



THE DEVELOPER'S
CONFERENCE

**INFLUÊNCIA DE TDD
NO DESIGN DE CÓDIGO**

Leonardo Amarilho

Agile Coach | Software Engineer

 leopoa

 leonardo-amarilho

Raphael Monteiro

Software Engineer

 raphaelmonteiro15

 raphael-monteiro





THE DEVELOPER'S
CONFERENCE

Afinal o que é DESIGN DE CÓDIGO



DESIGN DE CÓDIGO

“Criar artefatos que resolvam problemas!”

Características de um bom design:

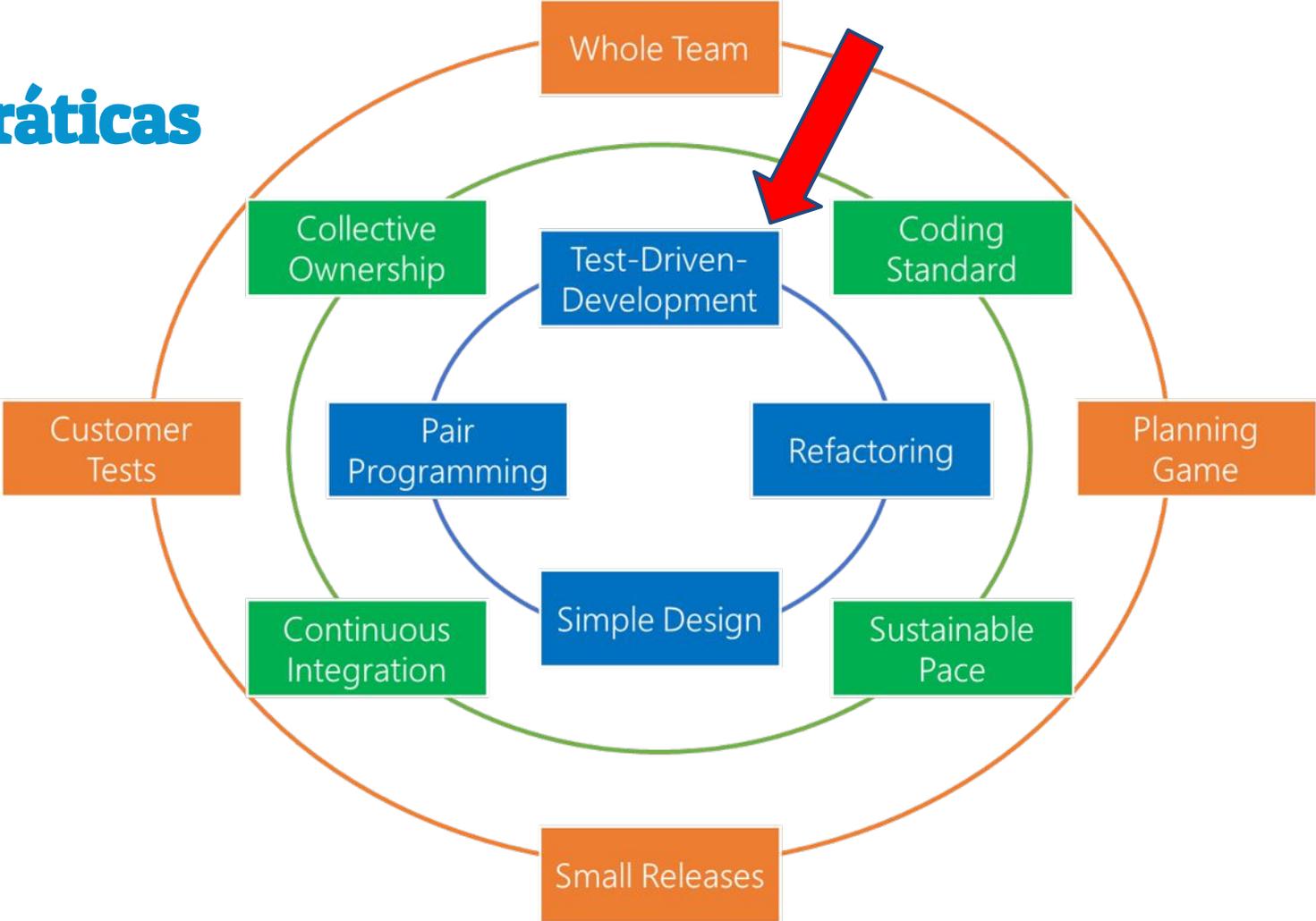
- ❑ *auxilia a extensão de código*
- ❑ *facilita uma alteração*
- ❑ *simplifica uma remoção*

