Beleza de Código

Nelson Lago / Fabio Kon

MAC-350 – Desenvolvimento de Sistemas de Software

Agosto/2017

Licensed under the Creative Commons Attribution 4.0 International License (CC-BY 4.0).

Visão geral

- **Beleza**
  - Por quê?
  - Histórico
- **O que é beleza no desenvolvimento de software?**
  - Beleza e qualidade
  - Alguns exemplos e opiniões
- **Aspectos específicos: nomes**
  - Consistência e expressividade
- **Aspectos específicos: funções**
  - Boas abstrações
  - Simplicidade
- **Aspectos específicos: comentários**
  - Bons e maus comentários

Visão geral

- **Beleza**
  - Por quê?
  - Histórico
- **O que é beleza no desenvolvimento de software?**
  - Beleza e qualidade
  - Alguns exemplos e opiniões
- **Aspectos específicos: nomes**
  - Consistência e expressividade
- **Aspectos específicos: funções**
  - Boas abstrações
  - Simplicidade
- **Aspectos específicos: comentários**
  - Bons e maus comentários

Por que falar de beleza?

- **É comum abordar a gerência de projetos de software como outros processos ou sistemas de produção**
  - Mas desenvolvimento de software tem um forte caráter de “artesanato”
- **Há um grande número de gerentes de software que jamais escreveram código**
  - Será possível que exista um bom *Chef de Cuisine* que nunca cozinhou na vida?

Por que falar de beleza?

- **Cientistas, engenheiros etc. muitas vezes tendem a superestimar o poder da ciência, da matemática e da engenharia**
  - Mas será que a ciência, a matemática e a engenharia sozinhas oferecem tudo o que é necessário para explicar e vivenciar o universo?
  - E, em particular, o desenvolvimento de software?

O que é beleza?

- **Segundo o dicionário Houaiss da língua portuguesa:**
  - Caráter do ser ou coisa que desperta sentimento de êxtase, admiração ou prazer através dos sentidos
  - Característica daquilo que possui harmonia, proporção, simetria, imponência etc.
  - Qualidade do ser ou coisa que suscita a admiração e um sentimento de adesão por seu valor moral ou intelectual
- **Parece um bom objetivo para qualquer coisa que se faça na vida!**
- **Trabalhar com isso parece ser bom!**

