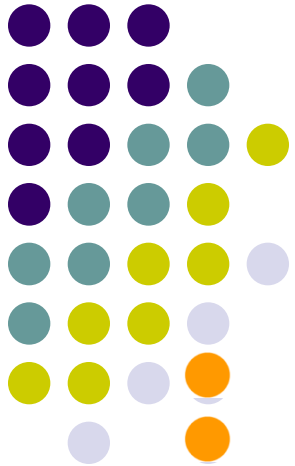
Arquitetura de Software

Modelo de Domínio

Domain & Model Driven Design, Linguagem Ubíqua

José Motta Lopes
josemotta@bampli.com



Agenda

- Domain-Driven Design
- Ingredientes da Modelagem
- Domain-Driven Team
- Ciclo do Processo
- Escolha de um Modelo
- Coração do Software
- Standard Patterns
- Model-Driven Design
- Smart UI anti-pattern
- Ubiquitous Language
- Design Flexível
- Intention Revealing Interfaces
- Side-Effect-Free Functions & Assertions
- Conceptual Contours

Domain-Driven Design

- Orientar projetistas de software
- Desenvolvimento de sistemas complexos
- Diversos tipos de negócios e tecnologias
- Decisões sobre projetos
- Desenvolvimento de vocabulário técnico
- Síntese das melhores práticas já aceitas
- Framework para encarar domínios complexos
- Sistematização projetos orientados a domínio

Ingredientes da Modelagem

TRANSFORMAR CONHECIMENTO EM MODELOS VALIOSOS

- Conectar modelo e a implementação: protótipo deve forjar o essencial logo cedo e se manter em sucessivas iterações.
- Cultivar uma linguagem baseada no modelo, entendida por todos, sem necessidade de tradução.
- Desenvolver modelo rico que captura conhecimento de diversos tipos.
- Destilar o modelo. Adicionar conceitos à medida que modelo se torna mais completo. Remover também conceitos que não se provem úteis.
- Experimentar e fazer brainstorm. A linguagem combinada com esboços e debates torna-se um laboratório do modelo.
- Centenas de variações experimentais podem ser exercitadas, tentadas e julgadas.
- À medida que o time percorre os cenários, as falas são um rápido teste de viabilidade do modelo proposto.

Domain-Driven Team

- Abordagem de projeto orientado a domínio
  - É bom para o desenvolvedor individual.
  - Melhor se equipe reunida aplica a abordagem.
- Modelo de Domínio movido para centro da discussão
  - Equipe compartilha uma linguagem ubíqua.
  - Enriquece a comunicação entre todos.
- Mantem conexão ao software
  - Produz uma implementação lúcida.
  - Implementação sincronizada com o modelo.
  - Alavanca o desenvolvimento da aplicação.

Ciclo do Processo

Em Ciclo:

- **Colocar o modelo de domínio para funcionar.**
- **Blocos de construção de projeto orientado a modelo.**
- **Refatoramento em direção a uma visão mais profunda.**

Design estratégico para situações específicas:

- **Projetos complexos, escala, etc.**
- **Projetos para grandes organizações**
- **Interação com sistemas externos**
- **Interação com sistemas legados**

Escolha de um Modelo

USOS BÁSICOS QUE DETERMINAM A ESCOLHA DE UM MODELO

- **O modelo e o coração do design se moldam.**
- **O modelo é a espinha dorsal da linguagem usada pelos membros do time.**
- **O modelo é o conhecimento destilado.**

