Engenharia de Software III

Aula 6
Domain-Driven Design (DDD)

l.bertholdo@ifsp.edu.br

Conteúdo

- Modelo de Domínio
- Domain-Driven Design (DDD)
- Construção do Conhecimento de Domínio
- Linguagem Ubíqua
- Criação da Linguagem Ubíqua

l.bertholdo@ifsp.edu.br

Modelo de Domínio

- Em geral, softwares são desenvolvidos para automatizar processos e solucionar problemas do mundo real, os quais estão relacionados a um determinado **domínio**.

Aqui, "domínio" refere-se a um conjunto de conceitos e informações comuns a uma determinada esfera de conhecimento, atividade ou negócio, para a qual um sistema será desenvolvido.

- O **domínio** é a **razão essencial** de o software existir. No entanto, código é a matéria elementar do software, e podemos ficar tentados a vê-lo simplesmente como um conjunto de classes e métodos.

*Considere a fabricação de carros: os trabalhadores são especialistas na produção das peças, mas geralmente têm uma **visão limitada do processo de fabricação**. Eles veem o carro como uma enorme **coleção de peças que precisam se encaixar**, mas um carro é muito mais do que isso. Um bom carro começa com uma visão, com especificações bem escritas, e continua com o design. Após meses ou anos de refinamentos, o projeto passa a refletir uma visão ideal, quando então entra em produção.*

l.bertholdo@ifsp.edu.br

Modelo de Domínio

- O desenvolvimento de software é semelhante. Não podemos simplesmente sentar e digitar o código, pelo menos não para criar sistemas de software complexos.
- Não é possível criar um sistema de software bancário a menos que se tenha uma boa compreensão de como funciona um banco, ou seja, é preciso entender o **domínio** das instituições financeiras.
- Analistas de requisitos, arquitetos de software e desenvolvedores não entendem de instituições financeiras, mas os bancários entendem.

*O sistema bancário é muito bem compreendido pelas pessoas de dentro, por seus especialistas. Eles conhecem todos os detalhes, regras, armadilhas e problemas possíveis. É por aqui que devemos começar a especificar e projetar o software: pelo **domínio**.*

l.bertholdo@ifsp.edu.br

Modelo de Domínio

- Quando iniciamos um projeto de software, devemos nos concentrar no **domínio** em que ele irá operar, pois todo o propósito do software é **dar apoio a um domínio específico**.

- Para isso, o software deve estar em **harmonia** com o domínio para o qual foi criado. Caso contrário, ele introduzirá tensão no domínio, provocando mau funcionamento, danos e até mesmo o caos.

- A melhor maneira de fazer isso é tornar o software um **reflexo** do domínio. O software precisa incorporar os **conceitos e elementos centrais do domínio** e implementar com precisão as **relações entre eles**. Em outras palavras, o software tem que **modelar** o domínio.

Modelo de Domínio

- Aprendemos muito sobre um domínio enquanto conversamos com seus especialistas. Mas esse conhecimento bruto não pode ser facilmente transformado em software, a menos que construamos uma **abstração** dele em nossas mentes.

- No início, essa representação abstrata estará incompleta, mas com o tempo, conforme trabalhamos nela, ela se tornará cada vez mais clara. Essa abstração nada mais é do que um **modelo do domínio**.

*Um **modelo de domínio** não é um diagrama em si, mas sim a ideia que um diagrama pode transmitir. Não é apenas o conhecimento na cabeça de um especialista do domínio. É uma abstração rigorosamente **organizada** e **seletiva** desse conhecimento. O diagrama serve apenas para representar e comunicar o modelo, da mesma forma que um código cuidadosamente escrito ou uma frase em português bem redigida.*

l.bertholdo@ifsp.edu.br

Modelo de Domínio

- O mundo ao nosso redor tem um excesso de informações para nossas cabeças lidarem. Mesmo considerando um domínio específico, seus detalhes podem exceder a capacidade da mente humana.

- Por isso, precisamos organizar a informação, sistematizá-la, dividi-la em pedaços menores, agrupar esses pedaços em módulos lógicos.