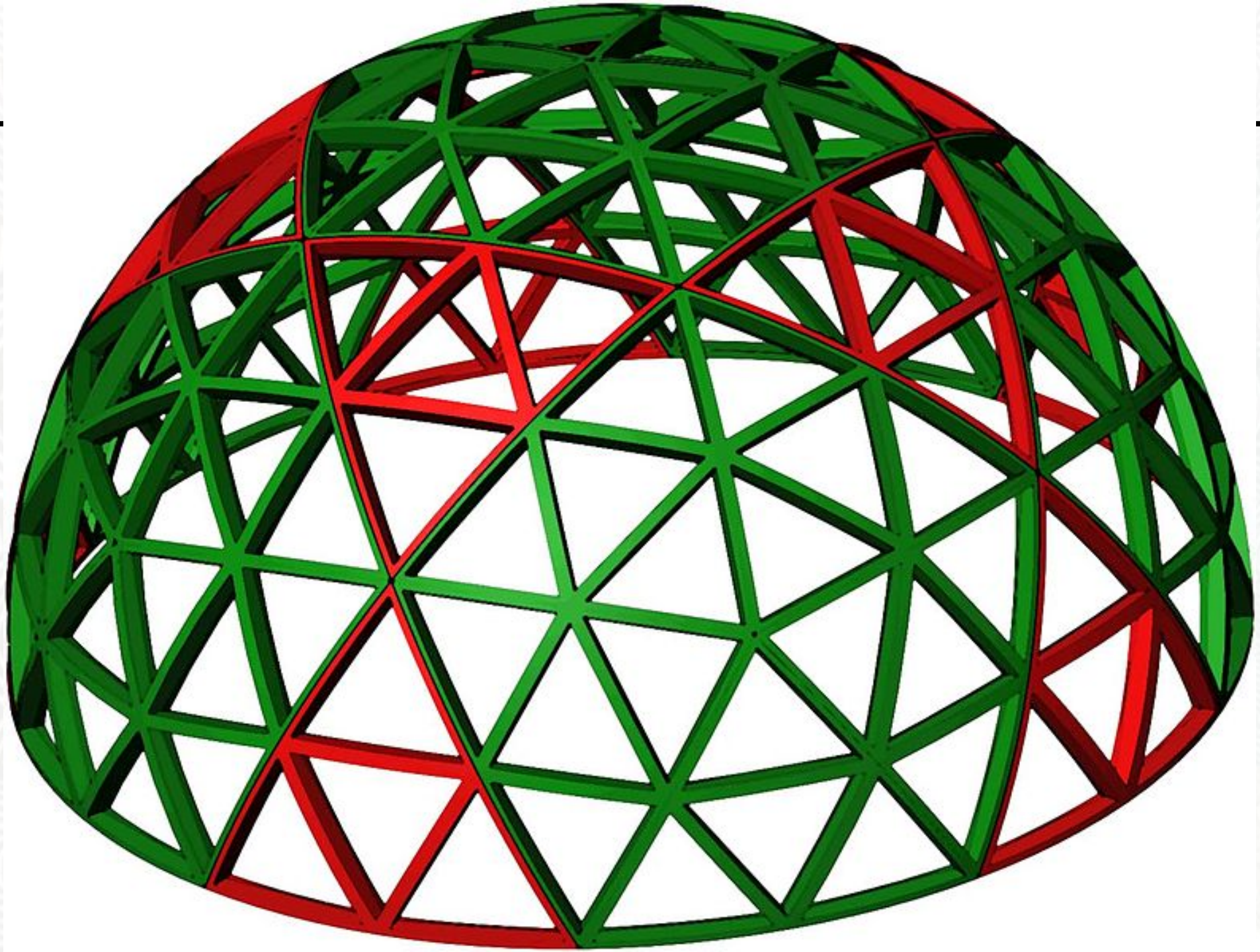
O que é beleza? – história

- **Várias formas na cultura ocidental:**

- **Pré-socráticos (pitágoras): beleza e matemática, proporção áurea, simetria**
  - Objetos cujas proporções seguem a proporção áurea são mais atraentes para o cérebro humano
    - Proporção áurea: a relação entre a soma das dimensões e a dimensão maior é igual à relação entre a dimensão maior e a dimensão menor, cerca de 1.618
  - Arquitetura grega é baseada na simetria e na proporção áurea

O que é beleza? – história

- **Várias formas na cultura ocidental:**
- **Romantismo: fragmentação e irregularidade como belo**
  - Talvez pela alusão à harmonia ausente
- **Modernismo: transparência e simplicidade**
  - “Less is more” (Mies van der Rohe, ~ 1940)
  - “a perfeição é atingida não quando não há mais o que acrescentar, mas quando não há mais o que remover” (Antoine de Saint-Exupéry, ~ 1939)
  - Cúpulas geodésicas de Buckminster Fuller (~ 1948)



O que é beleza? – história

- **Várias formas na cultura ocidental:**
- **Pós-modernismo: “tudo ao mesmo tempo agora”**
  - “Less is a bore” (Robert Venturi)

Visão geral

- **Beleza**
  - Por quê?
  - Histórico
- **O que é beleza no desenvolvimento de software?**
  - Beleza e qualidade
  - Alguns exemplos e opiniões
- **Aspectos específicos: nomes**
  - Consistência e expressividade
- **Aspectos específicos: funções**
  - Boas abstrações
  - Simplicidade
- **Aspectos específicos: comentários**
  - Bons e maus comentários

Beleza e tecnologia

- **As visões grega e moderna parecem uma boa base para pensar em beleza no contexto da ciência e tecnologia**
  - Ciência: “Faça as coisas da maneira mais simples possível, mas não mais simples que isso” (atribuído a Einstein)
  - Tecnologia: “integridade conceitual: o arquiteto deve desenvolver uma ideia do que o sistema deve fazer e garantir que essa visão é compreendida pelo restante da equipe. Para garantir um sistema fácil de usar, pode-se deliberadamente oferecer menos funcionalidades do que seria possível” (Fred Brooks)

Beleza e tecnologia

- **O mais importante produto de um projeto de desenvolvimento é o código**
- **Beleza é fundamental!**
- **Código bonito:**
  - Dá prazer ao leitor
  - Faz o escritor feliz
  - Torna o trabalho em equipe mais agradável
- **O que traz:**
  - Menos bugs
  - Melhor manutenibilidade
  - Maior produtividade da equipe
  - Ou seja, **qualidade**

Portanto, no desenvolvimento de software,

Beleza traz qualidade

O que é desenvolvimento de software?

