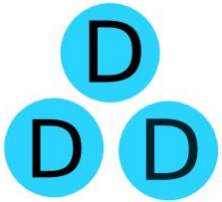
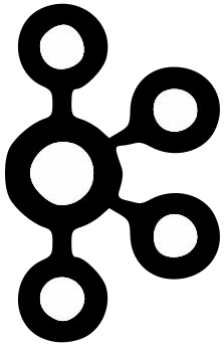
Event Driven Architecture & DDD & Event Storming.





Cléber da Silveira - Arquiteto de Software
Vinicius Soares - Arquiteto de Software

COFFEEANDIT

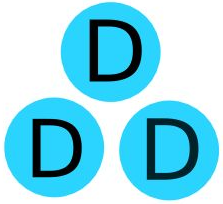
TECHNOLOGY INFORMATION

Agenda

EDA

Porquê?

Para quê?



Domain Driven Design

DEMO

Assuntos indiretos e possíveis.

Sagas

Programação Reativa

Apache Kafka

SRE

Measuring and Managing Reliability

Circuit Breaker

EDA Architecture

EDA Architecture

Mudanças de estado.

Dispare e esqueça.

Atores de eventos (consumidores x produtores).

Acontecimentos no passado.

EDA Architecture

Estados imutáveis;

Diminui acoplamento entre aplicações;

Processos assíncronos;

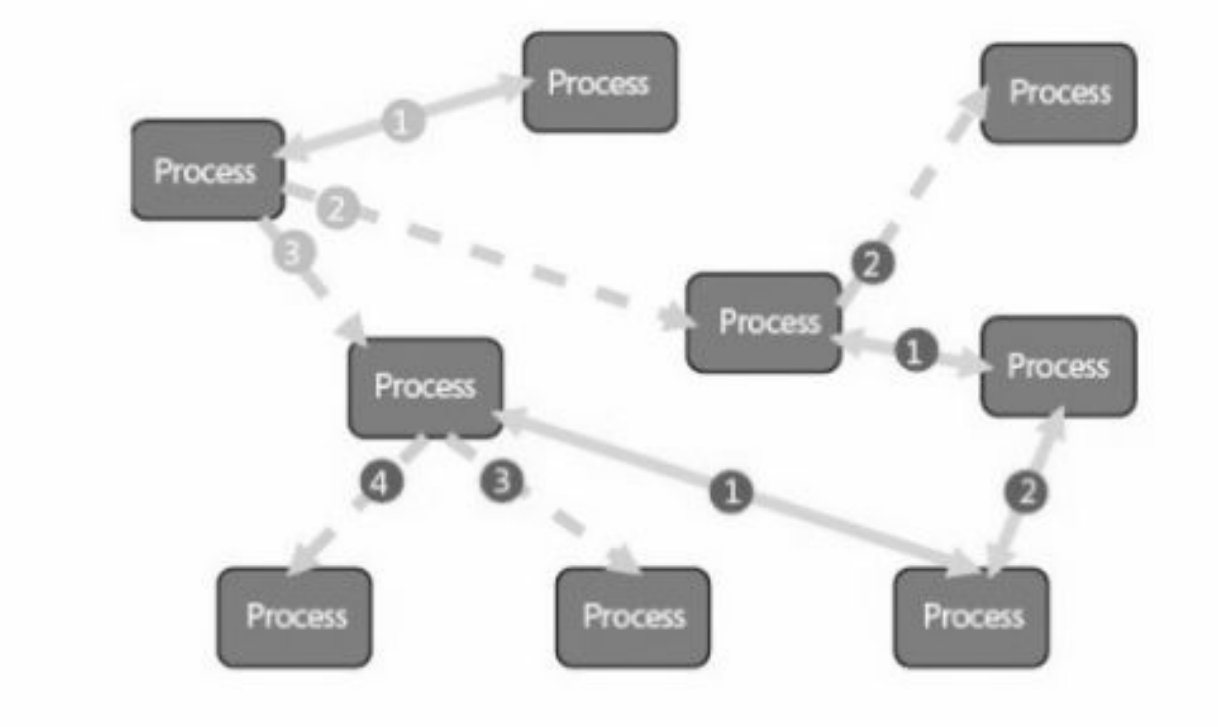
Encaixa perfeitamente com patterns (CQRS / DDD);

Reprodução de estados;