Coração do Software

HABILIDADE DE RESOLVER PROBLEMAS DO DOMÍNIO

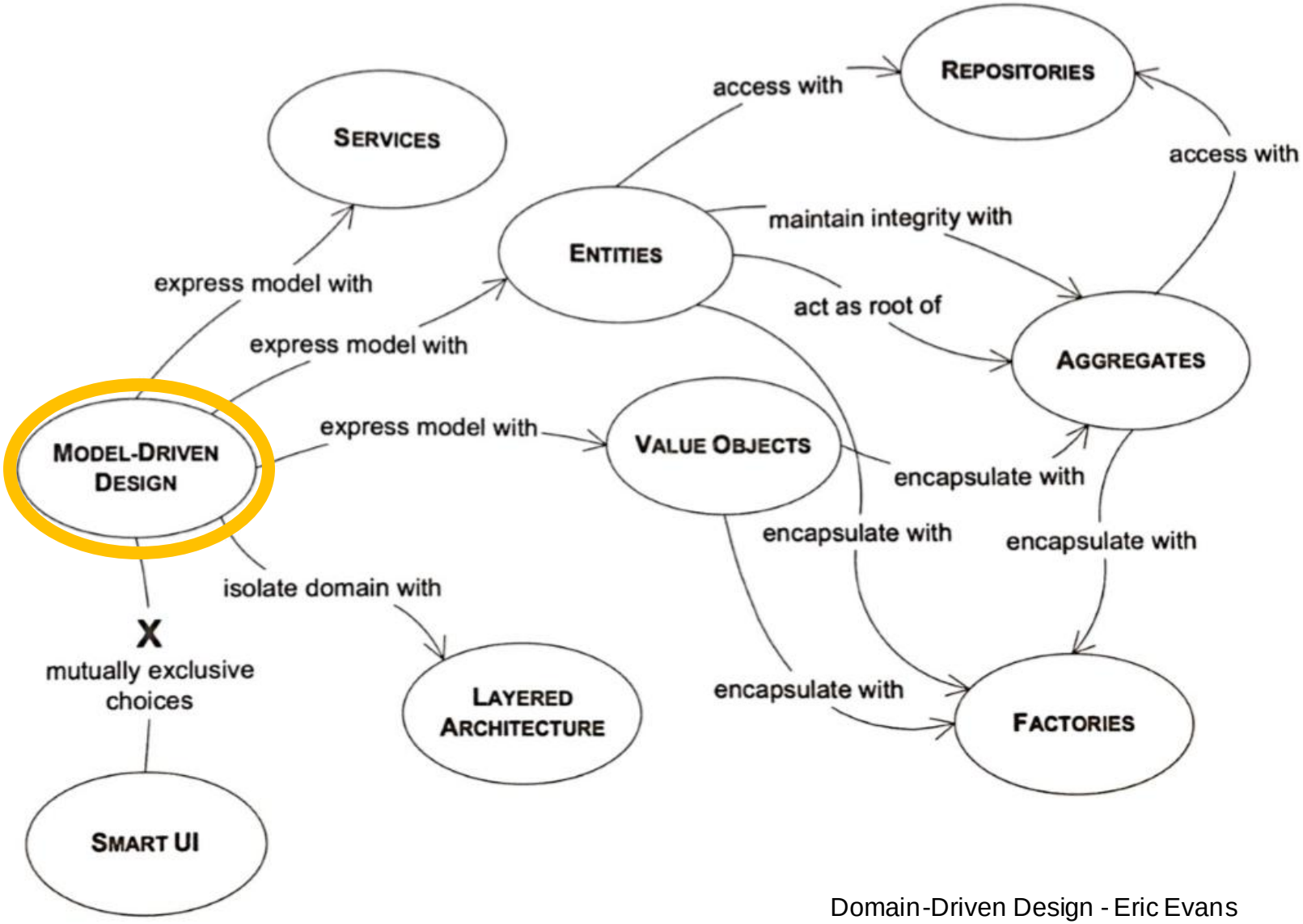
- **Resolver problemas do domínio para os seus usuários.**
- **Tarefa difícil de realizar em sistemas complexos.**
- **Esforço concentrado de pessoas talentosas e habilidosas.**
- **Desenvolvedores deveriam se focar no domínio:**
  - **Objetivo comum de construir conhecimento do negócio;**
  - **Exige talentos dispostos a aprender domínio específico;**
  - **Aprender novos conhecimentos complicados que não parecem adicionar muito à carreira de informática.**
- **Complexidade tem que ser abordada de frente!**
- **Frameworks elaborados tentam resolver problemas de domínio com tecnologia e deixam a modelagem do domínio para outros.**

