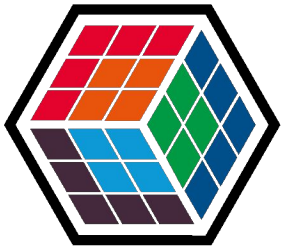


THE DEVELOPER'S CONFERENCE

DESIGN COM INFLUÊNCIA DE TDD

Trilha – Design de Código

Leonardo Amarilho

Agile Coach | Software Engineer

leopoa

leonardo-amarilho

Raphael Monteiro

Software Engineer

raphaelmonteiro15

raphael-monteiro

Afinal o que é

DESIGN DE CÓDIGO

DESIGN DE CÓDIGO

“Criar artefatos que resolvam problemas!”

Características de um bom design:

- ❏ ***auxilia a extensão de código***
- ❏ ***facilita uma alteração***
- ❏ ***simplifica uma remoção***

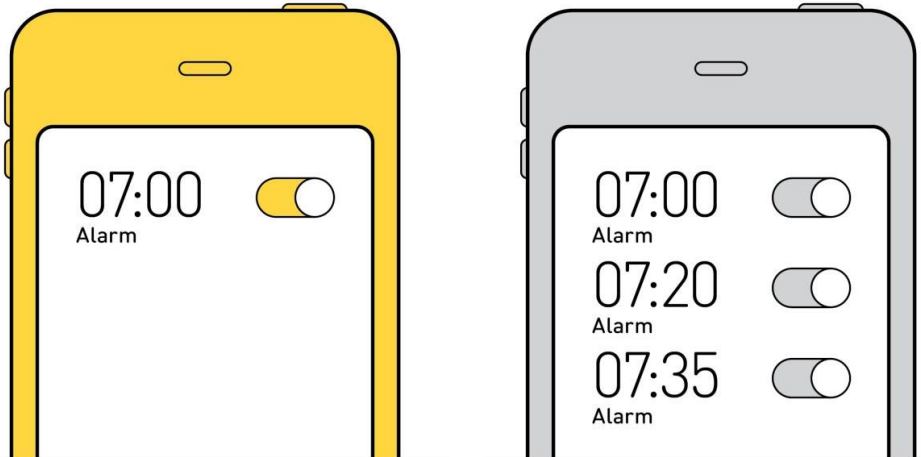
Escrever códigos elegantes e que não resolvem um problema, então não há solução e, portanto, não há design

TEST & CODE

EXISTEM 2 TIPOS DE CATEGORIAS

1. **TDD como uma prática de teste de software, e por consequência avaliam os efeitos dele na qualidade externa do software**

2. **TDD como uma prática de design e estão preocupados com os efeitos dele na qualidade interna do sistema.**