- Um domínio pode conter informações demais para incluí-las todas no modelo e precisamos deixar algumas partes de fora. Este é um desafio por si só: o que guardar e o que jogar fora?

O sistema bancário certamente deverá conhecer o endereço do cliente, mas não deve se preocupar com a cor dos olhos dele. Esse é um caso óbvio, mas outros exemplos podem não ser tão óbvios.

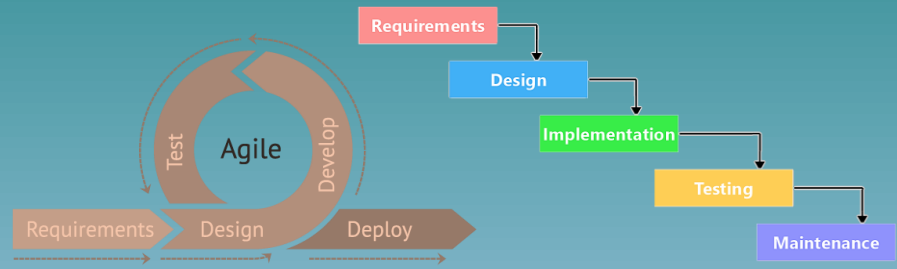
l.bertholdo@ifsp.edu.br

Modelo de Domínio

- Uma vez desenhado o **modelo de domínio**, precisamos apresentá-lo a especialistas de domínio, colegas *designers* e desenvolvedores.
- Para isso, é preciso criar formas de expressar este modelo aos outros, a fim de compartilhar conhecimento e informação. Isso deve ser feito com precisão, de forma completa e sem ambiguidades.
- Existem três recursos para fazer essa comunicação:
  - **Gráficos:** Diagramas, casos de uso, desenhos, imagens etc.
  - **Textos:** Sentenças que explicam nossa visão do domínio.
  - **Linguagem específica:** Conjunto de expressões e termos comuns ao domínio.

l.bertholdo@ifsp.edu.br

Modelo de Domínio



- Após definir o modelo de domínio, podemos começar a pensar no ***design* do software** (projeto de software), uma atividade que faz parte de diversos **modelos de processo** de software.

- Um deles é o **modelo cascata**, um fluxo de etapas unidirecional. Entre os problemas dessa abordagem está a dificuldade de se criar, no início do projeto, um modelo que abranja todos os aspectos de um domínio.

- Outro modelo de processo, no qual o *design* de software está inserido, são as **metodologias ágeis**, que também têm suas próprias limitações.

- Elas defendem a **simplicidade**, mas cada um tem sua própria visão do que isso significa. A **refatoração contínua** feita sem princípios sólidos de *design* produzirá código ruim. E a aversão **aos modelos tradicionais** pode levar à aversão de fazer um projeto bem pensado.

l.bertholdo@ifsp.edu.br

Domain-Driven Design (DDD)

- Nesse contexto, o ***Domain-Driven Design*** (projeto orientado por domínio) é uma abordagem de desenvolvimento que tem ajudado os diversos modelos de processo a modelar um determinado domínio de maneira eficiente e sustentável.

- Quando aplicadas, as diretrizes do DDD podem aumentar muito a capacidade de qualquer processo de desenvolvimento de tratar problemas complexos existentes em um domínio.

*Para isso, **DDD** combina atividades de requisitos, design e desenvolvimento, e reúne um conjunto de conceitos, princípios e técnicas que são empregados para criar um modelo do domínio para o software a ser desenvolvido.*

l.bertholdo@ifsp.edu.br

Construção do Conhecimento de Domínio

- Para ter o modelo de um domínio, é preciso **construir o conhecimento** desse domínio.
- Supondo um projeto de sistema de controle de tráfego aéreo e como o conhecimento desse domínio pode ser construído:

Um sistema de monitoramento de voos deve garantir que os milhares de voos em todo planeta não colidam no ar. Portanto, o projeto a ser proposto é um sistema que rastreia cada voo sobre uma determinada área, determina se o voo está seguindo sua rota ou não e se há possibilidade de colisão.