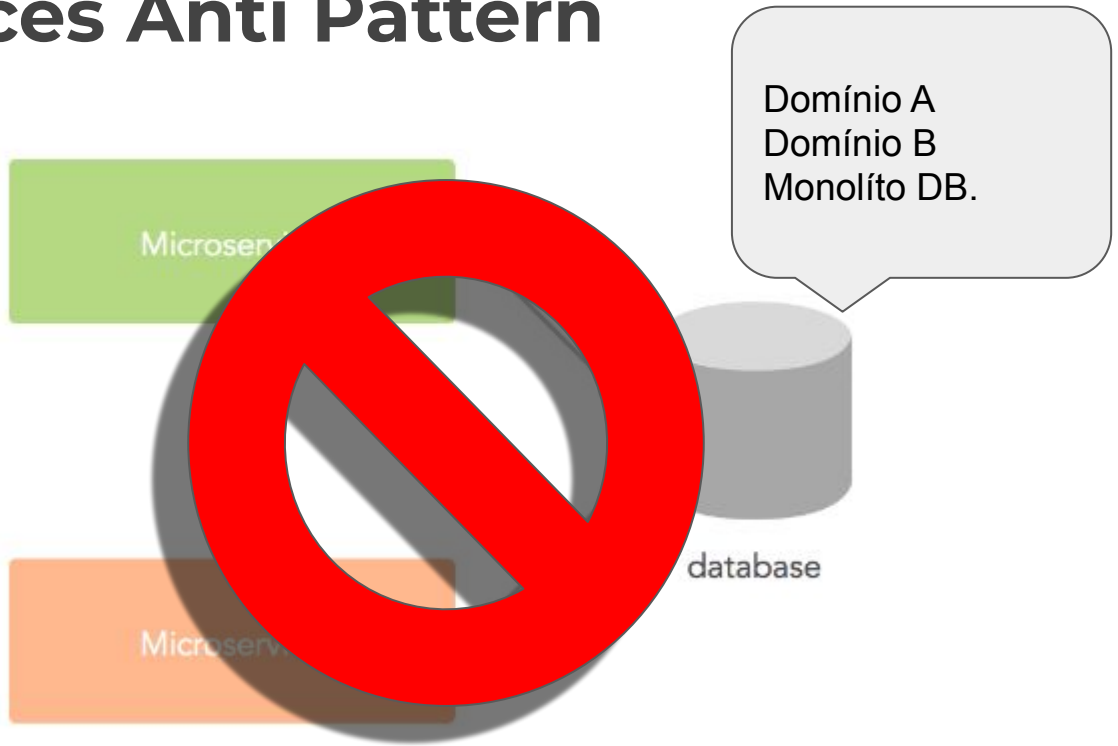
** DDD - Domain Driven Design

** Command Query Responsibility Segregation

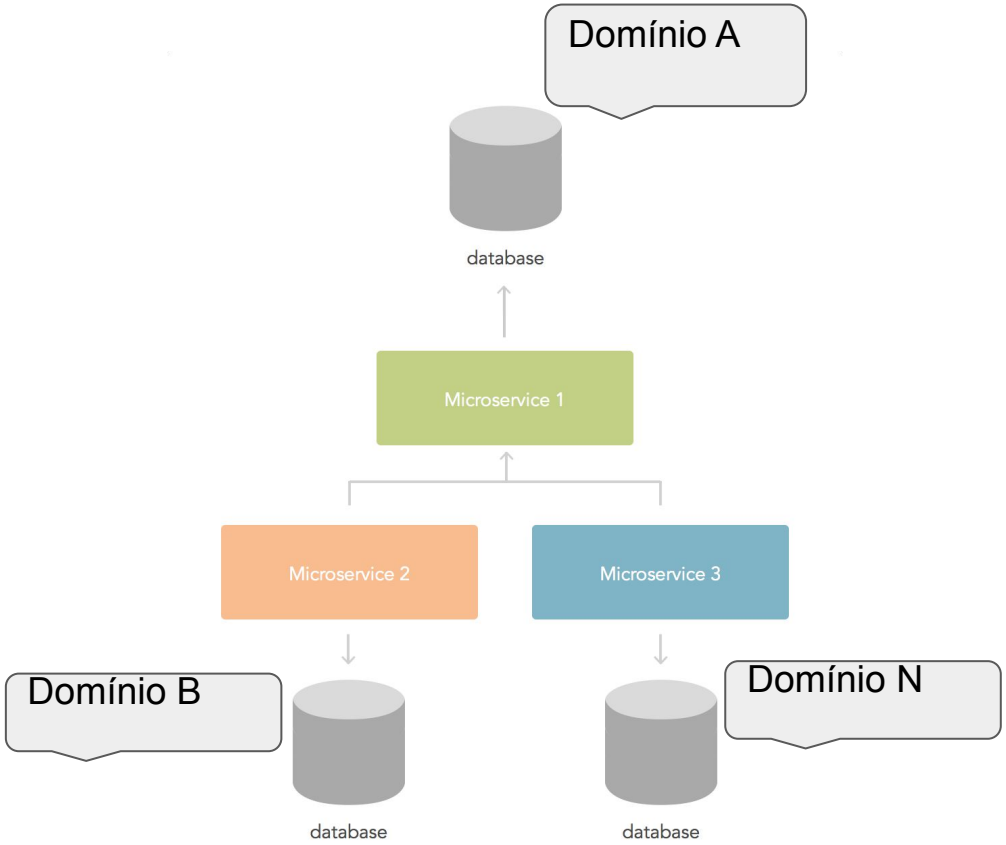
EDA Architecture



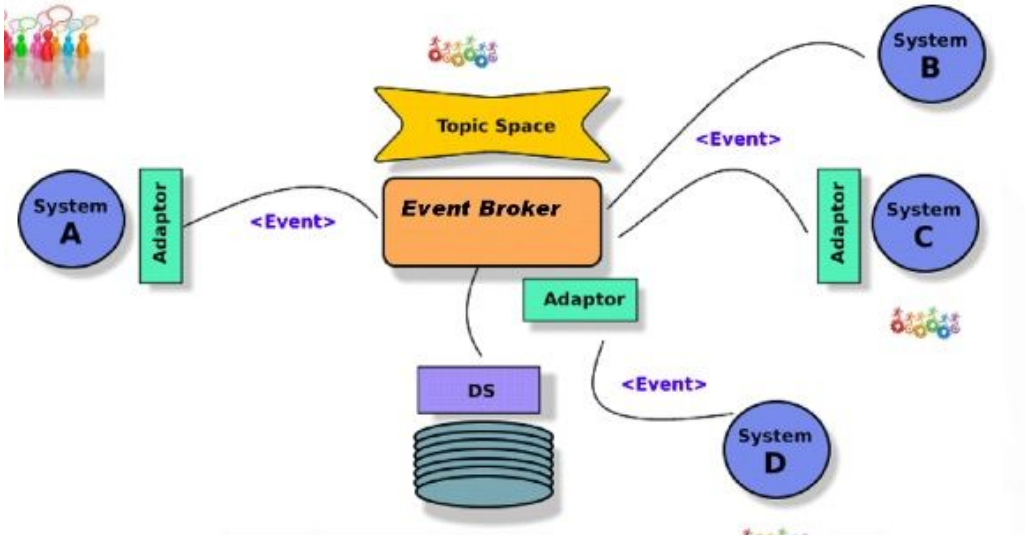
Microservices Anti Pattern



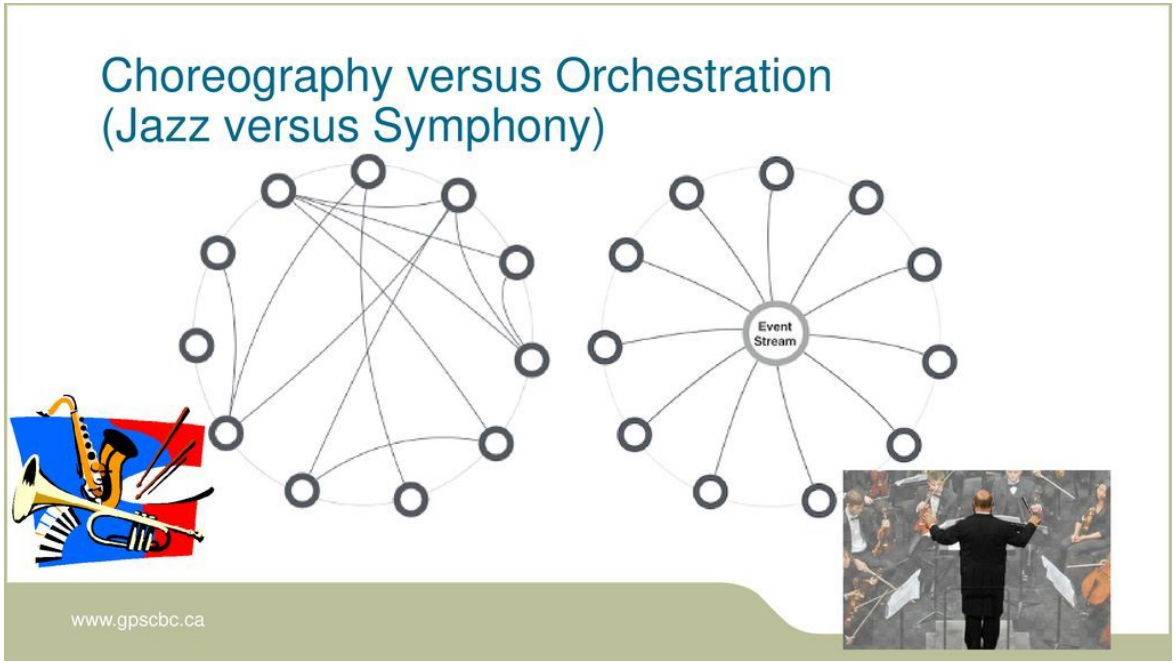
Microservices Pattern



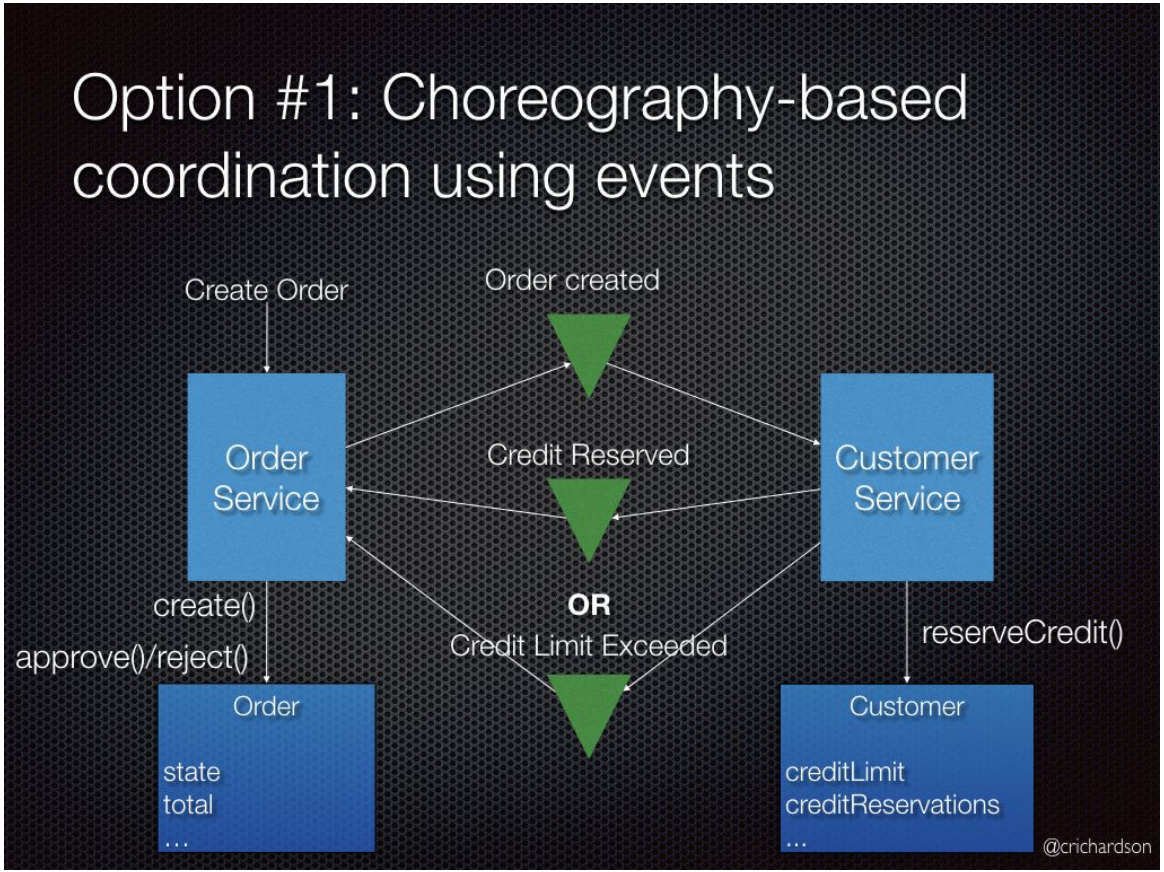
