Gerência

Professor: José Eurípedes Ferreira de Jesus Filho
jeferreirajf@gmail.com

Universidade Federal de Jataí – UFJ

Vamos relembrar como é o processo de desenvolvimento de software?

Domain Driven Design

- DDD possui duas premissas:

    1. Na maioria dos projetos de software o **principal foco deve ser o domínio** e a lógica do domínio;

    2. Designs de domínios complexos deve se basear em um modelo;

Domain Driven Design

- O DDD é:

  - Uma maneira de pensar e um conjunto de prioridades, voltado para a aceleração de projetos de software que precisa lidar com domínios complexos.

Domain Driven Design

- DDD precisa de duas condições para ser aplicado:

    1. Desenvolvimento interativo - Base do desenvolvimento Agile.

    2. Desenvolvedores e especialistas de domínio possuem relações estreitas.

Domain Driven Design

- **Um modelo** é uma simplificação da realidade que destaca os aspectos relevantes para resolver o problema e ignora os "detalhes estranhos“.

- Cada programa está relacionado a atividade ou interesse do usuário. Essa área de assunto em que o usuário aplica o programa é o **domínio do software**.

  - Domínios físicos: 1) Passagens aéreas;
  - Domínios intangíveis: 1) Finanças;

Domain Driven Design

- No geral, **o domínio de software** tem pouco haver com computadores. Contudo, há exceções:

    1. Domínio de uma IDE;
    2. Domínio de um SO, etc.

Domain Driven Design

- Um modelo é uma **forma de conhecimento** seletivamente simplificada e conscientemente estruturada.

- Um modelo de domínio **não é o diagrama** em específico, ele é a ideia que o diagrama deseja transmitir.

Domain Driven Design

- "Assim como um cineasta seleciona aspectos da experiência e os apresenta de forma idiossincrática para contar uma história ou se fazer entender, um modelador de domínios escolhe um modelo em particular pela sua utilidade".

Evans, Erick, Domain-Driven Design, 2017, 3ª Ed.

Domain Driven Design

- Três utilidades básicas para determinar a escolha de um modelo:

1. O modelo e o coração do design dão forma um ao outro;
2. O modelo é a espinha dorsal da linguagem utilizada por todos os membros da equipe;
3. O modelo é um conhecimento destilado.

Domain Driven Design

- O coração do software está na sua capacidade de resolver problemas **relacionados ao domínio**.

- A complexidade existente no coração do software é o principal motivo de um software existir e portanto deve ser combatida diretamente.

Domain Driven Design

- Ingredientes de uma modelagem eficaz:

  - Ligando o modelo e a implementação;
  - Cultivar uma linguagem baseada no modelo;
  - Desenvolver um modelo rico em conhecimento;
  - Destilando o modelo;
  - Colhendo ideias e experimentando;

Domain Driven Design

- Modeladores de domínios eficazes digerem conhecimento.

- **Aprendizado Contínuo**: Produz desenvolvedores melhores tecnicamente, melhores na modelagem de domínio mas também com mais conhecimento sobre o domínio específico.

Domain Driven Design – Design rico em conhecimento

- **As atividades e as regras de negócios** são tão fundamentais para um domínio quanto as próprias entidades.

- Portanto, elas devem estar refletidas no modelo, na linguagem e no código.

Domain Driven Design – Conceitos ocultos

- Exemplo de viagens navais e agendamento de cargas. Figura 1.10

Domain Driven Design – Conceitos ocultos

- Vantagens de um design mais explícito:
  - Destaque das regras de negócios importantes tornando-as perceptíveis para todos os envolvidos;
  - Facilidade na compreensão por parte dos especialistas dos domínios, facilitando feedback;

Domain Driven Design - Linguagem

- Um modelo de domínio pode ser a **base da linguagem comum** para um projeto de software.

- O modelo é um **conjunto de conceitos** desenvolvidos pelas pessoas que participam do projeto com termos e relações que **refletem a visão do domínio**. Esses termos e as inter-relações fornecem semântica a uma linguagem que é moldada de acordo com o domínio que ao mesmo tempo é suficiente para o desenvolvimento técnico.

