PATRÍCIO, Robério Gomes; MACEDO, Natália de Cassia Coelho; FRANÇA, Cícero Tadeu Pereira Lima. Pomodoro aliado a SCRUM para aumento da produtividade: um estudo de caso. In: CONGRESSO TECNOLÓGICO INFOBRASIL TI & TELECOM, IV, 2011, Fortaleza. Anais... Fortaleza, 2011. Disponível em: <http://www.infobrasil.inf.br/userfiles/Pomodoro%20aliado%20a%20SCRUM%20para%20aumento%20da.pdf>. Acessado em: 10 maio 2014.

[12] SOUZA, Thiago Silva de. Model Driven Architecture – Conceitos Fundamentais. [s.d.]. Disponível em <http://www.linhadecodigo.com.br/artigo/1953/model-driven-architecture-conceitos-fundamentais.aspx>. Acessado em 20 maio 2014.

[13] SPARX SYSTEMS PTY LTD.. Model Transformation. [s.d.]. Disponível em <http://www.sparxsystems.com/enterprise_architect_user_guide/9.3/model_transformation/mdastyletransforms.html>. Acessado em 10 abr. 2014.

[14] VARASCHI, Jacques Douglas. Implantando o SCRUM em um Ambiente de Desenvolvimento de Produtos para Internet. Monografia (Ciência da Computação) – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2009. Disponível em: <ftp://ftp.inf.puc-rio.br/pub/docs/techreports/09_07_varaschim.pdf>. Acessado em: 10 maio 2014.