Aplicação de Domain-Driven Design no gerenciamento de GRU de Cronotacógrafo no Inmetro/RS

Tiago O. de Farias[1]

[1]UniRitter Laureate International Universities
Rua Orfanatrófio, 555 - Alto Teresópolis - 90840-440 - Porto Alegre - RS - Brasil

tiago.farias.poa@gmail.com

Abstract. *Since 1997, when it was instituted the Brazilian Traffic Code, cargo vehicles with a gross weight exceeding 4536 kilograms and passengers with more than 10 seats must have tachograph. From 2009, the instruments should also be checked periodically by Inmetro (National Institute of Metrology, Quality and Technology), which increases the reliability of the measurements. All the tachograph verification process is managed through a web system. This paper presents a proposal for improving the current web system using some techniques Domain-driven design. The main objective is to provide an organized, highly reusable and production structure; where the application life cycle can be extended. Throughout this work is done a study on the concept of Domain-Driven Design, in addition to the current application improvement case study.*

Resumo. *Desde 1997, quando foi instituído o Código de Trânsito Brasileiro, veículos de carga com peso bruto superior a 4.536 kg e de passageiros com mais de 10 lugares devem possuir cronotacógrafo. A partir de 2009, os instrumentos também devem ser verificados periodicamente pelo Inmetro (Instituto Nacional de Metrologia, Qualidade e Tecnologia), o que aumenta a confiabilidade das medições. Todo o processo de verificação do cronotacógrafo é gerenciado através de um sistema web. Este trabalho apresenta uma proposta de melhoria do atual sistema web utilizando algumas técnicas de Domain-Driven Design. O principal objetivo é fornecer uma estrutura organizada, altamente reutilizável e produtiva; onde o ciclo de vida da aplicação pode ser prolongado. Ao longo deste trabalho é feito um estudo sobre o conceito de Domain-Driven Design, além do estudo de caso de melhoria da atual aplicação.*

1. Introdução

O Brasil está entre os países que tornaram o uso do cronotacógrafo obrigatório em ônibus e caminhões, o instrumento inibe os excessos e ajuda a reduzir os acidentes, uma vez que registra o histórico das velocidades desenvolvidas, distâncias percorridas e tempos de movimento e paradas do veículo.

O processo de verificação tem basicamente três grandes etapas: emissão e pagamento da GRU (Guia de Recolhimento da União), realização da selagem e realização do ensaio metrológico. O sistema de gerenciamento de verificação do cronotacógrafo existe há pelo menos oito anos e atende a todos os estados da federação. Inicialmente foi concebido para simples emissão de GRU onde o proprietário do veículo acessava o site do cronotaógrafo preenchia seus dados, emitia e pagava GRU, após se dirigia a um posto

de selagem para dar inicio ao processo, ao finalizar a selagem e dentro de um período determinado deveria procurar um posto de ensaio para realizar o ensaio metrológico e assim recebia o certificado do Inmetro que garante que o instrumento atende aos padrões vigentes sob penalização de ser autuado pela policia federal.

No dia 01 de janeiro de 2016 houve uma grande mudança no processo de emissão de GRU, hoje os postos de selagem e de ensaio emitem a GRU, e semelhante a um plano pré-pago de celular, o posto compra créditos para cada GRU emitida, cada crédito permite ao posto acessar o site do Inmetro e registrar os dados do serviço realizado, seja serviço de selagem ou de ensaio metrológico.

Este artigo está organizado da seguinte forma. A seção 2 descreve o que é a Linguagem Ubíqua e sua importância. A seção 3 introduz o conceito de DomainDriven Design. A seção 4 descreve como isolar o domínio das demais partes do sistema. A seção 5 fala sobre representação do modelo no software e a diferença entre entidade e objeto de valor. A seção 6 trata de padrões para se manter o ciclo de vida de um objeto de domínio. A seção 7 demonstra algumas técnicas descritas pelo DDD em um caso de uso.

2. Referencial Teórico

2.1. Domain-Driven Design

Domain-Driven Design é uma abordagem de desenvolvimento de software desenhado para gerir a complexa e grande escala de produtos de software.[Evans 2003]

A melhor maneira de justificar uma tecnologia ou técnica é fornecer valor ao negócio.[Carlos Buenosvinos and Akbary 2014]

Segundo [Evans 2003] é um processo que alinha o código do desenvolvedor com o problema real, um conjunto de técnicas de desenvolvimento de software, usadas principalmente em projetos complexos que provê conceitos e regras para ajudar no cilco do desenvolvimento de software, essas técnicas também tem o objetivo de ajudar clientes, gestores e todas as pessoas envolvidas no processo de desenvolvimento. Seguir a filosofia DDD dará aos desenvolvedores o conhecimento e as habilidades que eles precisam para enfrentar sistemas de negócios grandes e complexos de maneira eficaz.[Millett and Tune 2015]