EDA Architecture



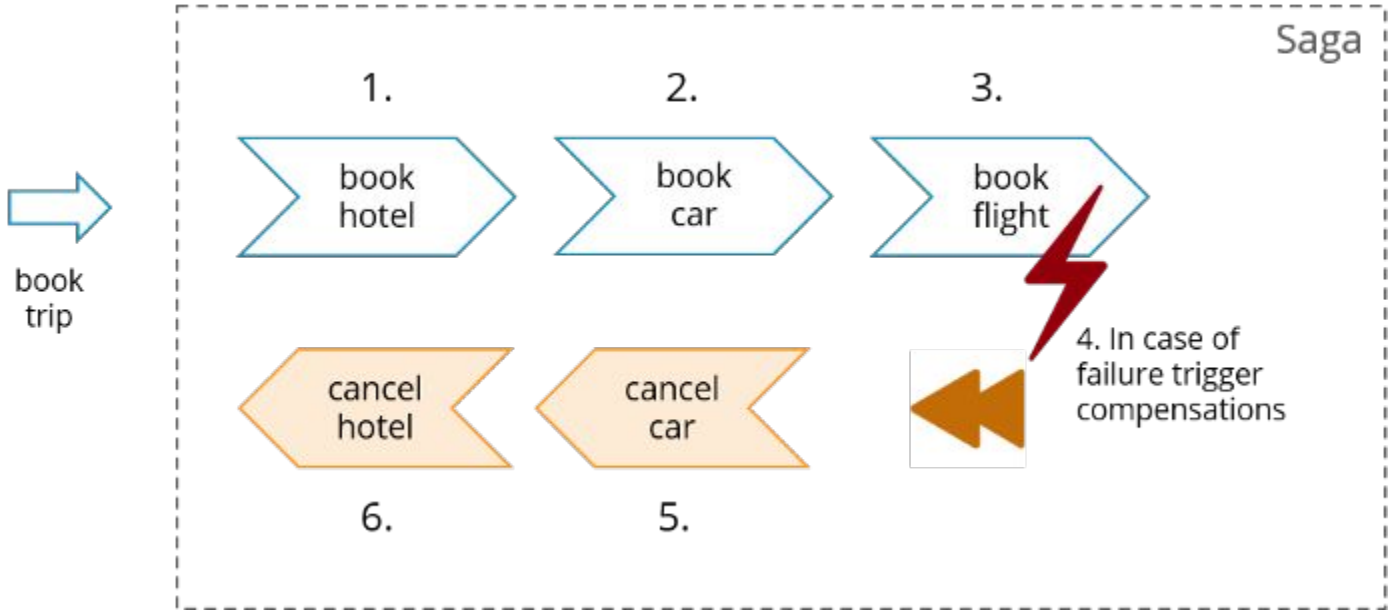
Coreografia versus Orquestração



Saga Pattern - Como atualizar diferentes domínios.



Saga Pattern - Compensar na Falha.



Porquê?

Porquê?

A necessidade da computação mudou.

Usuários querem seus dados agora; querem ver seus tweets agora, confirmar seus pedidos agora, jogos online precisam responder agora.

Aplicações orientadas a alto volumes de dados em tempo real.

Não queremos que nosso software fique bloqueado por um pedido de informações ou aguardando o resultado de uma computação.

Não queremos que nossa aplicação fique parada aguardando algum resultado, mas precisamos exibi-los assim que estiverem prontos.

Porquê?

O comportamento das aplicações evoluiu para lidar com dados empurrados.

Hoje em dia precisamos de ferramentas para **REAGIR** a estes eventos e informações.

Para quê?

Chamadas de serviços (Rest, GRPC, etc.).

Mensagerias.

Abstração para processos síncronos e assíncronos.

Eventos de UI.