Domain Driven Design - Linguagem

- Especialistas vs Desenvolvedores:
    - Jargões técnicos
    - Jargões específicos
- Problemas na criação de linguagens diferentes dentro do mesmo projeto.

Domain Driven Design – Linguagem Onipresente

- Linguagem Onipresente (*Ubiquitous Language*): **Linguagem comum a todos** os participantes do projeto. Deve incluir as classes, operações e regras de destaque.

- A mudança na linguagem é reconhecida como mudanças no modelo de domínio.

Domain Driven Design – Linguagem Onipresente

- “Use o modelo como espinha dorsal da linguagem. Enfrente dificuldades utilizando expressões alternativas que representam modelos alternativos. Faça a refatoração em todos os lugares que a expressão aparece (código, modelos, diagramas, etc). Reconheça que a linguagem onipresente está intimamente relacionada com o modelo.”

Evans, Erick, Domain-Driven Design, 2017, 3ª Ed.

Domain Driven Design – Modelagem em voz alta

- Explorar o modelo utilizando a fala em voz alta, fazendo várias construções a partir das possíveis variações do modelo é uma das melhores maneiras de refinar o modelo.

- *Pidgin*: Linguagem desenvolvida por pessoas com **históricos de linguagem totalmente diferentes** em um determinado contexto.

Domain Driven Design – Modelagem em voz alta

- Falar em voz alta a linguagem ubíqua acaba levando ao *pidgin* do modelo:
    - Naturalmente, passamos a compartilhar a linguagem (e o entendimento) que falamos de uma maneira que nunca seria possível através de desenhos e diagramas.

Domain Driven Design – Uma equipe, uma linguagem

- Se especialistas do domínio não estão entendendo o modelo então há algo de errado com o modelo.

- Os especialistas podem (e devem) utilizar a linguagem do modelo para descrever casos de uso e podem ainda trabalhar mais diretamente com o modelo especificando testes de aceitação.

- Em um processo *Agile*, **os requisitos evoluem** a medida que o projeto anda. Parte dessa evolução deve ser a restruturação dos requisitos de acordo com a linguagem ubíqua.

Domain Driven Design – Documentos de design escritos

- Geralmente os documentos escritos são deixados para trás pela evolução do código ou da linguagem do projeto.

- Diretrizes para avaliar um documento:

    1. Um documento deve complementar o código e a fala;
    2. Um documento não deve tentar fazer o que o código já faz bem (fornecer detalhes).

Domain Driven Design – Documentos de design escritos

- Um documento deve estar envolvido nas atividades do projeto: Será que ele fala a linguagem do projeto (agora)?

- Se um documento não está desempenhando um papel, então ele é um desperdício de forças.

- Deixe que a linguagem onipresente e sua evolução sejam os guias para a escolha dos documentos que vão sobreviver.

Domain Driven Design – Base executável

- Um modelo deve trazer consigo a implementação, o design e a comunicação da equipe.

- Não há necessidade de que os modelos explanatórios sejam modelos de objetos, e geralmente é melhor que não sejam.

Ligando o modelo a implementação

- Do que vale um modelo no papel a não ser que ele ajude diretamente o desenvolvimento do software?

- O DDD exige um modelo que não só seja base inicial do projeto mas que também seja a verdadeira base do design.

- Um software que não possui um modelo como base é, no máximo, uma ferramenta que realiza algumas atividades úteis sem explicar suas ações.

Ligando o modelo a implementação

- O DDD parte em busca de **captar conceitos fundamentais do domínio** de forma abrangente e expressiva e também na descrição de componentes que possam ser implementados e **que resolvam de forma efetiva os problemas impostos ao aplicativo**.

Ligando o modelo a implementação

- Cada objeto do design desempenha um papel conceitual descrito no modelo.

- Faça o design do sistema de forma a refletir o modelo de domínio de maneira literal para que o mapeamento seja óbvio.

- O código é uma expressão do modelo e portanto uma mudança no código pode afetar o modelo.

Ligando o modelo a implementação

- Um modelo precisa ser cuidadosamente elaborado para ser implementável na prática.

- O desenvolvimento se torna um processo interativo de refinamento do modelo, do design e do código.