TDC 2019

FUTURE PROOF IT

Fabrício Castro Vizzotto

Github: @FabricioVizzotto

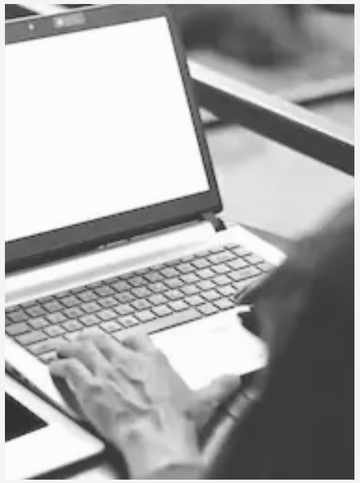
Luiza Henriques Pinheiro

Github: @luhpi

TELESSAÚDE RS

PROJETO UFRGS

PPG Epidemiologia



PLATAFORMA WEB

Abril de 2014

MUDANÇAS DE EQUIPE

Três no total

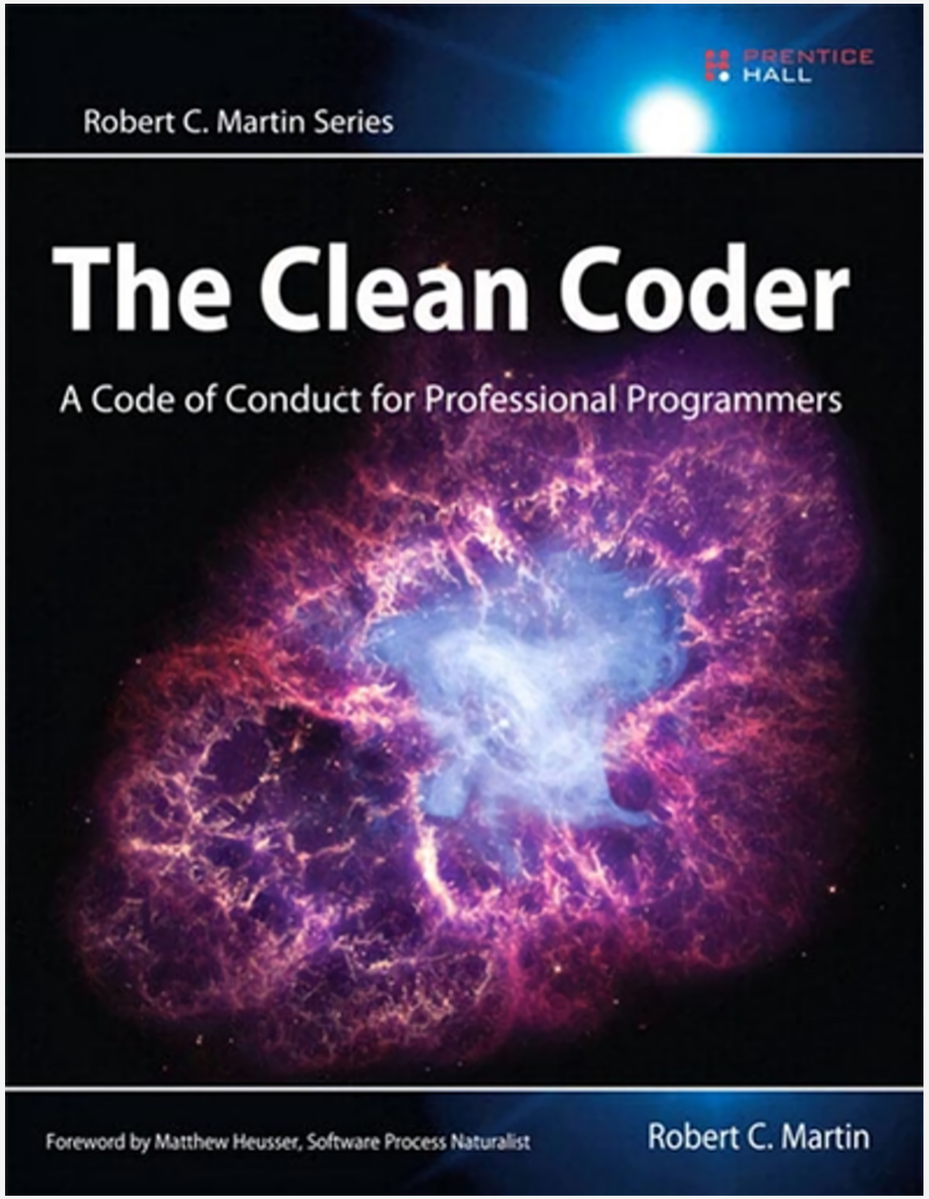
Como prever o futuro?