Domain Driven Design

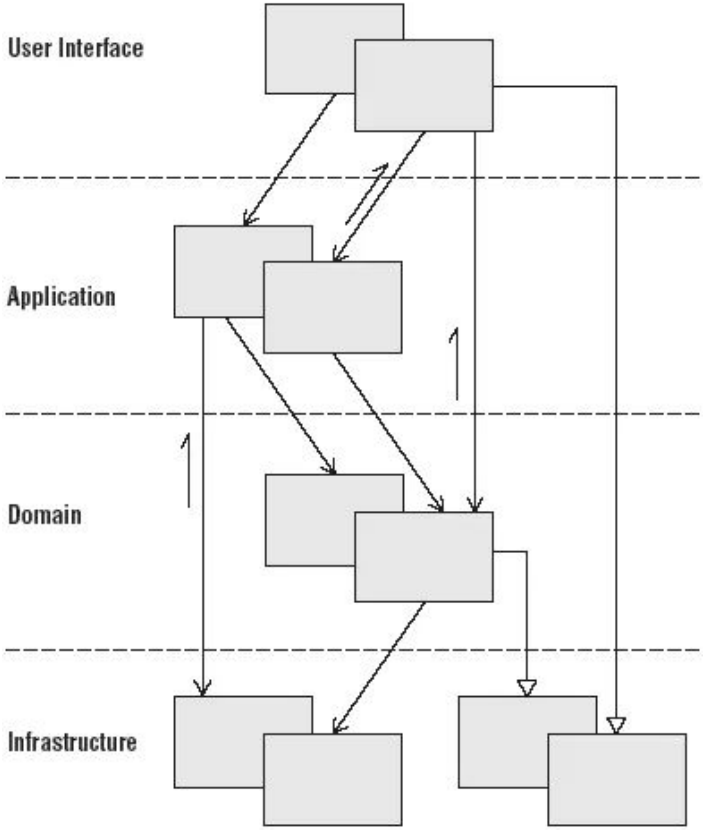
- E uma abordagem de desenvolvimento de software que reúne um conjunto de conceitos, princípios e técnicas cujo **foco** está no **domínio** e na lógica do domínio com o objetivo de criar um **Domain Model** ou (*modelo do domínio*).

- Significa desenvolver **software** de acordo com o **domínio** relacionado ao **problema** que estamos propondo **resolver**.

- O foco da abordagem é criar um domínio que "**fale a língua**" do usuário usando o que é conhecido como linguagem **Ubíqua**(*ubiquitous language* ou *linguagem Comum,Onipresente*)

Domain Driven Design

User Interface Layer	Accepts user commands and presents information back to the user
Application Layer	Manages transactions, translates DTOs, coordinates application activities, creates and accesses domain objects
Domain Layer	Contains the state and behavior of the domain
Infrastructure Layer	Supports all other layers, includes repositories, adapters, frameworks etc.



Domain Driven Design

- Linguagem **Ubíqua** (*linguagem comum*) entende-se que ao trabalhar com DDD devemos conversar usando uma **mesma língua**, em um único modelo, de forma que o mesmo seja compreendido pelo cliente, analista, projetista, desenhista, testador, gerente, etc. nesta linguagem, que seria a linguagem usada no dia a dia.

  Quais as vantagens em usar DDD ?

1. O código fica **menos acoplado** e mais coeso.
2. O negócio é melhor **compreendido** por todos da equipe o que **facilita** o **desenvolvimento**.
3. Alinhamento do **código** com o **negócio**.
4. Favorecer **reutilização**.
5. Mínimo de acoplamento.
6. Independência da Tecnologia.

Event Storming

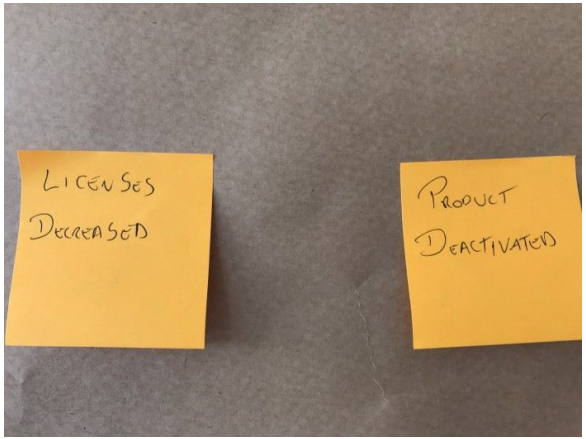
- Event Storming é uma técnica de design rápido que engaja especialistas do domínio de negócios com desenvolvedores para que alcancem um ciclo rápido de aprendizagem (aprender o máximo possível no menor tempo possível

