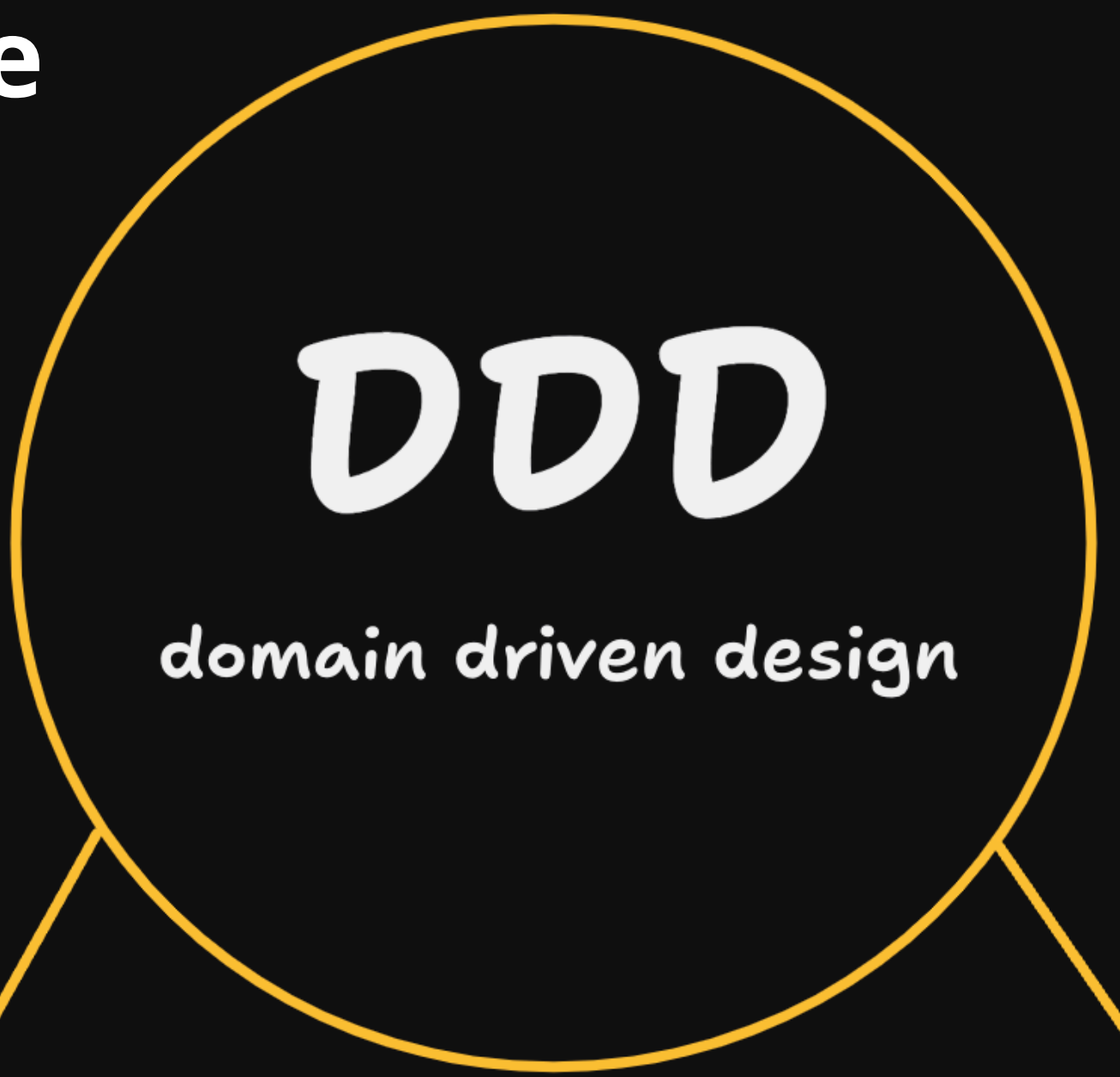
Desvendando DDD

Modelos (Anêmico - Rico)