- Começamos entendendo o **domínio** (monitoramento de tráfego aéreo). Os controladores de tráfego aéreo são os especialistas neste domínio, mas não são projetistas de software. Logo, não podemos esperar que eles entreguem toda a descrição necessária para desenvolver o sistema.

l.bertholdo@ifsp.edu.br

Construção do Conhecimento de Domínio

- Embora os controladores de tráfego aéreo tenham um vasto conhecimento sobre seu domínio, para construir um modelo é preciso selecionar as informações essenciais e organizá-las.

> *Quando você começar a falar com os controladores de tráfego, ouvirá muito sobre aeronaves decolando e pousando, aeronaves no ar e o perigo de colisão, aviões esperando antes de serem autorizados a pousar etc. Para encontrar ordem nesse conjunto caótico de informações, é preciso começar de algum lugar.*

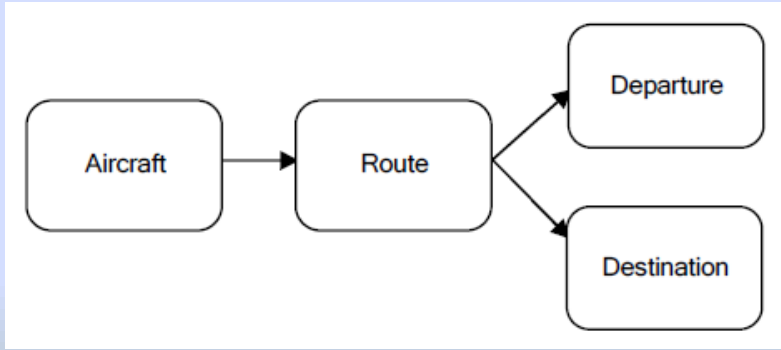
- Apesar da discrepância de visões, tanto você e quanto o controlador concordam que cada aeronave tem um aeroporto de partida e um de destino:

Construção do Conhecimento de Domínio

- Já sabemos que o avião decola de um lugar e pousa em outro, porém o mais importante é saber o que acontece enquanto ele está no ar.

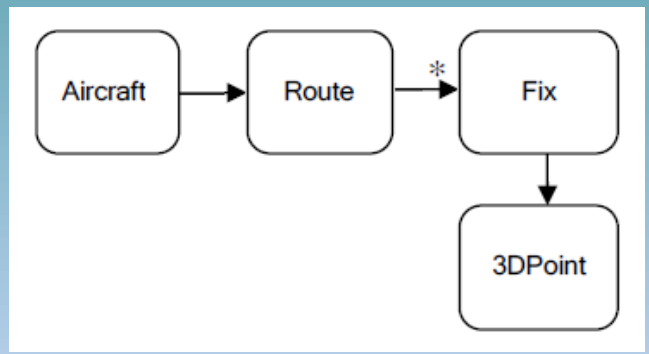
> *O controlador diz que cada avião recebe um **plano de voo** que descreve toda a viagem aérea. Em seguida, você ouve uma palavra que instantaneamente chama sua atenção: **rota**. A rota contém um importante conceito de viagem aérea. Isso é o que os aviões fazem enquanto voam, eles **seguem uma rota**, sendo que os pontos de partida e destino da aeronave são os pontos inicial e final da rota.*

- Então, em vez de associar a aeronave diretamente aos **pontos de partida e destino**, é mais natural associá-la uma **rota**, que por sua vez é associada à partida e destino correspondentes.



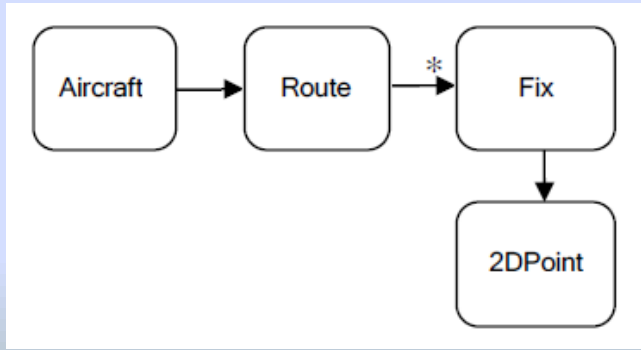
Construção do Conhecimento de Domínio

*Seguindo a conversa, você descobre que a rota é feita de pequenos **segmentos**, que juntos constituem uma espécie de linha torta da partida ao destino. E essa linha passa por **pontos fixos** (Fix), cada qual definido por meio de coordenadas que indicam sua posição (3DPoint).*



- Neste momento, partida e destino passam a ser **apenas mais dois pontos fixos**. Isso provavelmente é diferente de como o controlador os vê, mas é uma abstração necessária que pode ajudar mais tarde.