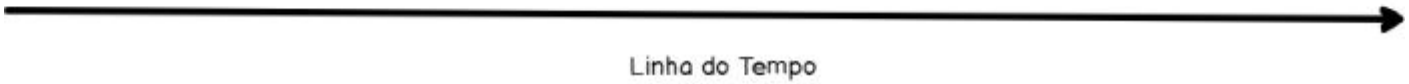
Etapas:

- Mapeando os Eventos
- Identificando os Comandos
- Associando os Aggregates
- Delimitando as Fronteiras do Modelo e
- Identificando Domínios de Negócio

Event Storming - Mapeando Eventos

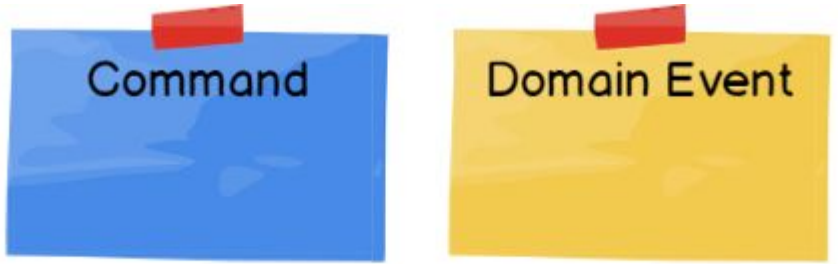
- A primeira etapa do Event Storming consiste em mapear os eventos que ocorrem no domínio que está sendo estudado.
- Um evento é qualquer coisa relevante que aconteceu no passado e tende a ser de simples compreensão para pessoas não técnicas.
- O padrão para descrever o evento é utilizar o **verbo** no **passado** e deve-se tentar mapear todos os eventos.

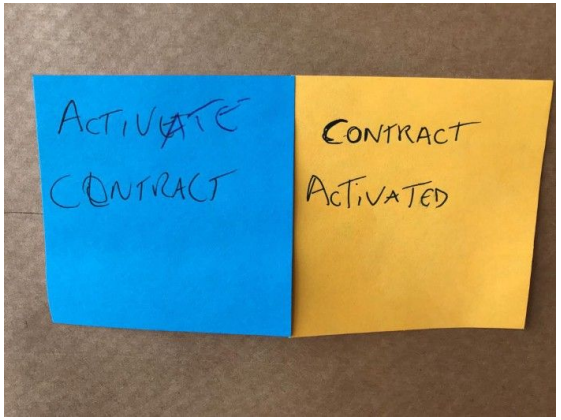


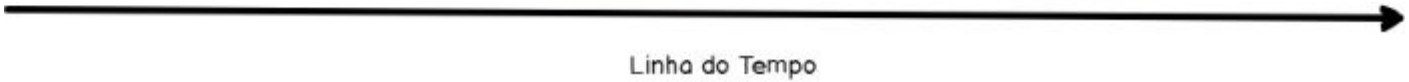


Event Storming - Identificando os Comandos

- Identificação dos **comandos** que geram os eventos.
- Geralmente os comandos estão associados à alguma **ação** do usuário, interação com sistema externo ou gerados por um temporizador/cron.
- **Verbo** na forma **imperativa**
- Deve ser colocado no lado esquerdo do evento que ele gera.
- Durante o processo, é comum identificar que um **comando** pode gerar **vários** eventos.

