

# MANUTENÇÃO DE SOFTWARE

ENGENHARIA DE SISTEMAS DE INFORMAÇÃO

---

Daniel Cordeiro

24 de outubro de 2017

Escola de Artes, Ciências e Humanidades | EACH | USP

O QUE FAZ DE UM CÓDIGO SER  
“LEGADO” E COMO MÉTODO ÁGIL  
PODE AJUDAR?

---

Já que manutenção de software consume 60% dos custos com o software, essa é provavelmente a fase mais importante do ciclo de vida do software...

*“Old hardware becomes obsolete; old software goes into production every night.”*

Robert Glass, *Facts & Fallacies of Software Engineering* (fato #41)

**O que podemos fazer**

para entender e modificar (com segurança) um código legado?

- Melhorias: 60% do custo de manutenção
- Correção de bugs: 17% do custo de manutenção

### Daí a regra de “60/60”

- 60% do custo de software é manutenção
- 60% do custo de manutenção é melhoria

# O QUE FAZ DE UM CÓDIGO SER “LEGADO”?

Ele ainda faz o que o cliente precisa,  
mas além disso:

- você não o escreveu e ele está mal documentado
- ou você o escreveu, mas há muito, muito tempo atrás (e ele está mal documentado)
- ou ele não tem bons testes (independentemente de quem o escreveu) — Feathers, 2004



## 2 MODOS DE ENCARAR A MODIFICAÇÃO DE CÓDIGO LEGADO

### Edite & Reze

— “Tipo assim, eu meio que acho que eu provavelmente não quebrei nada.”



## 2 MODOS DE ENCARAR A MODIFICAÇÃO DE CÓDIGO LEGADO

### Edite & Reze

— “Tipo assim, eu meio que acho que eu provavelmente não quebrei nada.”



### Cubra & Modifique

— Faça com que a **cobertura de testes** seja seu cobertor de segurança!



## COMO MÉTODOS ÁGEIS PODEM AJUDAR?

1. **Exploração**: determine onde você precisa fazer as mudanças (pontos de mudanças)

## COMO MÉTODOS ÁGEIS PODEM AJUDAR?

1. **Exploração**: determine onde você precisa fazer as mudanças (pontos de mudanças)
2. **Refatoração**: o código em volta dos seus pontos de mudança são (a) testados? (b) testáveis?

## COMO MÉTODOS ÁGEIS PODEM AJUDAR?

1. **Exploração**: determine onde você precisa fazer as mudanças (pontos de mudanças)
2. **Refatoração**: o código em volta dos seus pontos de mudança são (a) testados? (b) testáveis?
  - se (a) é verdadeiro: “bora” mexer

## COMO MÉTODOS ÁGEIS PODEM AJUDAR?

1. **Exploração**: determine onde você precisa fazer as mudanças (pontos de mudanças)
2. **Refatoração**: o código em volta dos seus pontos de mudança são **(a)** testados? **(b)** testáveis?
  - se **(a)** é verdadeiro: “bora” mexer
  - **!(a) && (b)**: aplique ciclos de BDD+TDD para melhorar a cobertura do teste

## COMO MÉTODOS ÁGEIS PODEM AJUDAR?

1. **Exploração**: determine onde você precisa fazer as mudanças (pontos de mudanças)
2. **Refatoração**: o código em volta dos seus pontos de mudança são **(a)** testados? **(b)** testáveis?
  - se **(a)** é verdadeiro: “bora” mexer
  - **!(a) && (b)**: aplique ciclos de BDD+TDD para melhorar a cobertura do teste
  - **!(a) && !(b)**: refatore

## COMO MÉTODOS ÁGEIS PODEM AJUDAR?

1. **Exploração**: determine onde você precisa fazer as mudanças (pontos de mudanças)
2. **Refatoração**: o código em volta dos seus pontos de mudança são **(a)** testados? **(b)** testáveis?
  - se **(a)** é verdadeiro: “bora” mexer
  - **!(a) && (b)**: aplique ciclos de BDD+TDD para melhorar a cobertura do teste
  - **!(a) && !(b)**: refatore
3. Adicione testes para **melhorar a cobertura** conforme for preciso