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



**QUAL
RELAÇÃO TDD
COM XP?**

XP Práticas







**A mudança é inevitável,
mas a mudança cria a
necessidade de feedback**

O XP usa o desenvolvimento orientado a testes (TDD) e a refatoração para ajudar a descobrir o design mais eficaz



EXISTEM 2 TIPOS DE CATEGORIAS



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

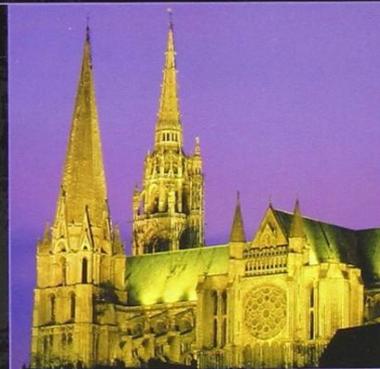
TDD se baseia em **pequenos ciclos** de repetições, onde para **cada funcionalidade** do sistema um **teste é criado antes**

The Addison-Wesley Signature Series

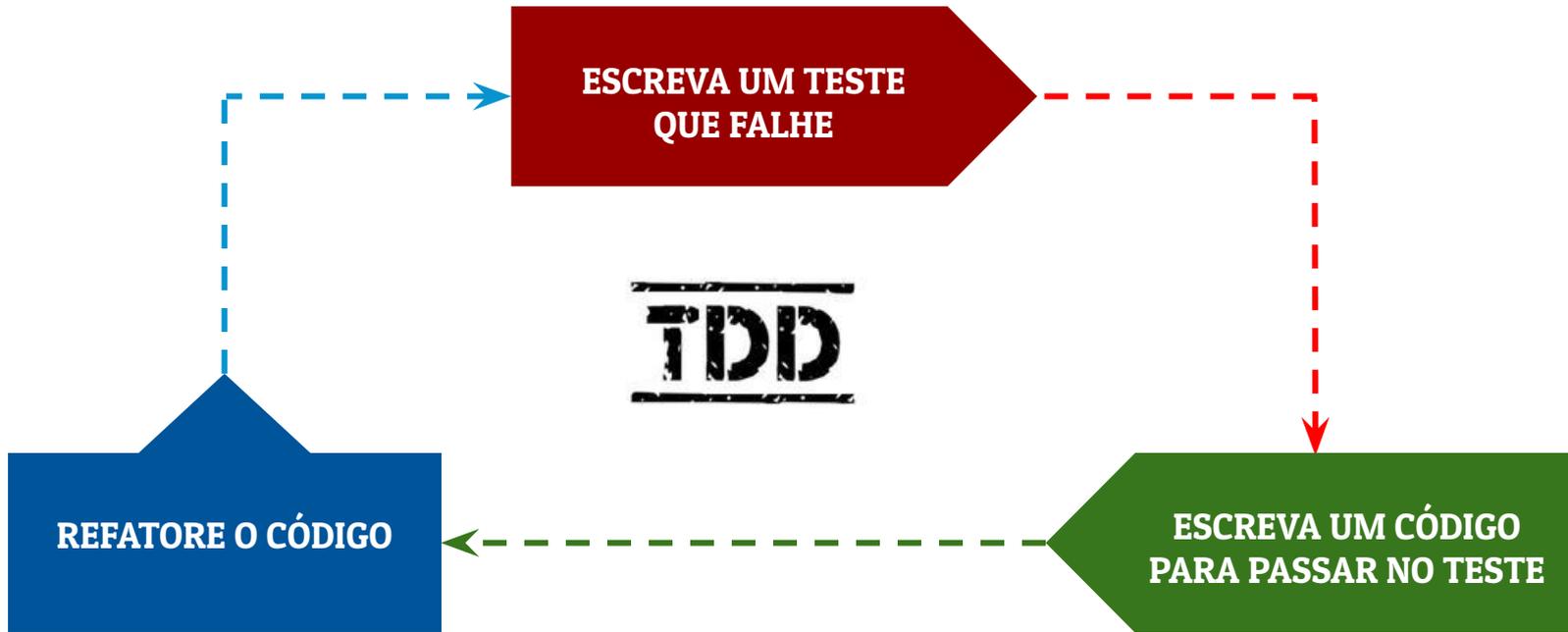
TEST-DRIVEN DEVELOPMENT

BY EXAMPLE

KENT BECK



A KENT BECK SIGNATURE BOOK



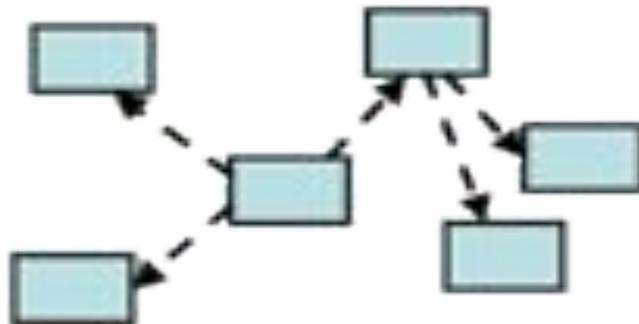