1 - parte



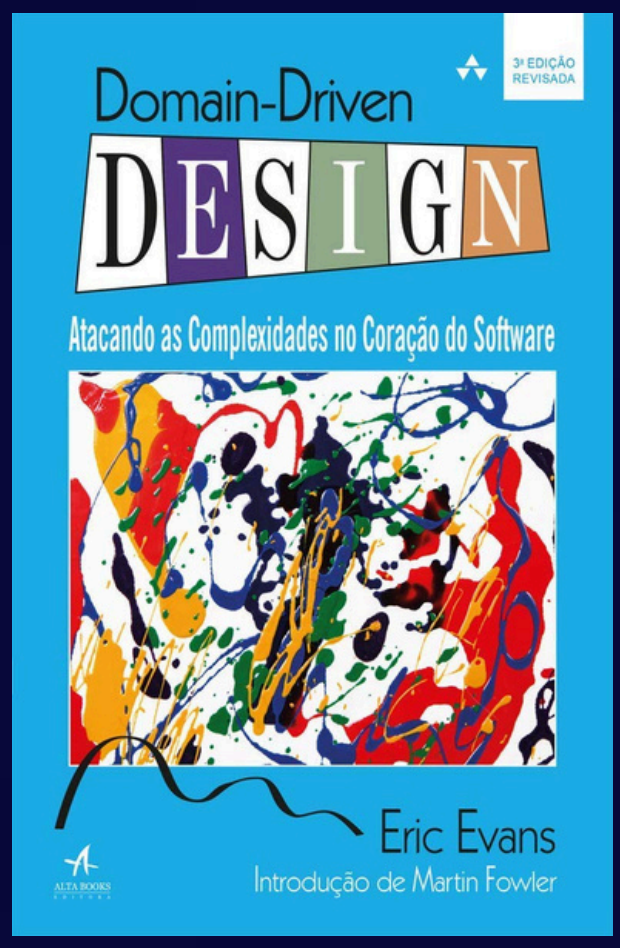
Modelo Anêmico

Modelo Rico

Isaac Gomes

TS

o que é DDD ?



Domain-Driven Design (DDD) é uma abordagem de desenvolvimento de software que foca em entender e **modelar o domínio** do problema, ou seja, a área específica de negócio para a qual o **software** está sendo criado

Isaac Gomes

o que é Domínio anêmico ?

Um **domínio anêmico** é um tipo de **modelo de domínio** onde as **entidades** do sistema são principalmente estruturas de dados simples, sem comportamento ou lógica de negócio significativa. Nesse modelo, as entidades contêm apenas propriedades com métodos de acesso (**getters e setters**), mas não possuem métodos que encapsulam ou executam lógica de negócio. Toda a lógica e regras de negócio são implementadas fora das entidades, geralmente em serviços ou **controladores**.

Isaac Gomes

TS

o que é Domínio anêmico ?

```
class BankAccount {
  constructor(
    private id: string,
    private email: string,
    private cpf: number,
    private name: string,
    private balance: number,
    private creditLimit: number,
    private creditUsed: number
  ) {}

  getEmail(): string {
    return this.email;
  }
1
  setEmail(email: string): void {
    this.email = email;
  }
}
```

note que quando usamos **getters e setters** o estado e comportamento passam a ser controlado fora do meu **dominio**

Isaac Gomes

o que é Domínio Rico ?

Um **domínio rico** é um conceito na modelagem de software onde as entidades de domínio **encapsulam** tanto os dados quanto o **comportamento associado** a esses dados. Isso significa que as **regras de negócio**, **validações** e operações relacionadas estão todas contidas dentro das próprias entidades, em vez de estarem **espalhadas** por outras **camadas**, como serviços ou **controladores**.