- **Modelagem (Jacobsen)**
- **Engenharia (Meyer)**
- **Disciplina (Humphreys)**
- **Poesia (Cockburn)**
- **Arte aplicada (*Craft*) (Knuth)**
- **Arte (Gabriel)**

(Lista de Alistair Cockburn)

De fato, tudo isso!

O que é beleza no desenvolvimento de software?

- **Rebecca Wirfs-Brock:**
  - É consenso que as estruturas, construções e a maneira de fazer as coisas são adequadas
  - É importante que um projeto ou equipe tenha um senso estético comum, ou haverá conflitos
  - Manutenção de todos os passos em um método no mesmo nível de abstração e intenção; dessa maneira, o código parece texto corrido
  - Características comumente presentes em código belo são equilíbrio, eficiência, expressividade e a execução extremamente precisa do que se espera dele

O que é beleza no desenvolvimento de software?

- **Um exemplo bastante popular: a classe "Collection" do SmallTalk (Rebecca Wirfs-Brock):**
  - Garante todas as funcionalidades básicas exigindo que as subclasses implementem três métodos:
    - add: anObject
    - remove: anObject ifAbsent: exceptionBlock
    - do: aBlock
  - Os métodos da classe abstrata "Collection" para adicionar e remover elementos, acumular diferentes resultados ao processar todos os elementos, checar se a classe está vazia etc. são todos implementados utilizando esses aspectos básicos ou através de outros procedimentos pré-definidos que dependem apenas dessas implementações; reúso belo e elegante!

What is Clean Code?

Bjarne Stroustrup

Inventor of C++

"I like my code to be **elegant** and **efficient**. The logic should be **straighforward** to make it hard for bugs to hide, the **dependencies minimal to ease maintenance**, **error handling complete** according to an articulated strategy, and **performance close to optimal** so as not to tempt people to make the code messy with unprincipled optimizations. **Clean code does one thing well.**"

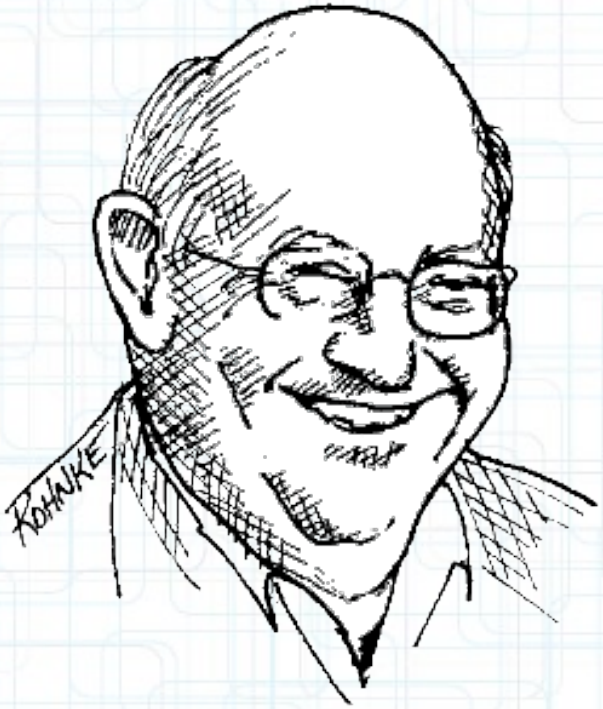
What is Clean Code?

"Clean code is **simple** and **direct**. Clean code **reads like well-written prose**. Clean code never obscures the designer's intent but rather is full of crisp [clearly defined] abstractions and **straighforward** lines of control."

Grady Booch

Author of *Object Oriented Analysis and Design with Applications*

What is Clean Code?



Dave Thomas

Founder of OTI, godfather of the Eclipse Strategy

"**Clean code can be read**, and enhanced by a developer other than its original author. **It has unit and acceptance tests**. It has **meaningful names**. It provides **one way** rather than many ways for doing one thing. It has **minimal dependencies**, which are explicitly defined, and provides a clear and **minimal API**. Code should be **literate** since depending on the language, not all necessary information can be expressed clearly in code alone."