- Você também percebe que cada ponto fixo é um **ponto tridimensional**, mas ao falar com o controlador, descobre que ele os vê como pontos na superfície da Terra determinados apenas por sua **latitude** e **longitude** (2DPoint). Logo, o diagrama correto é:



Construção do Conhecimento de Domínio

- Ao fazer as perguntas certas e processar as informações corretamente, você e os especialistas começarão a esboçar uma visão do domínio, um **modelo de domínio**. Esta visão não é completa nem perfeita, mas é o começo que você precisa.
- O *designer* de software deve ter uma mente analítica para descobrir os **conceitos-chave** do domínio durante as discussões com especialistas.
- O ***feedback*** entre as partes também é essencial para a criar um modelo melhor e uma compreensão mais clara e correta do domínio.

*Os especialistas de software e os especialistas de domínio devem criar o **modelo do domínio** juntos, pois o **modelo** é justamente o local onde ambas áreas de especialização devem se encontrar em harmonia.*

l.bertholdo@ifsp.edu.br

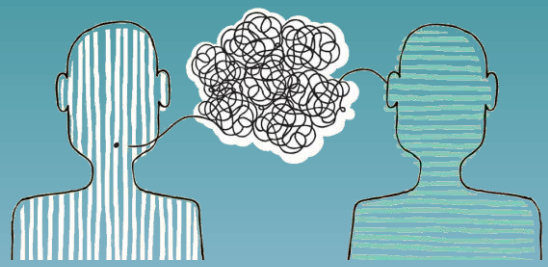
Linguagem Ubíqua

- O trabalho em conjunto de especialistas de software e de domínio é essencial para criar um modelo do domínio, porém essa abordagem geralmente apresenta algumas dificuldades iniciais.
- Desenvolvedores pensam em classes, métodos, algoritmos, padrões, APIs, bancos de dados e tendem sempre a fazer uma correspondência entre um conceito da vida real e um artefato de programação.
- Entretanto, os especialistas de domínio geralmente não sabem nada sobre isso. No exemplo anterior, eles entendem mesmo é de aviões, rotas, desvios, altitudes, latitudes e longitudes.

*Especialistas de software e de domínio falam os respectivos **jargões** de suas áreas, que muitas vezes não é fácil de entender. Isso pode afetar a comunicação para modelar o domínio. Se um fala algo e o outro não entende (ou entende errado), quais são as chances do projeto dar certo?*

l.bertholdo@ifsp.edu.br

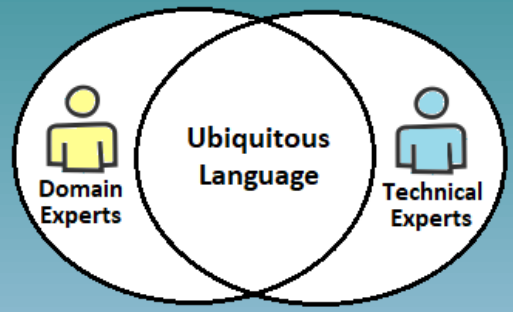
Linguagem Ubíqua



- Um projeto enfrenta sérios problemas quando os membros da equipe não compartilham uma **linguagem comum** para discutir o domínio.
- Durante as sessões de *design*, os desenvolvedores podem tentar explicar alguns padrões de *design* usando a linguagem de um leigo e, às vezes, sem sucesso. Os especialistas do domínio se esforçarão para esclarecer suas ideias, provavelmente criando um novo jargão.
- Nessas tentativas de tradução, a comunicação sofre e o processo de construção do conhecimento é bastante prejudicado.