## COMO MÉTODOS ÁGEIS PODEM AJUDAR?

1. **Exploração**: determine onde você precisa fazer as mudanças (pontos de mudanças)
2. **Refatoração**: o código em volta dos seus pontos de mudança são **(a)** testados? **(b)** testáveis?
  - se **(a)** é verdadeiro: “bora” mexer
  - **!(a) && (b)**: aplique ciclos de BDD+TDD para melhorar a cobertura do teste
  - **!(a) && !(b)**: refatore
3. Adicione testes para **melhorar a cobertura** conforme for preciso
4. **Faça mudanças** usando os testes como sua *referência base* (*ground truth*)

## COMO MÉTODOS ÁGEIS PODEM AJUDAR?

1. **Exploração**: determine onde você precisa fazer as mudanças (pontos de mudanças)
2. **Refatoração**: o código em volta dos seus pontos de mudança são (a) testados? (b) testáveis?
  - se (a) é verdadeiro: “bora” mexer
  - $!(a) \ \&\& \ (b)$ : aplique ciclos de BDD+TDD para melhorar a cobertura do teste
  - $!(a) \ \&\& \ !(b)$ : refatore
3. Adicione testes para **melhorar a cobertura** conforme for preciso
4. **Faça mudanças** usando os testes como sua *referência base* (*ground truth*)
5. **Refatore** ainda mais; deixe o código melhor do que você o encontrou

Isso sim é “abraçar as mudanças” em longo prazo

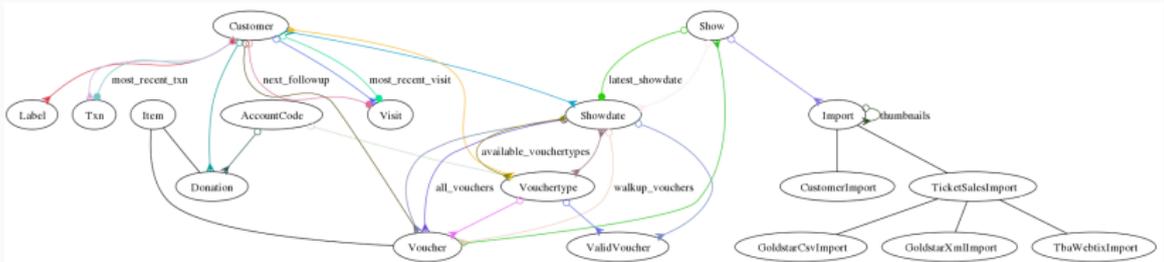
# ABORDAGEM E EXPLORAÇÃO DE CÓDIGO LEGADO

---

- Faça o *check out* de um *branch* de rascunho (que não será enviado novamente pro repositório) e faça ele rodar:
  - com uma configuração parecida com a produção ou com o ambiente de desenvolvimento
  - idealmente com algo que se pareça com uma cópia do banco de dados de produção
- Aprenda algumas histórias de usuário: converse com o cliente e peça para ele explicar o que ele faz com o software

# ENTENDA O ESQUEMA DO BANCO DE DADOS & AS CLASSES IMPORTANTES

- Inspecione o esquema do banco de dados (`rake db:schema:dump`)
- Crie um **diagrama de interação** automaticamente (`gem install railroady`) ou inspecione o código manualmente
- Quais são as *classes* principais (as mais conectadas), suas *responsabilidades* e seus *colaboradores*?



- Percepção geral do código
  - Qualidade de código subjetiva? (`rake metrics` depois de instalar a gema `metric-fu` ou usar o CodeClimate)
  - Razão código/teste? Tamanho do código? (`rake stats`)
  - Modelos/Visões/Controladores principais?
  - Testes Cucumber & RSpec
- Documentos informais do projeto
  - Esboços de interface lo-fi e histórias de usuário
  - E-mail arquivado, newsgroup, páginas do wiki interno, posts em blogs, etc. sobre o projeto
  - Anotações sobre a revisão do projeto (ex: Campfire ou Basecamp)
  - Logs do sistema de controle de versão, documentação RDoc

```
##  
# ClassModule is the base class for objects representing either a class or a  
# module.  
  
class RDoc::ClassModule < RDoc::Context  
  ##  
  # Constants that are aliases for this class or module  
  
  attr_accessor :constant_aliases  
  
  ##  
  # Comment and the location it came from. Use #add_comment to add comments  
  
  attr_accessor :comment_location  
  
  attr_accessor :diagram # :nodoc:  
  
  ##  
  # Class or module this constant is an alias for  
  
  attr_accessor :is_alias_for  
  
  ##  
  # Return a RDoc::ClassModule of class +class_type+ that is a copy  
  # of module +module+. Used to promote modules to classes.  
  #--  
  # TODO move to RDoc::NormalClass (I think)  
  
  def self.from_module class_type, mod  
    klass = class_type.new mod.name  
    ...  
  end  
end
```