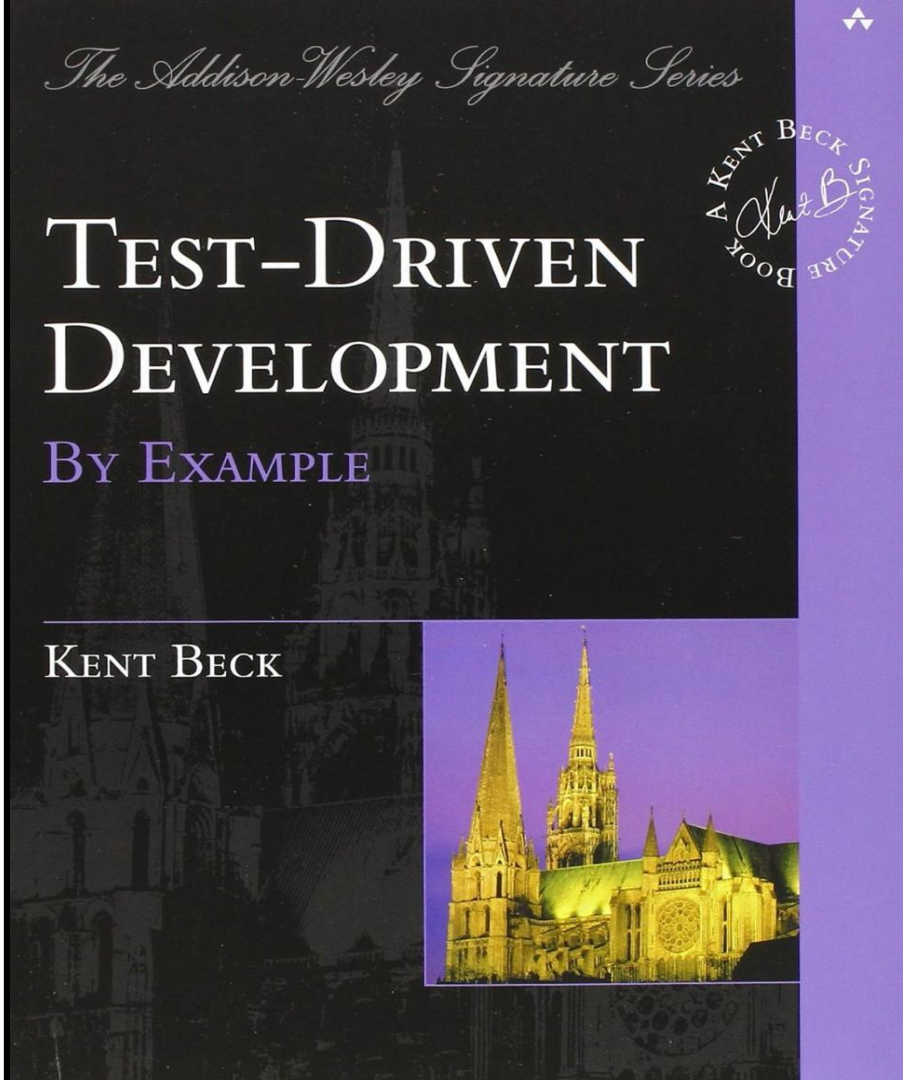
TDD

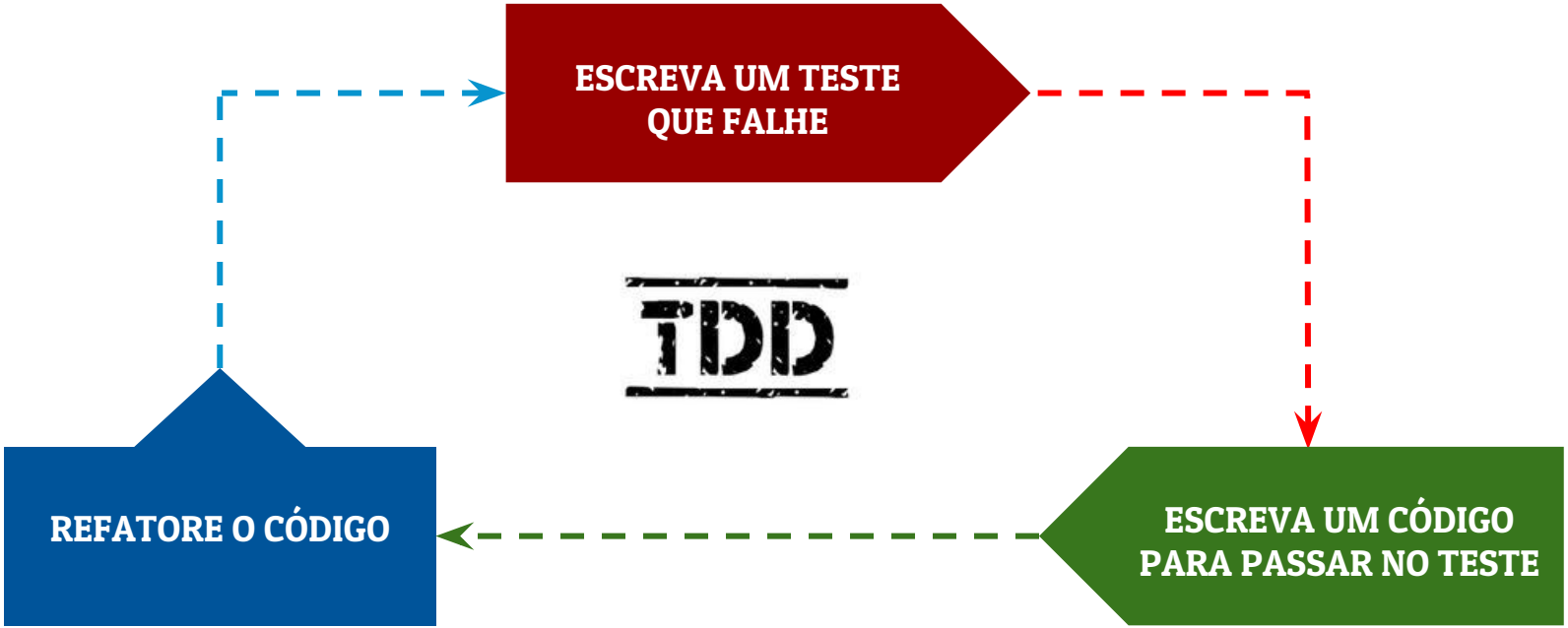
TEST DRIVEN DEVELOPMENT

“Desenvolvimento orientado a testes (TDD) encoraja designs de código simples e que inspiram confiança”

- KENT BECK

TDD se baseia em pequenos ciclos de repetições, onde para cada funcionalidade do sistema um teste é criado antes





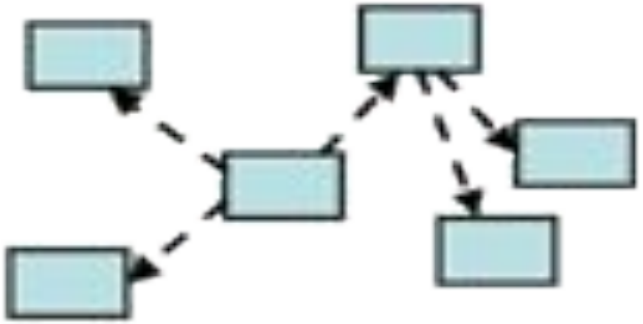
⚠ TODO O CÓDIGO POSSUI BUGS ATÉ QUE SE PROVE O CONTRÁRIO!

INFLUÊNCIA NO DESIGN

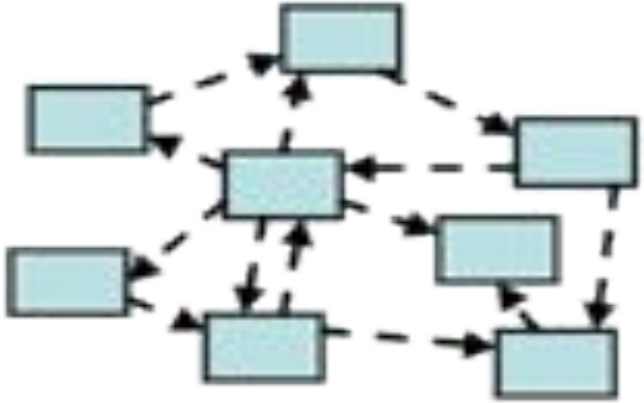
Quando criamos testes antes do código:

- ❏ criamos um código modular
- ❏ auxilia no baixo acoplamento
- ❏ possibilita alta coesão do código
- ❏ facilita o reuso

BAIXO ACOPLAMENTO
ALTA COESÃO



ALTO ACOPLAMENTO
BAIXA COESÃO



```
public User getUser(SearchUserModel searchUserModel){
    Boolean valid;
    String errorMessage = "";
    if(searchUserModel != null){
        if(searchUserModel.getName() != null &&
             searchUserModel.getName().length() > 3){
            if(searchUserModel.getTaxid() != null &&
                 searchUserModel.getTaxid().length()  == 11 ){
                return userRepository.getUser(searchUserModel);
            } else {
                valid = false;
                errorMessage = "Invalid tax id";
            }
        } else {
            valid = false;
            errorMessage = "Invalid name";
        }
    } else {
        valid = false;
        errorMessage = "Invalid search";
    }
    throw new InvalidParametersException(errorMessage);
}
```