What is Clean Code?

Michael Feathers

Author of *Working Effectively With Legacy Code*

"I could list all of the qualities that I notice in clean code, but there is one overarching quality that leads to all of them. **Clean code always looks it was written by someone who cares. There is nothing obvious that you can do to make it better.** All of those things were thought about by the code's author, and if you try to imagine improvements, you're led back to where you are, sitting in appreciation of the code someone left for you – **code left by someone who cares deeply about the craft.**"

What is Clean Code?

Ron Jeffries

Author of *Extreme Programming Installed*

"In recent years I begin, and nearly end, with Beck's rules of simple code. In priority order, simple code:

- **Runs all tests**
- **Contains no duplication**
- **Expresses all the design ideas** that are in the system
- **Minimizes the number of entities** such as classes, methods, functions, and the like."

What is Clean Code?

Ward Cunningham

Inventor of Wiki, Fit and much more
”Godfather of all those who care about code”

You know you are working on clean code when **each routine you read turns out to be pretty much what you expected**. You can call it **beautiful** code when the codes also **makes it look like the language was made for the problem**.”

What is Clean Code?

Simple

Efficient

Without obvious improvements

Straightforward

Expressive

Contains no duplications

Turns out to be what you expected

Runs all tests

Full of meaning

Literal

Reads well

Beautiful: when the language was made for the problem

Minimal

Written by someone who cares

Visão geral

- **Beleza**
  - Por quê?
  - Histórico
- **O que é beleza no desenvolvimento de software?**
  - Beleza e qualidade
  - Alguns exemplos e opiniões
- **Aspectos específicos: nomes**
  - Consistência e expressividade
- **Aspectos específicos: funções**
  - Boas abstrações
  - Simplicidade
- **Aspectos específicos: comentários**
  - Bons e maus comentários

Aspectos específicos: nomes

- **Nomes expressivos são fundamentais!**
  - Um programa é basicamente composto de palavras reservadas e nomes
  - Escolher bons nomes toma tempo; no entanto, isso acaba economizando mais tempo ainda
  - Os nomes devem ser expressivos e eliminar dúvidas
  - Exemplo:

```
public List<int[]> getThem() {
   List<int[]> list1 = new ArrayList<int[]>();
   for (int[] x : theList)
      if(x[0] == 4)
         list1.add(x);
   return list1;
}
```

Aspectos específicos: nomes

```
public List<int[]> getThem() {
   List<int[]> list1 = new ArrayList<int[]>();
   for (int[] x : theList)
      if(x[0] == 4)
         list1.add(x);
   return list1;
}
```

- **Dúvidas!!!**
  - O que esse método seleciona?
  - Quais tipos de coisa estão em theList?
  - Qual a relevância da posição zero?
  - O que significa “4”?

Aspectos específicos: nomes

```
public List<int[]> getThem() {
   List<int[]> list1 = new ArrayList<int[]>();
   for (int[] x : theList)
      if(x[0] == 4)
         list1.add(x);
   return list1;
}
```

- **Que tal assim:**

```
public List<int[]> getFlaggedCells() {
   List<int[]> flaggedCells = new ArrayList<int[]>();
   for (int[] cell : gameBoard)
      if(cell[STATUS_VALUE] == FLAGGED)
         flaggedCells.add(cell);
   return flaggedCells;
}
```

Aspectos específicos: nomes

```
public List<int[]> getFlaggedCells() {
   List<int[]> flaggedCells = new ArrayList<int[]>();
   for (int[] cell : gameBoard)
      if(cell[STATUS_VALUE] == FLAGGED)
         flaggedCells.add(cell);
   return flaggedCells;
}
```

- **Ahá!**
  - O que esse método seleciona? **As células marcadas!**
  - Quais tipos de coisa estão em theList? **theList é um tabuleiro com células!**
  - Qual a relevância da posição zero? **É o status!**
  - O que significa “4”? **Significa “marcado”!**

Aspectos específicos: nomes

```
public List<int[]> getFlaggedCells() {
   List<int[]> flaggedCells = new ArrayList<int[]>();
   for (int[] cell : gameBoard)
      if(cell[STATUS_VALUE] == FLAGGED)
         flaggedCells.add(cell);
   return flaggedCells;
}
```