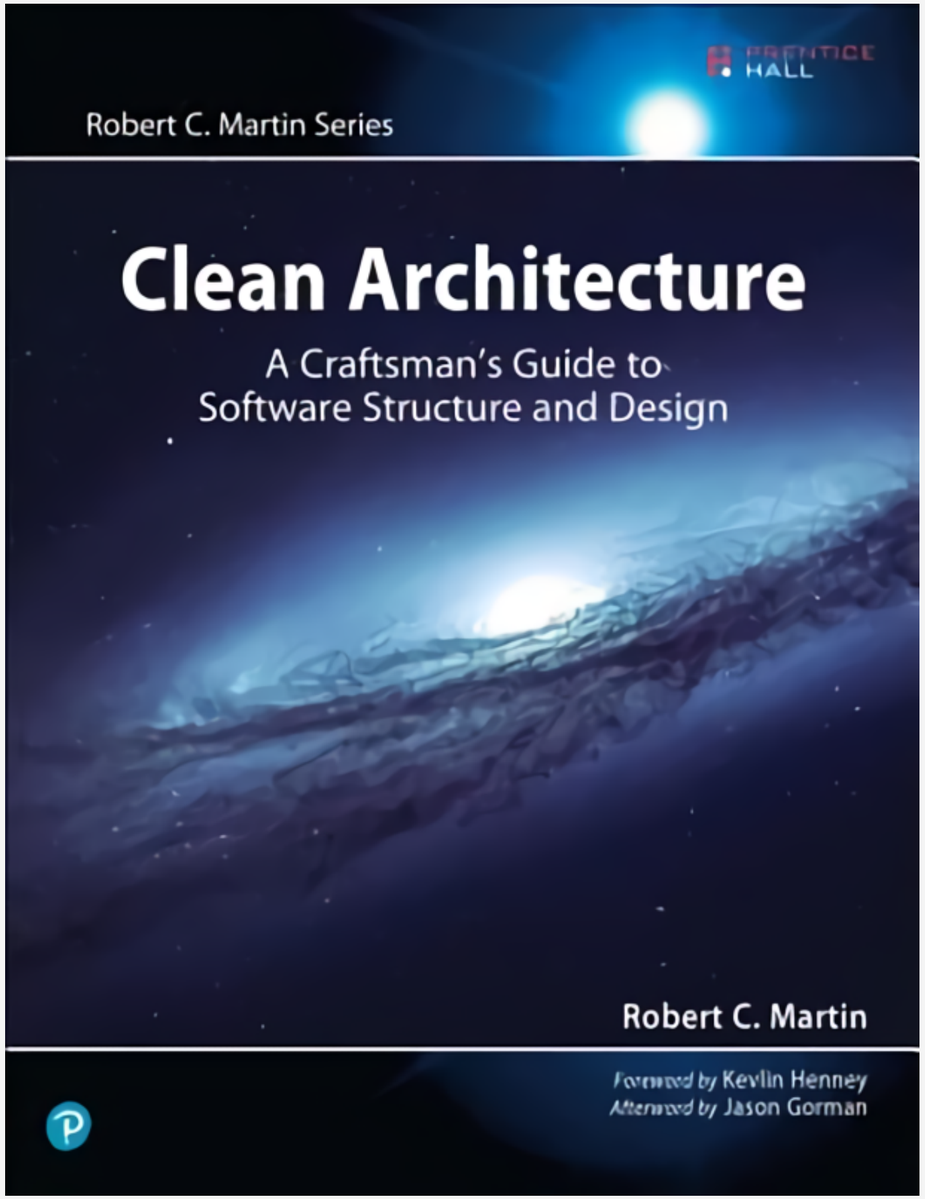
Robert Cecil Martin

MANIFESTO ÁGIL

SOLID

DIVERSAS PUBLICAÇÕES


PRENTICE HALL
Robert C. Martin Series
The Clean Coder
A Code of Conduct for Professional Programmers
Foreword by Matthew Heusser, Software Process Naturalist
Robert C. Martin


PRENTICE HALL
Robert C. Martin Series
Clean Architecture
A Craftsman's Guide to
Software Structure and Design
Robert C. Martin
Foreword by Kevlin Henney
Afterword by Jason Gorman