HADOUKEN CODE

```
public User getUser(SearchUserModel searchUserModel){
    Boolean valid;
    String errorMessage = "";
    if(searchUserModel != null){
        if(searchUserModel.getName() != null &&
            searchUserModel.getName().length() > 3){
          if(searchUserModel.getTaxid() != null &&
              searchUserModel.getTaxid().length()  == 11 ){
            return userRepository.getUser(searchUserModel);
          } else {
            valid = false;
            errorMessage = "Invalid tax id";
          }
        } else {
          valid = false;
          errorMessage = "Invalid name";
        }
    } else {
        valid = false;
        errorMessage = "Invalid search";
    }
    throw new InvalidParametersException(errorMessage);
}
```

QUAIS SAO OS REQUISITOS DE NEGÓCIOS / CASOS DE TESTES?

Entendendo os requisitos passados:

1. **Validar se usuário de pesquisa é válido**
2. **Validar se usuário de pesquisa possui NAME válido**
    a. **Não pode ser nulo**
    b. **Possuir mais que 3 caracteres**
3. **Validar se usuário de pesquisa possui TAX ID válido**
    a. **Não pode ser nulo**
    b. **Possuir exatamente 11 caracteres**

1. **Validar se usuário de pesquisa é válido**

   a. **Criar o teste e rode ele vai falhar**

```
public void testValidateSearchUserModelIsValid(){
    assertTrue(userValidator.validateSearchUserModel(new SearchUserModel()));
}
```

   b. **Criar o validador da maneira mais simples, para passar o teste**

```
public class UserValidator{
    public boolean validateSearchUserModel(SearchUserModel searchUserModel){
        return true;
    }
}
```

   c. **Rode o teste novamente, quando passar você pode refatorar o código**

```
public class UserValidator{
    public boolean validateSearchUserModel(SearchUserModel searchUserModel){
        return searchUserModel != null;
    }
}
```

RESULTADO FINAL

- ❏ criamos um código modular
- ❏ auxilia no baixo acoplamento
- ❏ possibilita alta coesão do código
- ❏ facilita o reuso

```
public User getUser(SearchUserModel searchUserModel){
    if(userValidator.validateSearchUserModel(searchUserModel))
        throw new InvalidParametersException("Invalid searchUserModel");
    if(userValidator.validateName(searchUserModel.getName()))
        throw new InvalidParametersException("Invalid name");
    if(userValidator.validateTaxId(searchUserModel.getTaxid()))
        throw new InvalidParametersException("Invalid taxId");
    return userRepository.getUser(searchUserModel);
}
```

Comparação entre as soluções

```
public User getUser(SearchUserModel searchUserModel){
    if(userValidator.validateSearchUserModel(searchUserModel))
        throw new InvalidParametersException("Invalid searchUserModel");
    if(userValidator.validateName(searchUserModel.getName()))
        throw new InvalidParametersException("Invalid name");
    if(userValidator.validateTaxId(searchUserModel.getTaxid()))
        throw new InvalidParametersException("Invalid taxId");
    return userRepository.getUser(searchUserModel);
}
```

```
public User getUser(SearchUserModel searchUserModel){
    Boolean valid;
    String errorMessage = "";
    if(searchUserModel != null){
        if(searchUserModel.getName() != null &&
            searchUserModel.getName().length() > 3){
            if(searchUserModel.getTaxid() != null &&
                searchUserModel.getTaxid().length()  == 11 ){
                return userRepository.getUser(searchUserModel);
            } else {
                valid = false;
                errorMessage = "Invalid tax id";
            }
        } else {
            valid = false;
            errorMessage = "Invalid name";
        }
    } else {
        valid = false;
        errorMessage = "Invalid search";
    }
    throw new InvalidParametersException(errorMessage);
}
```

REFATORAÇÃO

É o processo de modificar um sistema de software para melhorar a estrutura interna do código sem alterar seu comportamento externo.