Standard Patterns

STANDARD PATTERNS TRAZEM ORDEM AO DESIGN

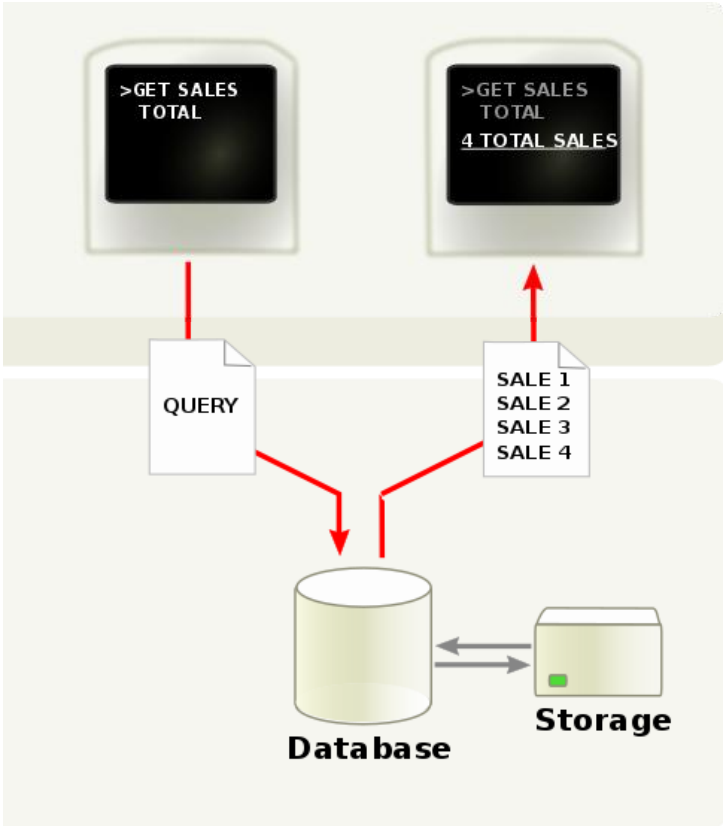
- **Manter implementação nítida e em sincronia com modelo.**
- **Compartilhamento de moldes padrões traz ordem ao design.**
- **Torna mais fácil para membros do time entenderem o trabalho uns dos outros.**
- ***Standard Patterns* se somam à *Ubiquitous Language* que os times usam para discutir o modelo e tomar decisões de projeto.**
- **Desenvolver um bom modelo de domínio é uma arte.**
- **Porém, o projeto e a implementação prática dos elementos individuais de um modelo pode ser relativamente sistemática.**
- **Definir elementos do modelo de acordo com certas distinções aguça seu significado.**
- **Seguir patterns comprovados ajuda a produzir um modelo prático de se implementar.**