Isaac Gomes

TS

o que é Domínio Rico ?

```
class BankAccount {
  constructor(
    private id: string,
    private email: string,
    private cpf: number,
    private name: string,
    private balance: number,
    private creditLimit: number,
    private creditUsed: number
  ) {}

  getEmail(): string {
    return this.email;
  }
1
  setEmail(email: string): void {
    this.email = email;
  }
}
```

como transformar esse domínio anêmico em um domínio rico?

Isaac Gomes

1 - fazer nosso Domínio se Auto-Valida

```
class BankAccount {
  constructor() {this.validate()}


  private validate(): void {
    if (!this.id || !this.email || !this.cpf || !this.name) {
      throw new Error('Todos os campos obrigatórios devem ser preenchidos.');
    }

    if (this.balance < 0) {
      throw new Error('O saldo não pode ser negativo.');
    }

    if (this.creditLimit < 0) {
      throw new Error('O limite de crédito não pode ser negativo.');
    }

    if (this.creditUsed < 0) {
      throw new Error('O crédito utilizado não pode ser negativo.');
    }

    if (this.creditUsed > this.creditLimit) {
      throw new Error('O crédito utilizado não pode exceder o limite de crédito.');
    }
  }
}
```

Um domínio rico se auto-valida, garantindo que ele nunca entre em um estado inconsistente.

Isaac Gomes

TS

2 - Encapsulamento de Lógica de Negócio

```
class BankAccount {
  constructor() {this.validate()}

  deposit(amount: number): void {
    if (amount <= 0) {
      throw new Error('O valor do depósito deve ser positivo.');
    }
    this.balance += amount;
  }
}
```

aqui **validamos** que é uma entrada valida para um **deposito**

Isaac Gomes

2 - Encapsulamento de Lógica de Negócio

```
class BankAccount {
  constructor() {this.validate()}
    withdraw(amount: number): void {
      if (amount <= 0) {
        throw new Error('O valor do saque deve ser positivo.');
      }
      if (amount > this.balance) {
        throw new Error('Saldo insuficiente.');
      }
      this.balance -= amount;
    }
}
```

aqui **validamos** que é um valor valido e um valor que condiz com o que ele tem

Isaac Gomes

2 - Encapsulamento de Lógica de Negócio

```
class BankAccount {
  constructor() {this.validate()}

  useCredit(amount: number): void {
    if (amount <= 0) {
      throw new Error('O valor do crédito deve ser positivo.');
    }
    if (this.creditUsed + amount > this.creditLimit) {
      throw new Error('Limite de crédito excedido.');
    }
    this.creditUsed += amount;
  }
}
```

aqui **validamos** a entrada e se ele tem credito suficiente

Isaac Gomes

TS

2 - Encapsulamento de Lógica de Negócio

```
class BankAccount {
  constructor() {this.validate()}

  payCredit(amount: number): void {
    if (amount <= 0) {
      throw new Error('O valor do pagamento deve ser positivo.');
    }
    if (amount > this.creditUsed) {
      throw new Error('Você não pode pagar mais do que o crédito utilizado.');
    }
    this.creditUsed -= amount;
  }
}
```

aqui validamos a entrada e o credito utilizado para conferir que estão de acordo

Isaac Gomes

TS

ta mas o que ganhamos com esse Domínio rico?