TODO O CÓDIGO POSSUI BUGS ATÉ QUE SE PROVE O CONTRÁRIO!

INFLUÊNCIA NO DESIGN

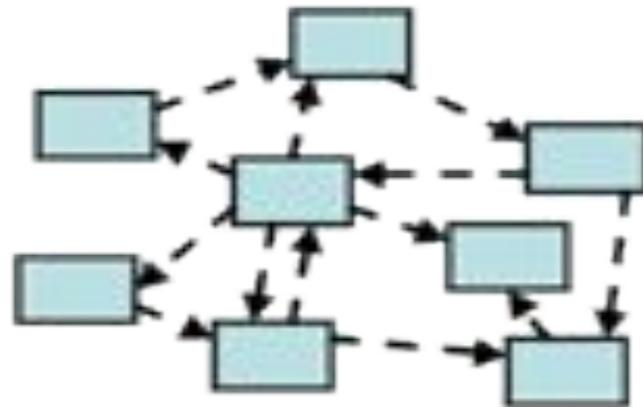
Quando criamos testes antes do código:

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

**BAIXO ACOPLAMENTO
ALTA COESÃO**



**ALTO ACOPLAMENTO
BAIXA COESÃO**



talk is cheap
show me the

CODE

EXEMPLO PRÁTICO

REQUISITOS DE NEGÓCIOS:

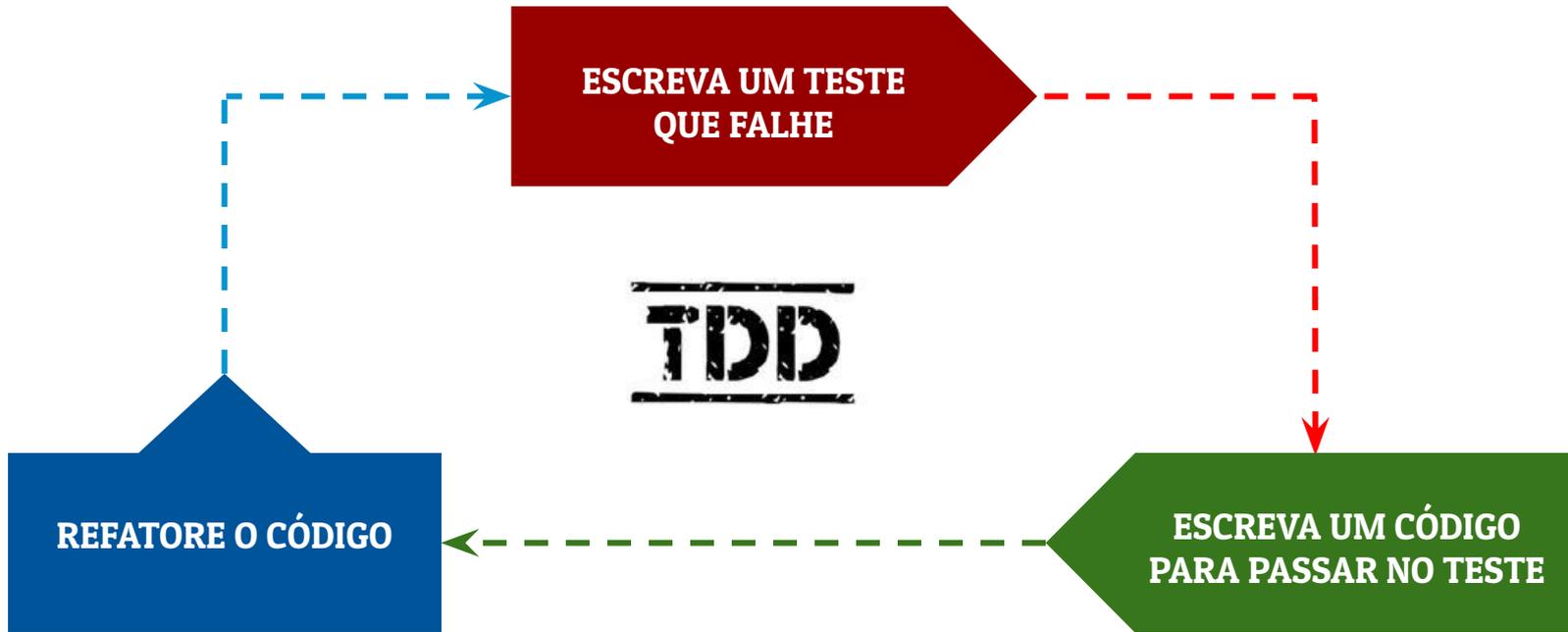
1. Validar se usuário de pesquisa é válido
2. Validar se usuário de pesquisa possui NAME válido
 - a. Não pode ser nulo
 - b. Possuir mais que 3 caracteres
3. Validar se usuário de pesquisa possui TAX ID válido
 - a. Não pode ser nulo
 - b. Possuir exatamente 11 caracteres

```
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);
}
```