DDD utiliza o princípio da separação de conceitos. Este princípio é usado para separar a aplicação do modelo em um modelo de domínio, que consiste de domínio relacionado com a funcionalidade, e domínio independente de funcionalidade, que é representado por diferentes serviços que facilitam a usabilidade do modelo de domínio [Uithol 2008]

A solução está em resolver o problema do domínio, focado no domínio do problema, falar a mesma linguagem dos especialistas do domínio. Pode ser aplicado em qualquer projeto desde que não torne complicado o desenvolvimento, algumas vezes o desenvolvimento não é tão complexo que necessite do DDD. O foco está na criação de uma linguagem comum conhecida como a linguagem ubíqua para descrever de forma eficiente e eficaz um domínio de problemas.[Millett and Tune 2015] De acordo com o conceito do DDD o mais importante em um software não é somente o código, não é somente a arquitetura, tampouco a tecnologia sobre o qual foi desenvolvido, mas sim o problema que o mesmo se propõe a resolver, a regra de negócio. Ela é a razão do software existir.

DDD é sobre o desenvolvimento de conhecimento em torno do negócio e de utilizar a tecnologia para fornecer valor.[Carlos Buenosvinos and Akbary 2014]

Ainda segundo [Carlos Buenosvinos and Akbary 2014] Domain-Driven Design não é uma bala de prata, como tudo em software, que depende do contexto. Como regra geral, deve ser utilizado para simplificar o domínio, nunca para adicionar mais complexidade. DDD é mais do que tecnologia ou metodologia, ou até mesmo um framework. É uma maneira de pensar, é um conjunto de prioridades que visa acelerar projetos de software que têm de lidar com domínios complicados. [Vlahovic]

Pensar nos problemas de forma técnica não é ruim, o único problema é que, às vezes, pensar menos tecnicamente é melhor. A fim pensar em comportamentos de objetos precisamos pensar na Linguagem universal em primeiro lugar. [Carlos Buenosvinos and Akbary 2014]

De forma geral [Carlos Buenosvinos and Akbary 2014] aifrma que os benefícios com a utilização do DDD são:

- Alinhamento com o modelo do domínio
- Especialistas do domínio contribuem para o design do software
- Melhor experiência do usuário
- Limites claros
- Melhor organização da arquitetura
- Modelagem contínua de forma ágil

2.2. Linguagem Ubiqua

Que está ou pode estar em toda parte ao mesmo tempo; onipresente. [Michaelis 2011]

A linguagem ubíqua é uma linguagem de equipe compartilhada. Ela é compartilhada por especialistas em domínio e desenvolvedores. Na verdade, ela é compartilhada por todos na equipe do projeto. Não importa o seu papel na equipe, uma vez que você está na equipe que usa a linguagem ubíqua do projeto. [Vernon 2013] Esta linguagem é composta de documentos, diagramas de modelo, e mesmo código. Se um termo está ausente no projeto, é uma oportunidade para melhorar o modelo incluindo-a.[Evans 2003] A linguagem ubíqua exige que os desenvolvedores trabalhem duro para entender o domínio do problema, mas também exige que a empresa trabalhe duro para ser precisa em suas nomenclaturas e descrição desses conceitos. [Haywood 2009] Se uma ideia não pode ser expressa usando este grupo de conceitos, o modelo deve ser extendido para procurar e remover as ambiguidades e as inconsistências.[Haywood 2009]

Especialistas do domínio podem se comunicar com os times de desenvolvedores de sistema sobre as regras do domínio através da linguagem ubiqua que também representa a especificação formal do sistema.[Vlahovic]

Para [Millett and Tune 2015] se a equipe de desenvolvimento não se envolver com especialistas de domínio para compreender plenamente a linguagem e usá-la no âmbito da implementação de código, muito do seu benefício é perdido. Para alcançar uma melhor compreensão, as equipes precisam se comunicar de forma eficaz. É a criação da linguagem onipresente que permite uma compreensão mais profunda do que vai permanecer após o código ser reescrito e substituído. [Millett and Tune 2015] Ainda segundo [Millett and Tune 2015] a utilidade da criação de uma linguagem ubiqua tem um impacto

que vai além da aplicação para o produto em desenvolvimento. Ela ajuda a definir explicitamente o que a empresa faz, revela uma compreensão mais profunda do processo e a lógica do negócio, e melhora a comunicação empresarial. Enquanto as equipes estão implementando o modelo em código, novos conceitos podem aparecer. Estes termos descobertos precisam ser levados de volta para os especialistas de domínio para validação e esclarecimentos. [Millett and Tune 2015]

Um projeto enfrenta sérios problemas quando os membros da equipe não compartilham uma linguagem comum para discutir o domínio. [Avram 2007] Ele ainda defende usar um modelo como espinha dorsal de uma línguagem. Solicitando que a equipe deve usar a linguagem de forma consistente em todas as comunicações e também no código.