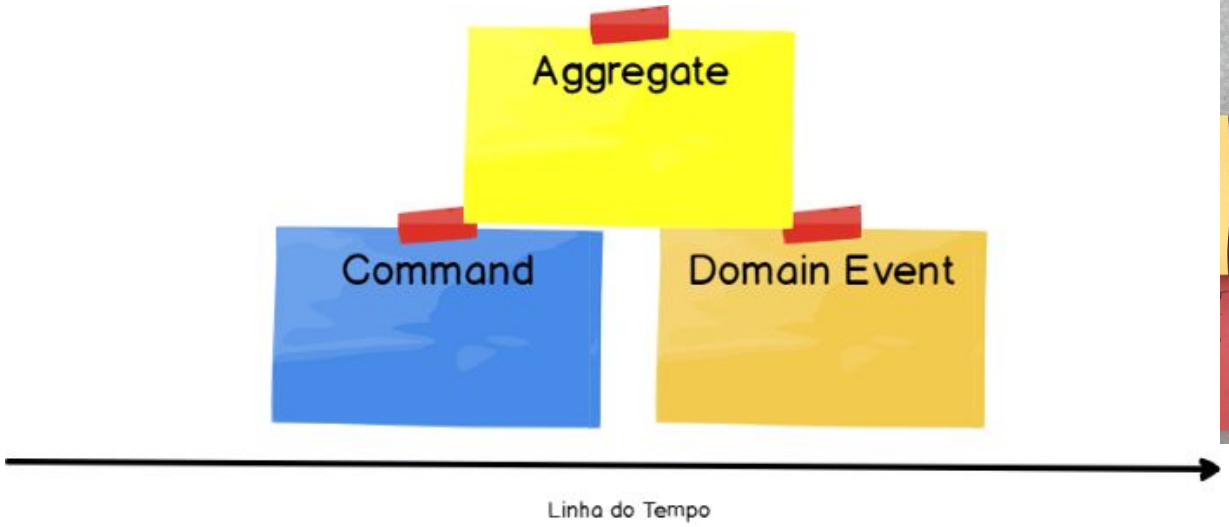


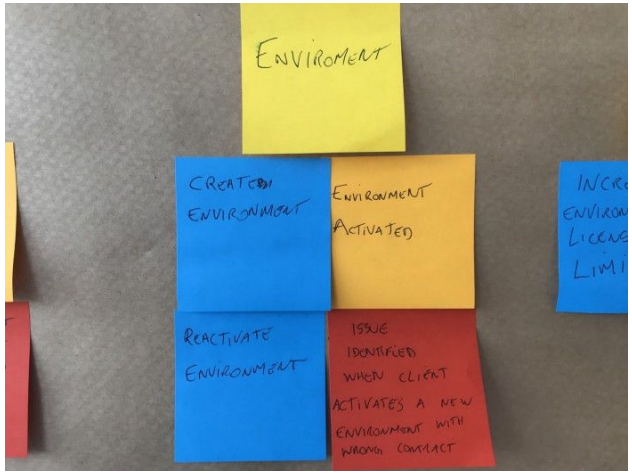




Event Storming - Associando os Aggregates

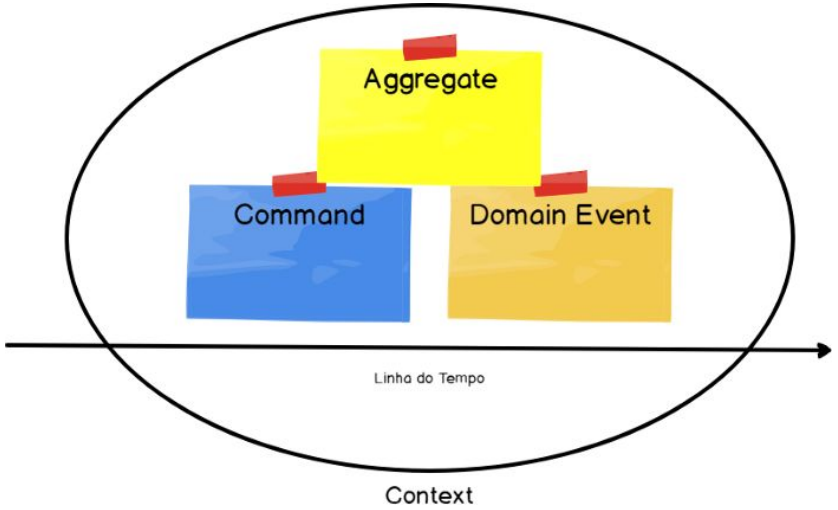
- São a parte do sistema que **recebem** os **comandos** e que geram os eventos, eles são os objetos que **armazenam** os **dados** e são modificados pelos comandos.
- Pode-se **utilizar** o nome **entidade** ou dado quando falar sobre Aggregate.
- Durante o exercício, pode ser que os Aggregates se repitam ao longo da linha do tempo, mas não se deve agrupá-los.





Event Storming - Boundarys

- Podem estar relacionados à **divisões** departamentais.
- Podem ser diferentes **visões** que os especialistas do **negócio** possuem sobre o mesmo conceito.
- Agregadores que são importantes.
- Mapear eventos que “**naveguem**” entre os domínios.



Hands - On

DÚVIDAS?

OBRIGADO!!

INSCREVA-SE