Home

Pages Classes Methods

Search

Parent

RDoc::Context

Methods

```

::from_module
::new
#ack_comment
#ancestors
#aref
#clear_comment
#complete
#description
#direct_ancestors
#document_self_or_methods
#documented?
#each_ancestor
#find_ancestor_local_symbol
#find_class_named
#full_name
#merge
#module?
#name=
#name_for_path
#non_aliases
#parse
#path
#remove_rdoc_children
#search_record
#store=
#superclass
#superclass=
#type
#update_aliases
#update_extends
#update_includes

```

## class RDoc::ClassModule

ClassModule is the base class for objects representing either a class or a module.

### Attributes

#### comment\_location [RW]

Comment and the location it came from. Use `add_comment` to add comments

#### constant\_aliases [RW]

Constants that are aliases for this class or module

#### is\_alias\_for [RW]

Class or module this constant is an alias for

### Public Class Methods

#### from\_module(class\_type, mod)

Return a RDoc::ClassModule of class `class_type` that is a copy of module `module`. Used to promote modules to classes.

#### new(name, superclass = nil)

Creates a new ClassModule with `name` with optional `superclass`

This is a constructor for subclasses, and must never be called directly.

Calls superclass method `RDoc::Context.new`

### Public Instance Methods

#### add\_comment(comment, location)

Adds `comment` to this ClassModule's list of comments at `location`. This method is preferred over `comment=` since it allows ri data to be updated across multiple runs.

#### ancestors()

Ancestors list for this ClassModule: the list of included modules (classes will add their superclass if any).

Returns the included classes or modules, not the include themselves. The returned values are either String or RDoc::NormalModule instances (see RDoc::Mixin#module)

- Avalie a base de código
- Identifique as classes e relações principais
- Identifique as estruturas de dados mais importantes
- Idealmente, identifique o(s) lugar(es) onde será necessário mudar o código
- Mantenha a documentação do projeto atualizada ao mudar o código:
  - diagramas
  - wiki do GitHub
  - comentários que você inserir usando o RDoc

# ESTABELECENDO A REFERÊNCIA BASE COM TESTES DE CARACTERIZAÇÃO

---

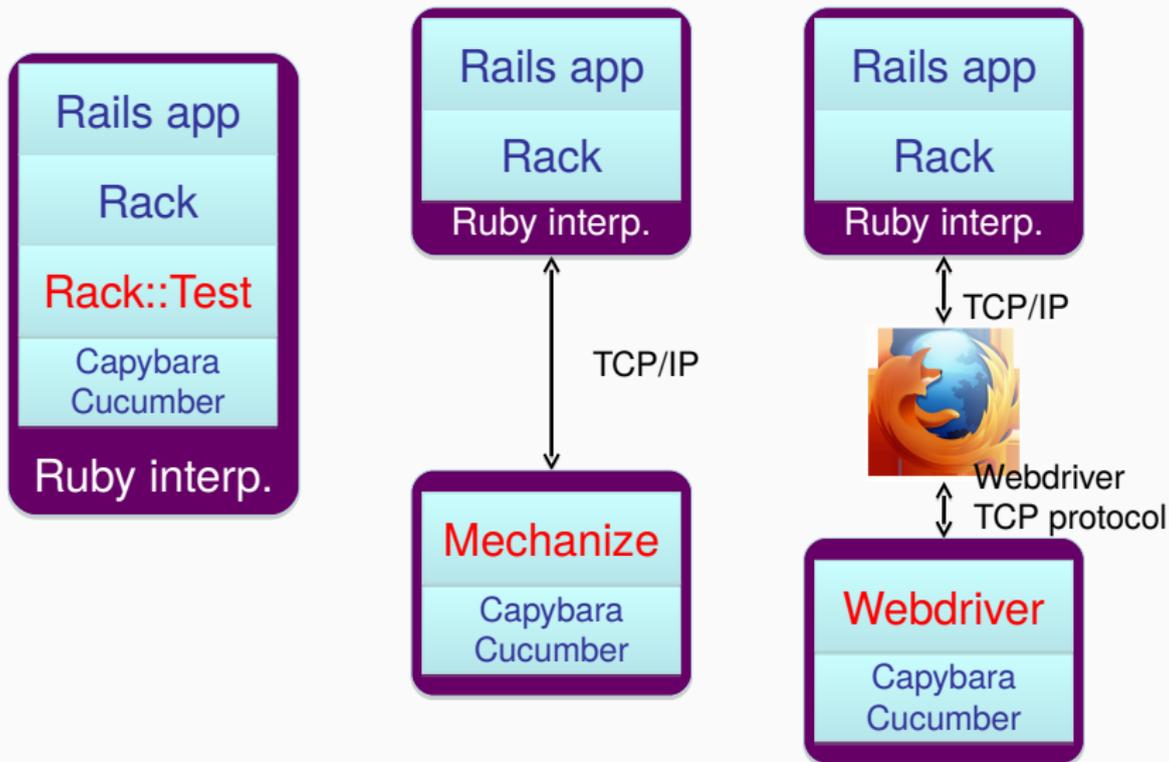
- Você não quer escrever código sem testes
- Você não tem os testes
- Você não pode criar testes sem entender o código