- **Mas ainda dá pra melhorar!**

```
public List<Cell> getFlaggedCells() {
   List<Cell> flaggedCells = new ArrayList<Cell>();
   for (Cell cell : gameBoard)
      if(cell.isFlagged())
         flaggedCells.add(cell);
   return flaggedCells;
}
```

Aspectos específicos: nomes

- **Seja claro e consistente**
  - Use nomes legíveis
    - XYZControllerHandlingOfStrings != XYZControllerStorageOfStrings
  - É melhor que os nomes sejam relativamente curtos, desde que sejam claros
    - Economizar alguns toques no teclado não é uma boa razão para promover a obscuridade
  - Use nomes que possam ser usados em buscas
  - Use os padrões da linguagem (camelCase etc.)

Aspectos específicos: nomes

- **Comunique bem o que você quer dizer**
  - Use nomes do domínio da solução
    - Padrões de projeto, algoritmos, termos matemáticos etc.
  - Use nomes do domínio do problema
    - Atores envolvidos, tipos de produtos etc.
  - Não confunda o leitor
    - Uma palavra para cada conceito (get, fetch, retrieve)
    - Evite piadinhas ou metáforas de significado obscuro

Aspectos específicos: nomes

Falar é fácil, mas isso é importante MESMO

Se você encontrar um mau nome,
mude-o imediatamente!

Visão geral

- **Beleza**
  - Por quê?
  - Histórico
- **O que é beleza no desenvolvimento de software?**
  - Beleza e qualidade
  - Alguns exemplos e opiniões
- **Aspectos específicos: nomes**
  - Consistência e expressividade
- **Aspectos específicos: funções**
  - Boas abstrações
  - Simplicidade
- **Aspectos específicos: comentários**
  - Bons e maus comentários

Aspectos específicos: funções

- **O que a função faz deve ser óbvio**
  - Permite entender o problema
- **A implementação da função deve ser simples**
  - Permite entender a solução

Aspectos específicos: funções

O que a função faz deve ser óbvio

- **Uma função ↔ uma abstração**
  - Uma função faz uma e apenas uma coisa (corretamente!)
- **Evitar efeitos colaterais**
  - **OU** modifica um objeto **OU** devolve alguma coisa

Aspectos específicos: funções

- **DRY: Don’t Repeat Yourself**
  - Se um trecho de código aparece em mais de um lugar, há uma abstração implícita
    - E, portanto, as funções que usam esse trecho fazem mais de uma coisa!
- **Uma função que faz apenas uma coisa não pode ser dividida em seções**
  - Se não é possível extrair uma sub-função de dentro dela cujo nome não seria o mesmo que o nome que ela já tem, ela provavelmente faz só uma coisa

Aspectos específicos: funções

- **Exemplo:**

```
public void pay() {
    for (Employee e : employees) {
        if (e.isPayday()) {
            Money pay = e.calculatePay();
            e.deliverPay(pay);
        }
    }
}
```

- **Essa função faz mais que uma coisa!**
  - Itera por todos os empregados
  - Checa para ver quais empregados precisam ser pagos
  - Paga os empregados

Aspectos específicos: funções

```
public void pay() {
    for (Employee e : employees)
        payIfNecessary(e);
}
```

Apenas itera

```
private void payIfNecessary(Employee e) {
    if (e.isPayday())
        calculateAndDeliverPay();
}
```

Verifica se o funcionário precisa ser pago

```
private void
calculateAndDeliverPay(Employee e) {
    Money pay = e.calculatePay();
    e.deliverPay(pay);
}
```

paga o funcionário

- **Será um exagero?**
  - Cada função é mais simples, mas zilhões de funções também são um aumento na complexidade

Aspectos específicos: funções

- **Uma função ↔ uma abstração**
  - "The primary cost of abstraction is indirection. In my experience, you should only abstract when the added abstraction clarifies things more than the added indirection confuses them." – Bryan Edds
    - » *http://wiki.c2.com/?TooMuchAbstraction*
  - Uma nova função pode ser a abstração correta hoje, mas a errada amanhã
    - "Prefer duplication over the wrong abstraction"
      - » *https://www.sandimetz.com/blog/2016/1/20/the-wrong-abstraction*

Só a refatoração salva! ;)

Aspectos específicos: funções

- **O que a função faz deve ser óbvio**
  - Permite entender o problema
- **A implementação da função deve ser simples**
  - Permite entender a solução

