Aula 02 - Domain-Driven Design e Linguagem Onipresente

Disciplina: Manutenção de Software

Prof. Me. João Paulo Biazotto

UniCesumar

Agenda

- Revisão dos tipos de manutenção
- Domain-driven design
- Linguagem Onipresente
- Atividade

Categorias de Manutenção

- Manutenção Corretiva
- Manutenção Preventiva
- Manutenção Adaptativa
- Manutenção perfectiva ou Evolutiva

Domain-driven design (DDD)

- Abordagem para desenvolvimento de software que visa alinhar o código com o domínio do negócio, priorizando a compreensão e a resolução dos problemas desse domínio.

Domain-driven design (DDD)

- Propõe uma série de conceitos e práticas para melhorar a modelagem e a implementação de um sistema, levando em consideração a complexidade do negócio e a colaboração entre desenvolvedores e especialistas do domínio.

Domain-driven design (DDD)

1. Compreender o domínio do negócio
2. Identificar os conceitos fundamentais (modelagem do domínio)
3. Modelagem do domínio
4. Definição de bounded contexts (contextos delimitados)
5. Design da arquitetura
6. Implementação do código

Compreender o domínio do negócio

- A primeira etapa é adquirir um conhecimento aprofundado do **domínio do negócio**.
- Isso envolve trabalhar em estreita **colaboração** com especialistas do domínio, como usuários, analistas de negócio ou **stakeholders**.
- O objetivo é entender as **regras de negócio**, os processos, as interações e as terminologias utilizadas no domínio em questão.

Identificar os conceitos fundamentais

- Com base na compreensão do domínio do negócio, identifique os **conceitos e as entidades fundamentais** que são essenciais para o sistema.
- Esses conceitos podem ser representados por **classes ou objetos** que encapsulam o comportamento e as informações relacionadas a eles.
- Essa modelagem do domínio é um aspecto crítico do DDD, pois ajuda a estabelecer uma **linguagem comum** entre especialistas do domínio e desenvolvedores.

Modelagem do domínio

- Com base no conhecimento adquirido e na identificação do **domínio**, você **modela o domínio do problema** em termos de entidades, agregados, objetos de valor e regras de negócio.
- Essa modelagem deve **refletir a linguagem do domínio**, ou seja, utilizar termos e conceitos do negócio em vez de termos técnicos. Diagramas, como Diagramas de Domínio ou **Diagramas de Classes**, podem ser utilizados para visualizar a estrutura e as relações entre os elementos do domínio.

Definição de contextos delimitados

- Um **bounded context** (contexto delimitado) é uma fronteira lógica que define um contexto claro e limitado em que um **conjunto específico de entidades e regras** de negócio operam.
- Nesta etapa, você identifica e delimita os bounded contexts no domínio, levando em consideração os limites **de responsabilidade e as interações** entre as partes do sistema.

Design da arquitetura

- Projetar a arquitetura do sistema, identificando as diferentes camadas (como a camada de **aplicação**, a camada de **domínio**) e as formas de comunicação e integração entre elas.
- É importante **considerar os padrões e princípios arquiteturais**, como o princípio de inversão de dependência (Dependency Inversion Principle) e a separação de preocupações (**Separation of Concerns**).

Implementação do código

- Com a arquitetura definida, você pode começar a implementar o código do sistema, seguindo as práticas e padrões de codificação apropriados.
- É fundamental manter o alinhamento com a modelagem do domínio e utilizar a linguagem do domínio na implementação, tornando o código mais expressivo e próximo do problema real.

Linguagem Onipresente

- A "Linguagem Onipresente" (Ubiquitous Language, em inglês) é um conceito introduzido por Eric Evans no livro "Domain-Driven Design" (Projeto Orientado a Domínio).
- A ideia por trás da Linguagem Onipresente é que desenvolvedores e especialistas de um determinado domínio compartilhem uma linguagem comum para descrever e discutir o sistema que estão construindo.

Linguagem Onipresente

- A Linguagem Onipresente é uma linguagem rica em termos de negócio, expressões e conceitos que refletem o domínio em que o software está sendo desenvolvido.
- Ela deve ser compreensível tanto pelos especialistas do domínio quanto pelos desenvolvedores, criando uma ponte de comunicação efetiva entre eles.

Benefícios da Linguagem Onipresente

- **Comunicação eficaz:** Todos os envolvidos no projeto, incluindo especialistas do domínio e desenvolvedores, falam a mesma linguagem, o que facilita a comunicação, a compreensão mútua e evita mal-entendidos.
- **Modelagem do domínio:** A Linguagem Onipresente ajuda a capturar e expressar o conhecimento do domínio no código-fonte. Isso permite que o modelo de domínio seja refletido de forma mais precisa e semântica no software.

Benefícios da Linguagem Onipresente

- **Maior produtividade:** A linguagem compartilhada e compreendida por todos facilita o desenvolvimento, a manutenção e a evolução do software. Isso reduz o tempo gasto em traduções entre diferentes terminologias e melhora a colaboração em equipe.
- **Evolução contínua:** A Linguagem Onipresente evolui junto com o domínio do negócio. À medida que os especialistas do domínio e os desenvolvedores trabalham em conjunto, a linguagem é enriquecida e refinada, permitindo uma modelagem de software cada vez mais precisa e alinhada com as necessidades do negócio.

Benefícios da Linguagem Onipresente

- É importante destacar que a Linguagem Onipresente não é apenas sobre a escolha de palavras, mas também sobre o entendimento profundo do domínio e a criação de um modelo de domínio compartilhado e bem definido.

- Ela é uma parte essencial do Domain-Driven Design (DDD) e pode contribuir significativamente para o sucesso de um projeto de software.

DÚVIDAS?