TIO BEN NOS
ENSINOU SOBRE
RESPONSABILIDADE,
TIO BOB NOS
ENSINOU SOBRE
CÓDIGO LIMPO

Arquitetura

e suas peculiaridades

BASE ESTRUTURAL

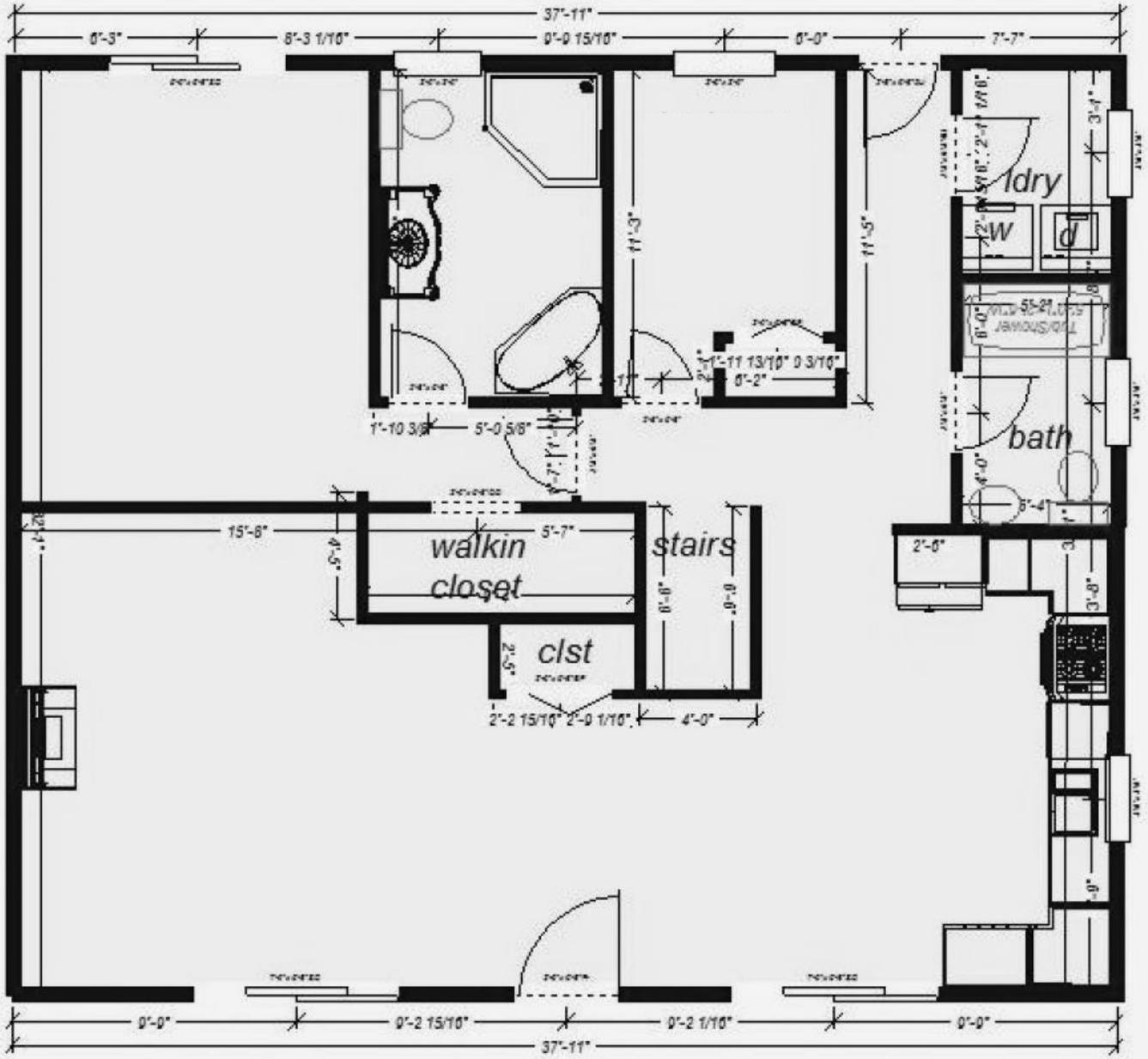
do projeto

LEITURA

por máquinas e por pessoas

RESPONSABILIDADE

é do desenvolvedor


37'-11"
0'-3"
8'-3 1/16"
9'-0 15/16"
6'-0"
7'-7"
11'-3"
11'-5"
ldry
w
d
bath
walkin
closet
stairs
clst
15'-6"
5'-7"
6'-0"
9'-0"
2'-6"
4'-0"
0'-0"
9'-2 15/16"
0'-2 1/16"
0'-0"
37'-11"