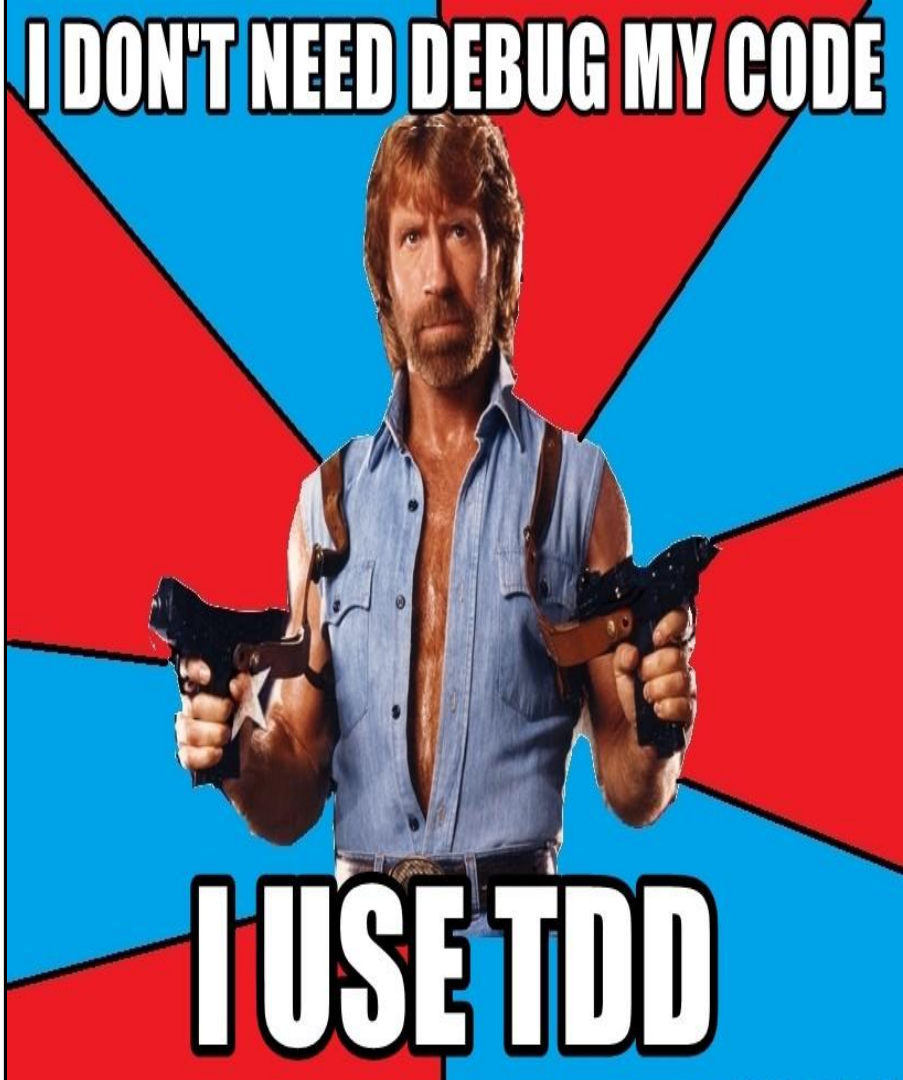
IDENTIFICAÇÃO DE PROBLEMAS DE DESIGN:

- ❑ Dificuldade em escrever testes
- ❑ Excesso de testes para uma unidade
- ❑ Excesso de asserts em um único método
- ❑ Excesso de testes para um componente
- ❑ Efeito em cascata ao modificar um código

Um bom conjunto de teste permite refatorar, pois trazem segurança e permite melhorar seu design ao longo do tempo

---

Fazer uma refatoração em um código sem teste, pode ocasionar efeitos colaterais e/ou em cascata



CONCLUSÕES

THE DEVELOPER'S CONFERENCE

- TDD por si só não cria bons ou maus projetos, mas temos evidências, sugerindo que ele cria designs diferentes
- TDD também auxilia a dividir e cortar histórias, porque é uma técnica que ajuda a pensar no caso de uso mínimo que deve ser implementado para começar
- TDD torna alguns problemas de design mais óbvios

Leonardo Amarilho

Agile Coach | Software Engineer

leopoa

leonardo-amarilho

Raphael Monteiro

Software Engineer

raphaelmonteiro15

raphael-monteiro