Tendemos a usar nossos próprios dialetos durante as sessões de design, mas nenhum desses dialetos pode ser um idioma comum, porque nenhum atende às necessidades de todos. Definitivamente, precisamos falar a mesma língua quando nos reunimos para falar sobre o modelo.

Linguagem Ubíqua



- Um princípio básico do DDD é construir uma **linguagem** que reflita o **modelo**, já que o modelo é o local onde o software encontra o domínio. Essa linguagem é chamada de **linguagem ubíqua**.

- Ao compartilhar conhecimento, todos os envolvidos devem usar essa linguagem consistentemente em todas as formas de comunicação utilizadas: na fala, na escrita, nos diagramas e, até mesmo, no código.

*A **linguagem ubíqua** é uma linguagem comum, com termos bem definidos, que fazem parte do domínio do negócio. Por meio dela, os membros da equipe podem descobrir que conceitos iniciais estavam incorretos ou foram usados de forma inadequada, ou que novos elementos precisam ser considerados no projeto.*

Linguagem Ubíqua

- Linguagens não aparecem da noite para o dia. É preciso muito trabalho e foco para encontrar os conceitos-chave que definem o **domínio** e o **modelo**, e depois achar as palavras correspondentes para eles.

- Caso o sistema já exista, refatore o código, renomeando classes, métodos e módulos de acordo com a linguagem especificada.

- O modelo e a linguagem devem estar fortemente interconectados, de modo que uma mudança na linguagem deve se tornar uma mudança no modelo.

É normal, e esperado, que especialistas de domínio se oponham a termos que são inadequados para o domínio. Se eles não conseguem entender alguma coisa, é provável que haja algo errado com o modelo ou a linguagem comum. Por outro lado, os desenvolvedores devem observar possíveis ambiguidades ou inconsistências que podem aparecer no modelo.

Criação da Linguagem Ubíqua

- Mas, como começar a construir uma linguagem comum?

Desenvolvedor: *Vamos iniciar o projeto do sistema de controle tráfego aéreo. Por onde começamos?*

Especialista: *Vamos começar com o básico. Todo esse tráfego é feito de aviões. Cada avião decola de um local de partida, e pousa em um local de destino.*

Desenvolvedor: *Isso é fácil. Quando voa, o avião pode simplesmente escolher qualquer rota aérea? Cabe ao piloto decidir que caminho seguir, desde que cheguem ao destino?*

Especialista: *Oh não. Os pilotos recebem uma rota a qual eles devem seguir e se posicionar o mais próximo possível.*

Desenvolvedor: *Estou pensando nessa rota como um caminho 3D no ar. Se usarmos um sistema de coordenadas, então a rota é simplesmente uma sequência de pontos 3D.*

Especialista: *Nós não vemos a rota dessa maneira. A rota é na verdade a projeção no solo da trajetória aérea esperada do avião. A rota passa por uma série de pontos no terreno determinados por seus latitude e longitude.*

Desenvolvedor: *OK, então vamos chamar cada um desses pontos de "fix", porque são pontos fixos da superfície da Terra. E usaremos então uma série de pontos 2D para descrever o caminho. E, a propósito, a partida e o destino são apenas "fixes". Não devemos considerá-los como conceitos separados. A rota chega ao destino como chega a qualquer outro "fix". O avião deve seguir a rota, mas isso significa que ele pode voar tão alto ou tão baixo quanto o piloto quiser?*

Especialista: *Não. A altitude que um avião deve ter em um determinado momento também é estabelecida no plano de voo.*

Desenvolvedor: *Plano de voo? O que é isso?*

Especialista: *Antes de deixar o aeroporto, os pilotos recebem um plano de voo detalhado que inclui todos os tipos de informações sobre o voo: rota, altitude, velocidade de cruzeiro, tipo de avião, informações sobre os membros da tripulação.*