2.3. Diagrama de Casos de Uso

É a especificação de uma seqûencia de interações entre um sistema e os agentes externos que utilizam esse sistema. [Bezerra 2006].

O diagrama de casos de uso procura, por meio de uma linguagem simples possibilitar a compreensão do comportamento externo do sistema por qualquer pessoa, tentanto apresentar o sistema através da perspectiva do usuário. [Guedes 2009]

Ainda segundo [Guedes 2009] o diagrama de casos de uso sendo uma linguagem informal e apresentar uma visão geral do comportamento do sistema a ser desenvolvido, pode e deve ser apresentado durante as reuniões inicias com os clientes com uma forma de ilustrar o comportamento do sistema, facilitando a compreenção dos usuários e auxiliando na identificação de possíveis falhas de especificação, verificando se os requisitos do sistema foram bem compreendidos.

3. Estado da Arte

4. Solução Implementada

4.1. Subsections

The subsection titles must be in boldface, 12pt, flush left.

5. Considerações Finais

6. Figures and Captions

Figure and table captions should be centered if less than one line (Figure 1), otherwise justified and indented by 0.8cm on both margins, as shown in Figure 2. The caption font must be Helvetica, 10 point, boldface, with 6 points of space before and after each caption.

In tables, try to avoid the use of colored or shaded backgrounds, and avoid thick, doubled, or unnecessary framing lines. When reporting empirical data, do not use more decimal digits than warranted by their precision and reproducibility. Table caption must be placed before the table (see Table 1) and the font used must also be Helvetica, 10 point, boldface, with 6 points of space before and after each caption.

Figure 1. A typical figure

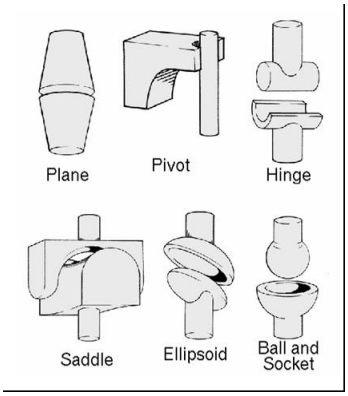

Figure 2. This figure is an example of a figure caption taking more than one line and justified considering margins mentioned in Section 6.

7. Referencias

Bibliographic references must be unambiguous and uniform. We recommend giving the author names references in brackets, e.g. [Knuth 1984], [Boulic and Renault 1991], and [Smith and Jones 1999].

The references must be listed using 12 point font size, with 6 points of space before each reference. The first line of each reference should not be indented, while the subsequent should be indented by 0.5 cm.

References

Avram, A. (2007). *Domain-driven design Quickly*. Lulu. com.

Bezerra, E. (2006). *Princípios De Análise E Projeto De Sistemas Com Uml-3ª Edição*, volume 3. Elsevier Brasil.

Boulic, R. and Renault, O. (1991). 3d hierarchies for animation. In Magnenat-Thalmann, N. and Thalmann, D., editors, *New Trends in Animation and Visualization*. John Wiley & Sons ltd.

Table 1. Variables to be considered on the evaluation of interaction techniques

	Chessboard top view	Chessboard perspective view
Selection with side movements	6.02 ± 5.22	7.01±6.84
Selection with in-depth movements	6.29±4.99	12.22±11.33
Manipulation with side movements	4.66± 4.94	3.47±2.20
Manipulation with in-depth movements	5.71 ±4.55	5.37 ±3.28

Carlos Buenosvinos, C. S. and Akbary, K. (2014). *Domain-Driven Design in PHP*. Leanpub. Real examples written in PHP showcasing DDD Architectural Styles, Tactical Design, and Bounded Context Integration.

Evans (2003). *Domain-Driven Design: Tacking Complexity In the Heart of Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Guedes, G. T. (2009). Uml 2. *Uma Abordagem Prática, São Paulo, Novatec*.

Haywood, D. (2009). *Domain-driven design using naked objects*. Pragmatic Bookshelf.

Knuth, D. E. (1984). *The TEX Book*. Addison-Wesley, 15th edition.

Michaelis, D. (2011). Disponível em: http://michaelis.uol.com.br. *Acesso em 28/05/2016*, 7.

Millett, S. and Tune, N. (2015). Patterns, principles, and practices of domain-driven design.

Smith, A. and Jones, B. (1999). On the complexity of computing. In Smith-Jones, A. B., editor, *Advances in Computer Science*, pages 555–566. Publishing Press.

Uithol, M. (2008). Security in domain-driven design.

Vernon, V. (2013). *Implementing domain-driven design*. Addison-Wesley.

Vlahovic, N. Implications of domain-driven design in complex software value estimation and maintenance using dsl platform.