Model-Driven Design



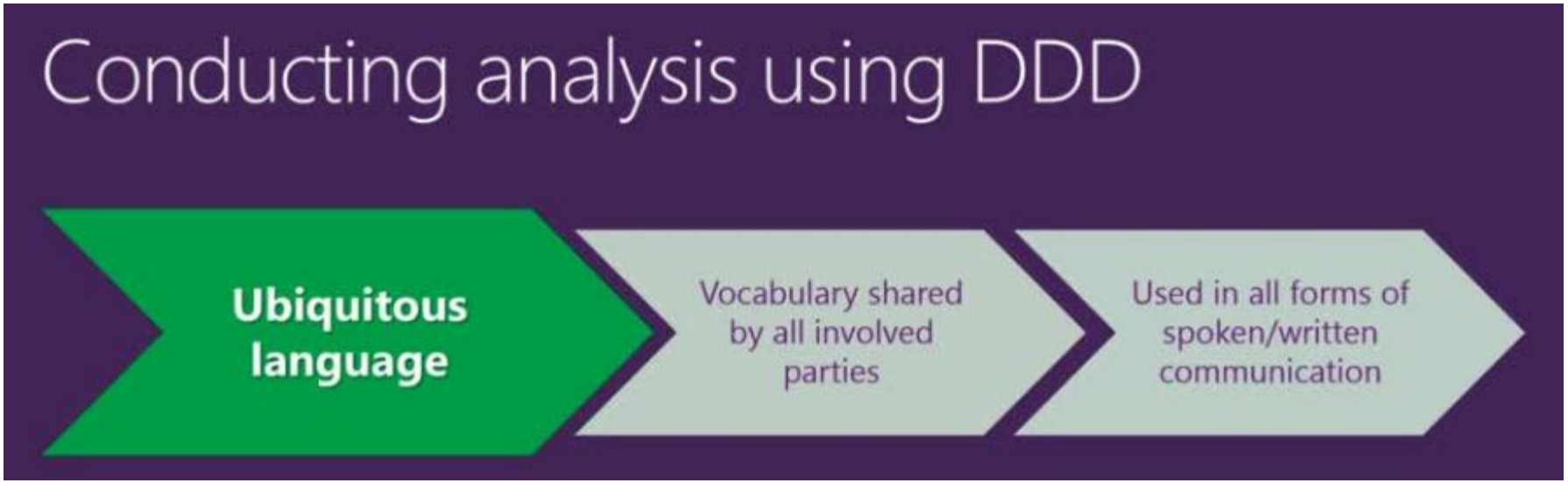
Smart UI anti-pattern

- Projetos simples e pequenos
- Lógica de negócios movida para a interface do usuário.
- DB compartilha o repo
- Geração automatizada UI



Ubiquitous Language

Usar o vocabulário de um determinado domínio de negócios em discussões de **design** também, não apenas em discussões sobre os requisitos de um produto de software, é uma abordagem de design descrita em " *Domain Driven Design* " por **Eric Evans** . Além de "linguagem onipresente" há outras técnicas complementares, mas a intenção principal é essa linguagem.



https://site.gitscrum.com/agile-glossary-ubiquitous-language/

Ubiquitous Language

> *Ao usar a linguagem baseada em modelo de forma generalizada e não ficar satisfeito até que ela flua, abordamos um modelo que é completo e compreensível, composto de elementos simples que se combinam para expressar idéias complexas.*
>
> *...*
>
> *Especialistas de domínio devem se opor a termos ou estruturas que são inadequados ou inadequados para transmitir a compreensão do domínio; os desenvolvedores devem observar a ambigüidade ou inconsistência que atrapalhará o design.*
>
> *- Eric Evans*

https://martinfowler.com/bliki/UbiquitousLanguage.html

Linguagem Ubíqua

ATRITO CONSTANTE EM DESENVOLVIMENTO DE SOFTWARE

Vocabulários Técnicos Diferentes

- **Domínio do Negócio**
  - **Linguagem específica para cada tipo de negócio.**
  - **Especialistas em negócio trabalham com ela.**

- **Desenvolvedores:**
  - **Enquadram trabalho com algoritmos e computação.**
  - **Não há equivalente direto no vocabulário de negócios.**

Linguagem Ubíqua

HABILIDADE DE RESOLVER PROBLEMAS DO DOMÍNIO

- **Especialistas do domínio tem:**
  - **Entendimento limitado do jargão técnico utilizado em desenvolvimento de software;**
  - **Utilizam jargão próprio específico de sua área de trabalho.**
- **Desenvolvedores entendem e discutem o sistema em termos funcionais e descritivos:**
  - **Desprovido do significado transmitido por especialistas de domínio;**
  - **Adicionam abstrações que satisfazem seu projeto mas não são entendidas pelos especialistas de domínio;**
  - **Desenvolvedores de diferentes partes do problema elaboram conceitos próprios de design e descrição do domínio.**

Linguagem Ubíqua

COMO LIDAR COM UMA DIVISÃO LINGUÍSTICA?