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

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) {  
        if(searchUserModel != null) return true;  
    }  
    return false;  
}
```

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);
}
```

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

I DON'T NEED DEBUG MY CODE

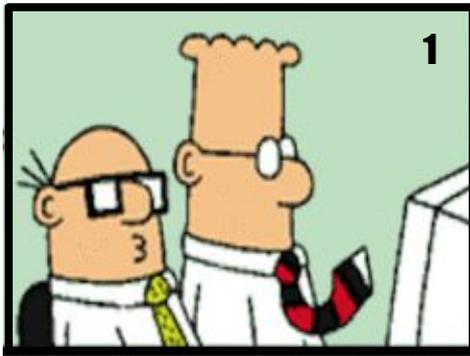


I USE TDD

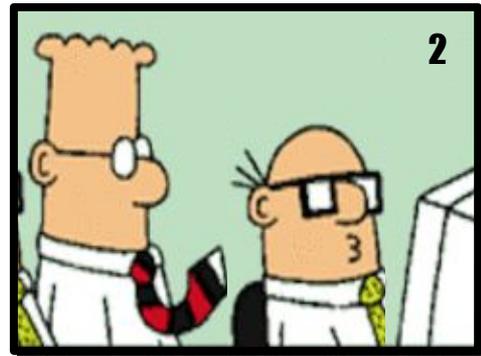
TDD Ping Pong

Ao invés de iniciar individualmente a praticar TDD, recomendamos essa técnica, que além de usarmos TDD, unimos com pair programming

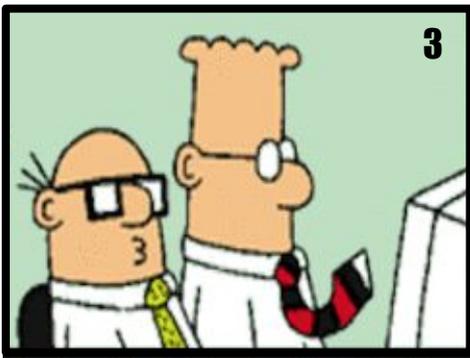
Programador A escreve teste que falha



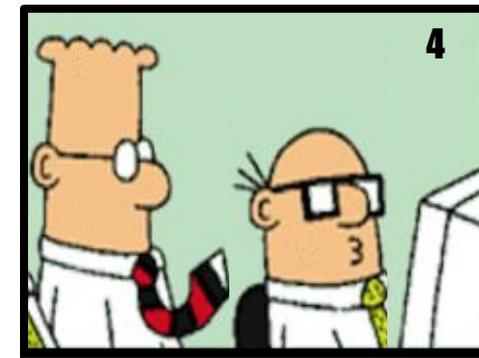
Programador B escreve código para passar teste



Programador A refatora o código



Programador B escreve teste que falha

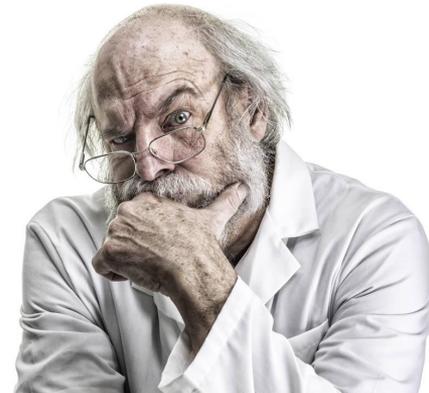


CONCLUSÕES



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



THANK
YOU!