incapsulamento de Lógica de Negócio

Coesão

Auto-Validação

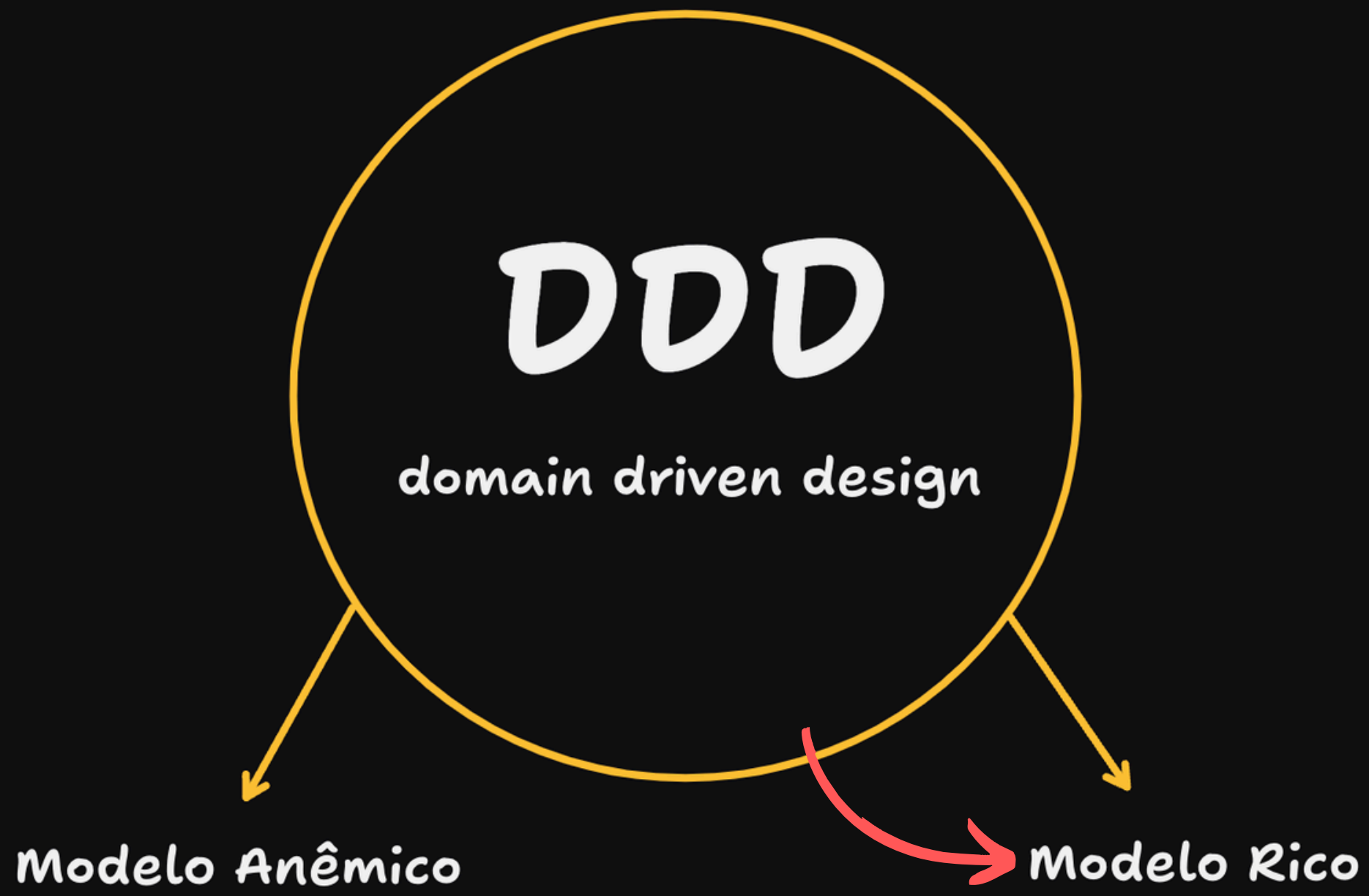
Expressividade

*veja como passamos a ter mais **clareza** e **segurança** no que diz respeito ao nosso **Domínio***

Isaac Gomes

TS

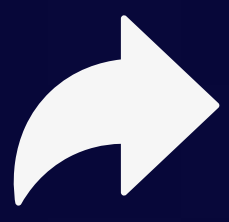
agora esse parte de dominio
passa a fazer mais sentido



Isaac Gomes

Gostou?

Curta



Compartilhe

Salve

Isaac Gomes