- **Especialistas do domínio descrevem vagamente o que querem.**
- **Desenvolvedores lutam para entender vagamente o domínio novo.**
- **Poucos membros do time se tornam bilingues:**
  - **Passam a ser gargalos do fluxo de informação;**
  - **Suas traduções são inexatas.**
- **Cismas fazem com que membros do time utilizem os termos de forma diferente, sem se dar conta disso.**
- **Malha de traduções confusas reflete falta de uma linguagem comum.**

Linguagem Ubíqua

VOCABULÁRIO DA LINGUAGEM

- **Nomes de classes e operações mais proeminentes.**
- **Termos para discutir as regras explicitadas no modelo.**
- **Complementa com princípios de organização de alto nível impostas ao modelo (escalamento por exemplo).**
- **Enriquecido com nomes de padrões que o time aplica comumente ao modelo de domínio.**

Design Flexível

- **À medida que o programa evolui, desenvolvedores irão rearranjar e reescrever cada parte. Mesmo após anos, o código estará sendo mudado e extendido.**

- **Em um software complexo, a falta de um bom design dificulta a refatoração ou a combinação de elementos.**

- **Duplicação começa a aparecer, pois o desenvolvedor não se julga confiante em prever as implicações completas de um cálculo.**

- **Duplicação é forçada se elementos do design são monolíticos e as partes não podem ser combinadas.**