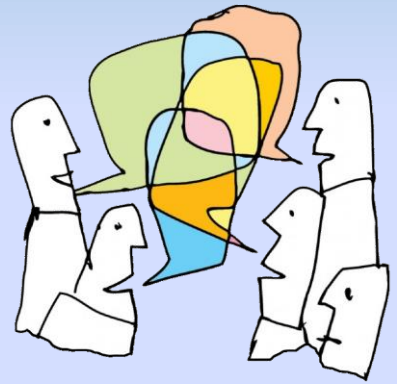
Desenvolvedor: *Hum, o plano de voo me parece muito importante. Vamos incluí-lo no modelo.*

Desenvolvedor: *Ótimo, agora percebo que quando estamos monitorando o tráfego aéreo, não estamos realmente interessados nos aviões em si, se são brancos ou azuis, ou se são Boeing ou Airbus. Estamos interessados em seus voos.*

Criação da Linguagem Ubíqua

- Nesse diálogo hipotético, podemos observar como o desenvolvedor e o especialista de domínio vão aos poucos criando uma linguagem composta pelas **palavras em negrito** e, também, como a criação dessa linguagem é capaz de **mudar o modelo** (inclusão do plano de voo).

- No entanto, na vida real, esse diálogo é muito mais detalhado e as pessoas muitas vezes falam sobre as coisas indiretamente, entram em muitos detalhes ou expõe conceitos de forma equivocada, o que pode dificultar muito a criação da linguagem.

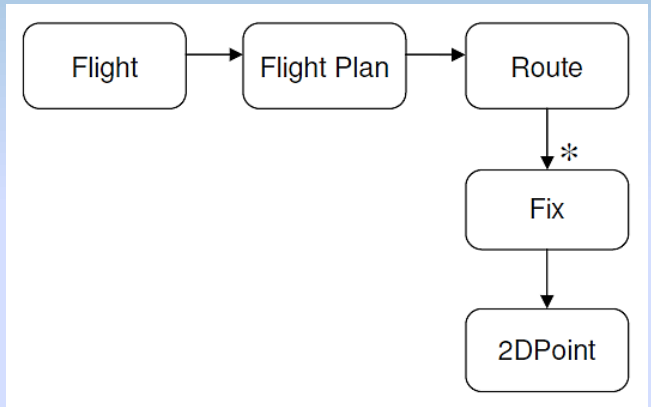


Para criar uma linguagem comum, todos os membros da equipe devem estar cientes de sua necessidade e que, uma vez criada, devem utilizá-la sempre que necessário, procurando usar o próprio jargão o mínimo possível.

Criação da Linguagem Ubíqua

- Os desenvolvedores devem implementar os principais conceitos do modelo no **código**. Por exemplo, pode haver uma classe para a **Route** e outra para **Fix**. A classe **Fix** pode herdar de uma classe **2DPoint** ou pode conter um atributo **2DPoint**.

- Com relação à representação do modelo, a **UML** pode ser uma boa ferramenta para anotar conceitos-chave como classes, atributos, métodos e seus relacionamentos.



*Os diagramas UML são muito úteis, porém eles não podem transmitir um dos aspectos mais importantes de um modelo: o **significado dos conceitos**. Uma solução é complementar a UML com outros recursos. Por exemplo, **adicionar texto aos diagramas**, para tentar explicar um aspecto importante do domínio.*

l.bertholdo@ifsp.edu.br

Criação da Linguagem Ubíqua

- Também é possível representar o modelo diretamente no **código**, uma abordagem amplamente defendida pela comunidade XP, já que um código bem escrito pode ser bastante comunicativo. Contudo, essa abordagem pode não ser eficaz.

- Embora a estrutura de um código, os nomes de classes, métodos e atributos possam falar por si só, nem sempre um código bem feito e que faz a coisa certa consegue expressar claramente um conceito.

Seja qual for a forma de representação escolhida, desconfie de documentos longos. Eles podem se tornar obsoletos antes de serem concluídos e será difícil mantê-los sincronizados com o modelo.

Referências

- EVANS, Eric. **Domain-Driven Design:** Atacando as complexidades no coração do software. 3. ed. Rio de Janeiro: Alta Books, 2016.

- MARINESCU, Floyd; AVRAM, Abel. **Domain-Driven Design Quickly**. Lulu Press, 2007.