Boas práticas

DESENVOLVIMENTO E EQUIPE

1. SOLID

Single responsibility

UM
STAKEHOLDER
POR MUDANÇA

Open/closed principle

ABERTO PARA
EXPANSÃO
FECHADO PARA
MUDANÇAS

```python
class Animal():
    def __init__(self, name):
        self.name = name

    def poop(self):
        pass


class Dog(Animal):
    def bark(self):
        print("Woof!")


class Cat(Animal):
    def meow(self):
        print("Meow!")
```

```python
class Robot():
    def __init__(self, name):
        self.name = name

    def drive(self):
        print("Vruuum")


class MurderRobot(Robot):
    def kill(self, obj):
        del obj


class CleaningRobot(Robot):
    def clean(self, obj):
        for key in obj:
            obj[key] = None
```

```
class MurderRobotDog(MuderRobot):
    def bark(self):
        print("Woof!")


def main():
    dog = Dog('Spike')
    cat = Cat('Aslan')
    clean_robot = CleaningRobot('Bender')
    murder_robot_dog = MurderRobotDog('Benny')

    dog.bark()
    murder_robot_dog.drive()
    murder_robot_dog.kill(dog)
```

```
class Thing:
    def __init__(self, name):
        self.name = name
        self.position = 0
        self.speed = 100


class Driver:
    def __init__(self, name=None, obj=None):
        if obj:
            self.base = obj
        else:
            self.base = Thing(name)
    def drive(self):
        self.base.position += self.base.speed
        print("I am "+self.base.name+" my position is:"+ str(self.base.position))
    def __getattr__(self, attr):
        return getattr(self.base, attr)


class Murderer:
    def __init__(self, name=None, obj=None):
        if obj:
            self.base = obj
        else:
            self.base = Thing(name)
    def kill(self, thing):
        print("I'm gonna kill "+thing.name+", because I am "+self.base.name)
        del thing
    def __getattr__(self, attr):
        return getattr(self.base, attr)
```

```
class Cleaner:
    def __init__(self, name=None, obj=None):
        if obj:
            self.base = obj
        else:
            self.base = Thing(name)
    def clean(self, thing):
        print("I'm gonna clean "+thing.name+", because I am "+self.base.name)
        for key in thing.__dict__.keys():
            setattr(thing, key, None)
    def __getattr__(self, attr):
        return getattr(self.base, attr)


class Eater:
    def __init__(self, name=None, obj=None):
        if obj:
            self.base = obj
        else:
            self.base = Thing(name)
    def eat(self):
        pass
    def __getattr__(self, attr):
        return getattr(self.base, attr)
```

```
class Woofer:
    def __init__(self, name=None, obj=None):
        if obj:
            self.base = obj
        else:
            self.base = Thing(name)
    def bark(self):
        print("Woof! I am "+ self.base.name)
    def __getattr__(self, attr):
        return getattr(self.base, attr)


class Meower:
    def __init__(self, name=None, obj=None):
        if obj:
            self.base = obj
        else:
            self.base = Thing(name)
    def meow(self):
        print("Meow! I am "+ self.base.name)
    def __getattr__(self, attr):
        return getattr(self.base, attr)


def main():
    dog = Woofer(obj=Eater(name='Spike'))
    cat = Meower(obj=Eater(name='Aslan'))
    murder_cat = Murderer(obj=Meower(name='Avenger'))
    murder_robot_dog = Woofer(obj=Murderer(obj=Driver(name='Benny')))
    clean_robot = Cleaner(obj=Driver(name='Bender'))

    dog.bark()
    clean_robot.drive()
    clean_robot.drive()
    clean_robot.clean(cat)
    murder_robot_dog.drive()
    murder_robot_dog.kill(dog)
    murder_cat.meow()
```

Liskov substitution

CLASSES DERIVADAS
DEVEM SER
SUBSTITUÍVEIS
POR SUAS
CLASSES BASES

```
def main():
    dog = Woofer(obj=Eater(name='Spike'))
    print(dog.name)
    dog = Eater(name='Spike that doesnt bark')
    print(dog.name)
    dog = Thing('Spike, but no bark and no eating')
    print(dog.name)
```

Interface segragation

UM PROJETO DE
(N) MÓDULOS
DEVE TER
(N-1) INTERFACES

Dependency inversion

CLASSES
MAIS ESPECÍFICAS
DEPENDEM DE CLASSES
MAIS ABSTRATAS

2. Programação funcional

Programação funcional

FUNÇÕES PURAS

INTERFACES FUNCIONAIS

TESTES

3. Orientação a objetos sem herança

Orientação a objeto sem herança

Menor rigidez

Não gera "problema do diamante"

Maior legibilidade

Maior modularidade

Mudança tem que vir de fora também.

A EQUIPE INTEIRA TEM DEVERES SOBRE ISSO

Soluções a longo prazo

CODE REVIEW

Melhor olhar crítico

CODE REFACTORING

Hábito constante da revisão

TREINAMENTO

POCs como material de apoio

REQUISITOS

Documentar minimamente

Código Future Proofed

Não é a prova do futuro, é feito para se adaptar.

FABRICIO CASTRO VIZZOTTO

Github: @FabricioVizzotto
fabricio.vizzotto@telessauders.ufrgs.br

LUIZA HENRIQUES PINHEIRO

Github: @luhpi
luiza.pinheiro@telessauders.ufrgs.br

TELESSAÚDE RS

https://www.ufrgs.br/telessauders/
contato@telessauders.ufrgs.br

Obrigada