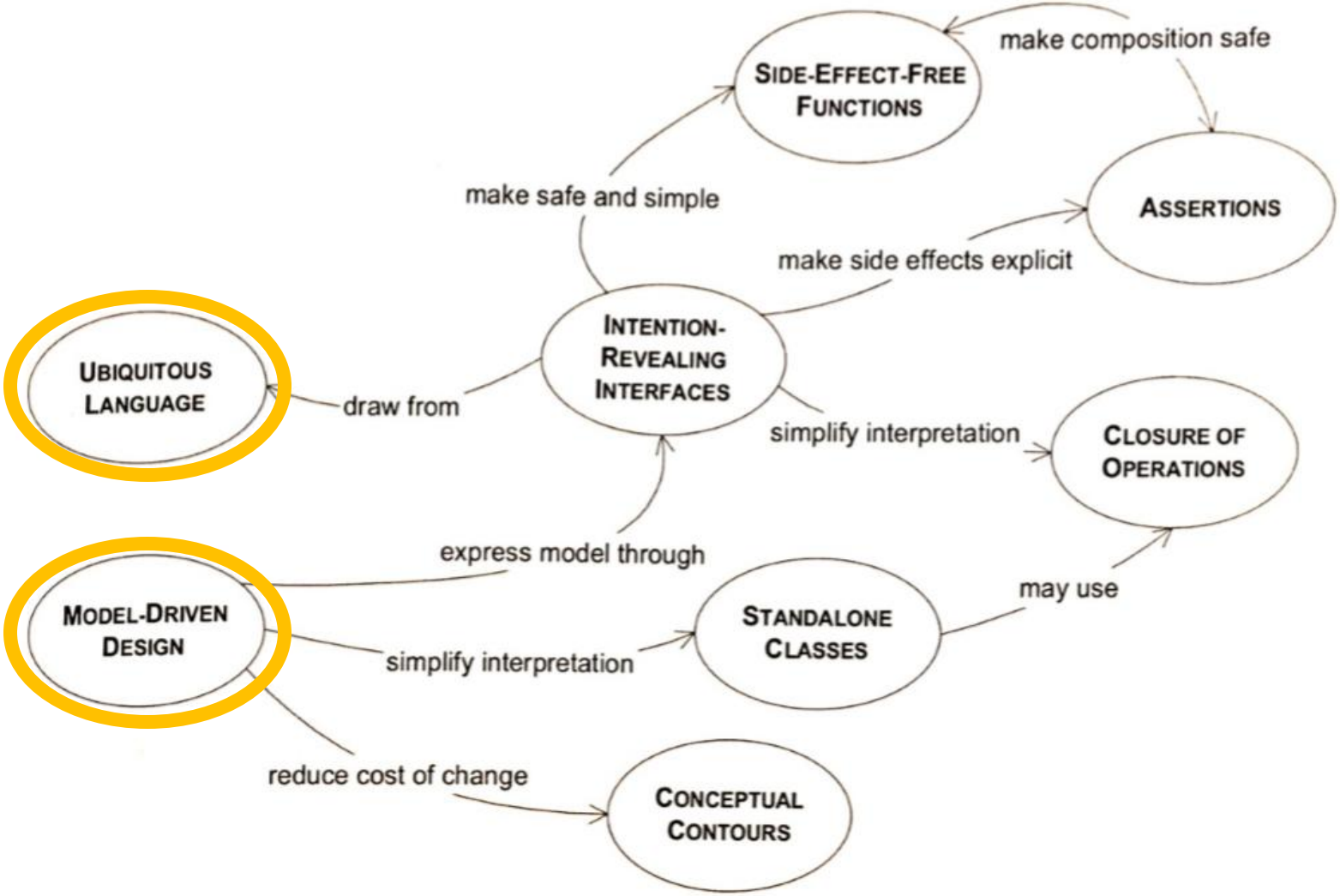
Design Flexível

- **Design flexível complementa a modelagem profunda. Aos explicitar conceitos que estavam implícitos, tem-se a matéria prima.**

- **Através de ciclos iterativos, molda-se o material em algo útil, cultivando um modelo que captura as principais preocupações.**

- **Molda-se um projeto que permita a um desenvolvedor cliente realmente colocar o modelo para funcionar.**

Design Flexível



Domain-Driven Design - Eric Evans

Design Flexível

INTENTION REVEALING INTERFACES

- **Sem uma clara conexão com modelo, é difícil entender o efeito do código ou antecipar o efeito de uma mudança.**
- **Se a interface não informa ao desenvolvedor cliente o que ele precisa saber para usar o objeto de forma eficiente, a maior parte do valor do encapsulamento se perde.**
- **Lutar contra a sobrecarga cognitiva. Aliviar a mente do desenvolvedor cliente dos detalhes sobre como o componente realiza seu trabalho.**
- **Vale mesmo para pessoas que realizam ambos os trabalhos!**
- **Nomeie classes e operações descrevendo seu efeito e propósito, sem referências aos meios utilizados para entregar o que prometem.**
- **Estes nomes devem obedecer à Linguagem Ubíqua, de forma que o membros do time possam rapidamente deduzir seu significado.**

Design Flexível

SIDE-EFFECT-FREE FUNCTIONS & ASSERTIONS

- **Funções livres de efeitos colaterais.**
- **Interações de múltiplas regras ou composições de cálculo se tornam extremamente difícil de prever.**
- **O desenvolvedor deve antecipar os resultados das operações que usa, bem como da implementação de todas as suas delegações.**
- **A falta de abstrações previsíveis:**
  - **Limita a explosão combinatória e**
  - **Rebaixa o teto da riqueza de comportamento viável de se ter.**
- **Coloque tanta lógica quanto possível em funções e operações que retornem resultados sem efeito colateral observável.**

Design Flexível

CONCEPTUAL CONTOURS

- **Contornos conceituais.**
- **Se elementos do modelo estão embutidos em construções monolíticas, sua funcionalidade fica duplicada. Seu significado é difícil de entender, pois diferentes conceitos se misturam juntos.**
- **Por outro lado, quebrar classes e métodos podem complicar inutilmente, forçando os objetos do cliente a terem que entender como pedaços minúsculos se encaixam.**
- **Pior, um conceito pode se perder completamente. Meio átomo de urânio não é urânio!**
- **Decomponha elementos do design (operações, interfaces, classes e agregados) em unidades coesas.**
- **Leve em conta sua intuição das divisões importantes do domínio.**
- **Alinhe o modelo com aspectos consistentes do domínio.**