Aspectos específicos: funções

A implementação da função deve ser simples

- **Funções “pequenas”**
- **Um único nível de abstração**
  - Quando detalhes se misturam a conceitos mais abstratos, mais e mais detalhes tendem a se insinuar
  - É o primeiro passo rumo às funções gigantes
- **Evitar estruturas aninhadas**
  - Blocos de if's/while's/else's devem ser diretos no que fazem (provavelmente, apenas chamar uma função)
    - Condicionais provavelmente devem ir para uma função separada

Aspectos específicos: funções

- **Exemplo:**

```
public void gameLoop() {
    advanceTimer();
    moveMonsters();
    movePlayers();
    removeDeadBodies();
    if (keyPressed) {
        if key.scancode == “X”; then...
    }
}
```

Hein?!?!

- **Mas aqui também vale evitar os exageros!**

Aspectos específicos: funções

- **Toda abstração “vaza”**
  - » *https://www.joelonsoftware.com/2002/11/11/the-law-of-leaky-abstractions/*
  - Muitas vezes é preciso entender o que está “por baixo” (implementação, suas limitações e *tradeoffs*)
  - É preciso cuidar dos aspectos que “vazam”
  - Definir uma boa abstração que “vaze menos” envolve entender o que está “por cima” (diferentes usos possíveis da abstração)

Aspectos específicos: funções

- **O número de argumentos de uma função deve ser pequeno**
  - Argumentos dificultam os testes
  - um grande número de argumentos sugere que a função faz mais que uma coisa
  - um grande número de argumentos sugere que a função é usada de maneiras muito díspares
  - Argumentos que na verdade são “flags” praticamente garantem que a função faz mais que uma coisa
    - rotaciona (int angulo, bool sentidoHorario)
    - rotacionaHorario (int angulo), rotacionaAntiHorario (int angulo)

Visão geral

- **Beleza**
  - Por quê?
  - Histórico
- **O que é beleza no desenvolvimento de software?**
  - Beleza e qualidade
  - Alguns exemplos e opiniões
- **Aspectos específicos: nomes**
  - Consistência e expressividade
- **Aspectos específicos: funções**
  - Boas abstrações
  - Simplicidade
- **Aspectos específicos: comentários**
  - Bons e maus comentários

Aspectos específicos: comentários

- **Comentários são, no máximo, um mal necessário**
  - O quase único uso dos comentários é compensar nossa *incapacidade* de expressão através do código
- **Comentários são mentirosos**
  - Comentários nem sempre acompanham as mudanças no código
  - Mantê-los requer tempo; Na prática, mantê-los acaba sendo impossível
  - A verdade está em um único lugar: no *código*

A verdade está em um único lugar: no *código*

Aspectos específicos: comentários

- **Bons comentários:**
  - Comentários sobre aspectos legais que influenciam o código
  - Alguns tipos de comentários informativos, como explicações sobre expressões regulares
  - Esclarecimentos sobre decisões tomadas em função não da implementação, mas do problema
  - Esclarecimentos sobre bibliotecas de terceiros que não são tão claras
  - Comentários que frisam a importância de um determinado elemento
  - Comentários tipo TODO e JavaDoc para APIs públicas

Visão geral

- **Beleza**
  - Por quê?
  - Histórico
- **O que é beleza no desenvolvimento de software?**
  - Beleza e qualidade
  - Alguns exemplos e opiniões
- **Aspectos específicos: nomes**
  - Consistência e expressividade
- **Aspectos específicos: funções**
  - Boas abstrações
  - Simplicidade
- **Aspectos específicos: comentários**
  - Bons e maus comentários

Bibliografia

- **Fontes relevantes sobre o tema:**
  - Robert C. Martin. *Clean Code - A Handbook of Agile Software Craftsmanship*. Prentice Hall. 2008.
  - Andy Oram and Greg Wilson. *Beautiful Code*. O'Reilly. 2007.
  - “Bad Smells in Code” – capítulo 3 do livro “Refactoring”, de Martin Fowler
  - beaut.e(code) - exposição de arte por Bob Hanmer, Karen Hanmer e Andrea Polli. Algum material pode ser acessado em http://karenhanmer.com/gallery/?gallery=beautecode
  - Conversas e entrevistas
  - Estes slides foram baseados no trabalho de João Machini de Miranda - IME/USP