**Por onde começar?**

- Estabeleça a *referência base sobre como o código funciona hoje*, como a base para a cobertura
  - Faça com que comportamentos conhecidos fiquem **Repetíveis**
  - Aumente a confiança de que você não está quebrando nada
- **Armadilha: não tente fazer melhorias nesse estágio!**

- Primeiro passo natural: caixa-preta / nível de integração
  - não depende de entender a estrutura interna do app
- Use e abuse do Cucumber
  - *back-ends* do Capybara como o Mechanize faz com possamos automatizar quase tudo com um script
  - escreva os cenários imperativos agora
  - converta-os para declarativo ou melhore os passos **Given** depois, quando tiver um entendimento melhor de como funciona o app por dentro

# IN-PROCESS VS. OUT-OF-PROCESS



Escreva testes para ir aprendendo sobre o código  
Veja o screencast em <https://youtu.be/8QwvqtMp5QM>

```
class TimeSetter
  def self.convert(d)
    y = 1980
    while (d > 365) do
      if (y % 400 == 0 ||
          (y % 4 == 0 && y % 100 != 0))
        if (d > 366)
          d -= 366
          y += 1
        end
      else
        d -= 365
        y += 1
      end
    end
    return y
  end
end
```

# TESTES DE CARACTERIZAÇÃO NO NÍVEL DE UNIDADE E FUNCIONAL

```
it "should calculate sales tax" do
  order = mock('order')
  expect(order.compute_tax).to eq -99.99
end
# object 'order' received unexpected message 'get_total'
```

```
it "should calculate sales tax" do
  order = mock('order', :get_total => 100.00)
  expect(order.compute_tax).to eq -99.99
end
# expected compute_tax to be -99.99, was 8.45
```

```
it "should calculate sales tax" do
  order = mock('order', :get_total => 100.00)
  expect(order.compute_tax).to eq 8.45
end
```

# TESTES DE CARACTERIZAÇÃO NO NÍVEL DE UNIDADE E FUNCIONAL

```
it "should calculate sales tax" do
  order = mock('order')
  expect(order.compute_tax).to eq -99.99
end
# object 'order' received unexpected message 'get_total'
```

```
it "should calculate sales tax" do
  order = mock('order', :get_total => 100.00)
  expect(order.compute_tax).to eq -99.99
end
# expected compute_tax to be -99.99, was 8.45
```

```
it "should calculate sales tax" do
  order = mock('order', :get_total => 100.00)
  expect(order.compute_tax).to eq 8.45
end
```

## TESTES DE CARACTERIZAÇÃO NO NÍVEL DE UNIDADE E FUNCIONAL

```
it "should calculate sales tax" do
  order = mock('order')
  expect(order.compute_tax).to eq -99.99
end
# object 'order' received unexpected message 'get_total'

it "should calculate sales tax" do
  order = mock('order', :get_total => 100.00)
  expect(order.compute_tax).to eq -99.99
end
# expected compute_tax to be -99.99, was 8.45

it "should calculate sales tax" do
  order = mock('order', :get_total => 100.00)
  expect(order.compute_tax).to eq 8.45
end
```