PROVAS DE TI

TUDO PARA VOCÊ PASSAR

Clean Code

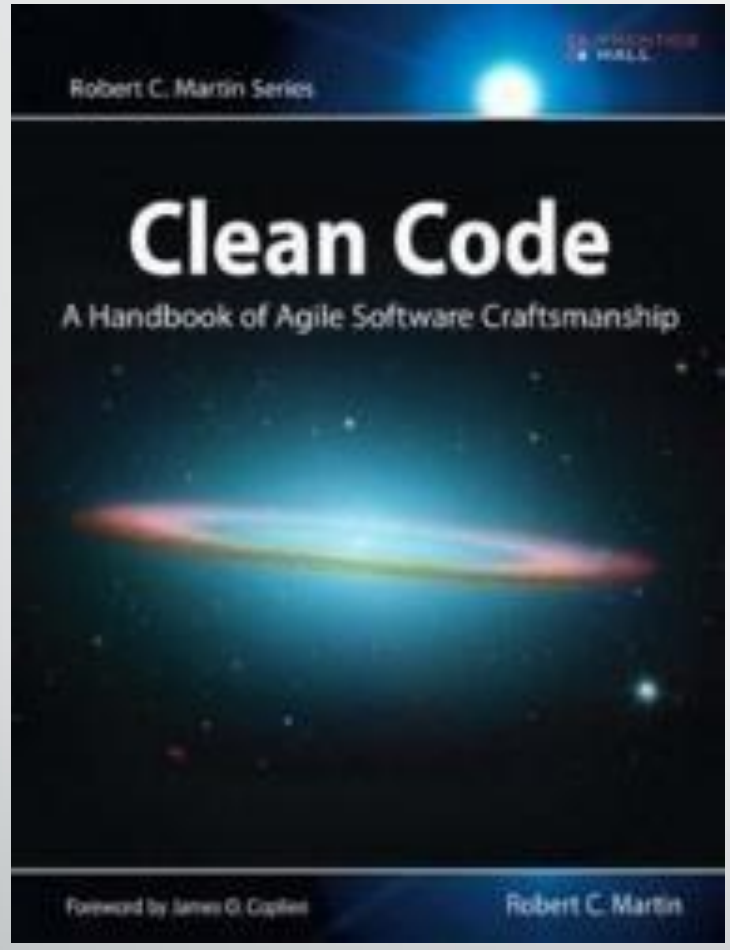
http://www.itnerante.com.br/profile/RodrigoMacedo

gabaritou <TI>

http://www.provasdeti.com.br/mdl-httprex-http-e-rest-em-exercicios-prof-rodrigo-macedo.html

http://www.provasdeti.com.br/iso9126-para-concurso-de-ti.html

Bibliografia

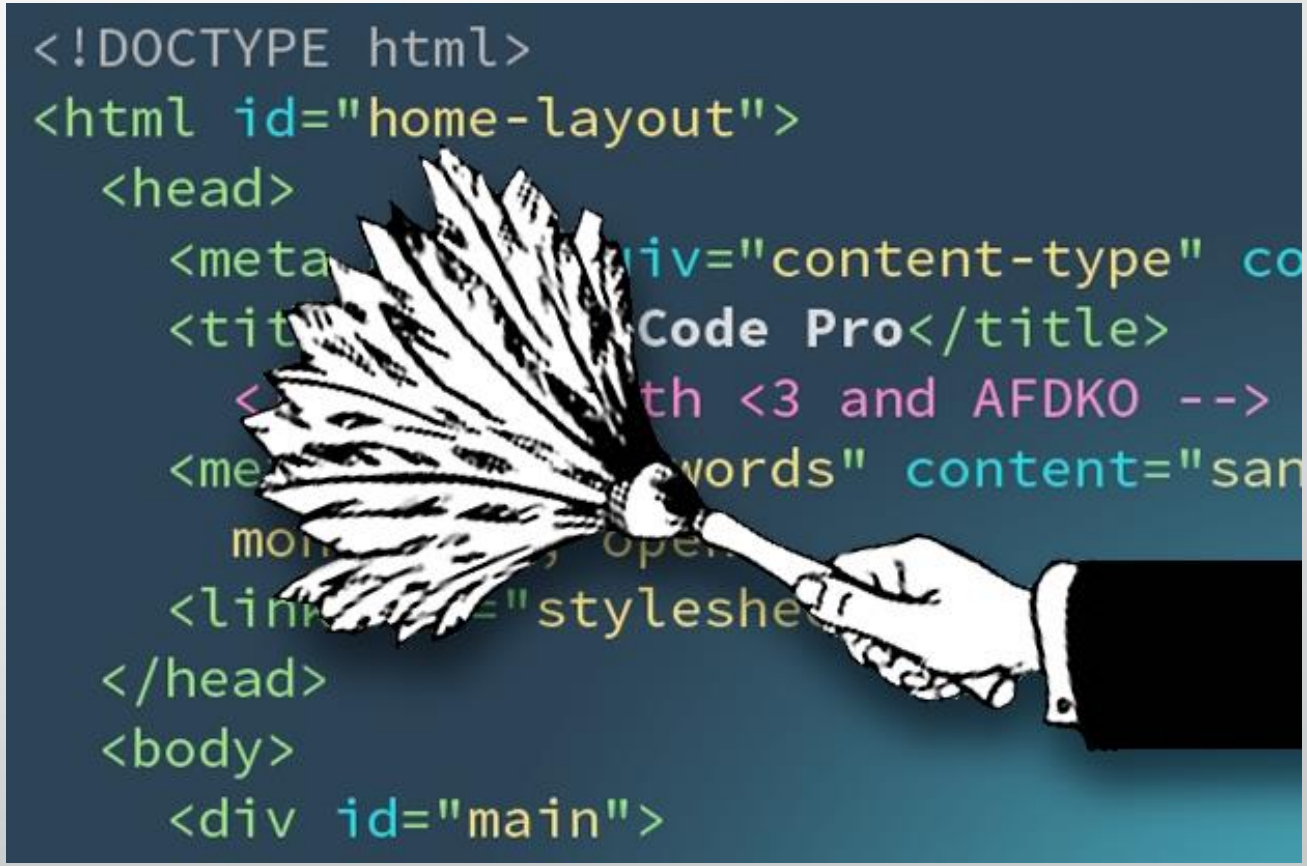


Relevância do Tema

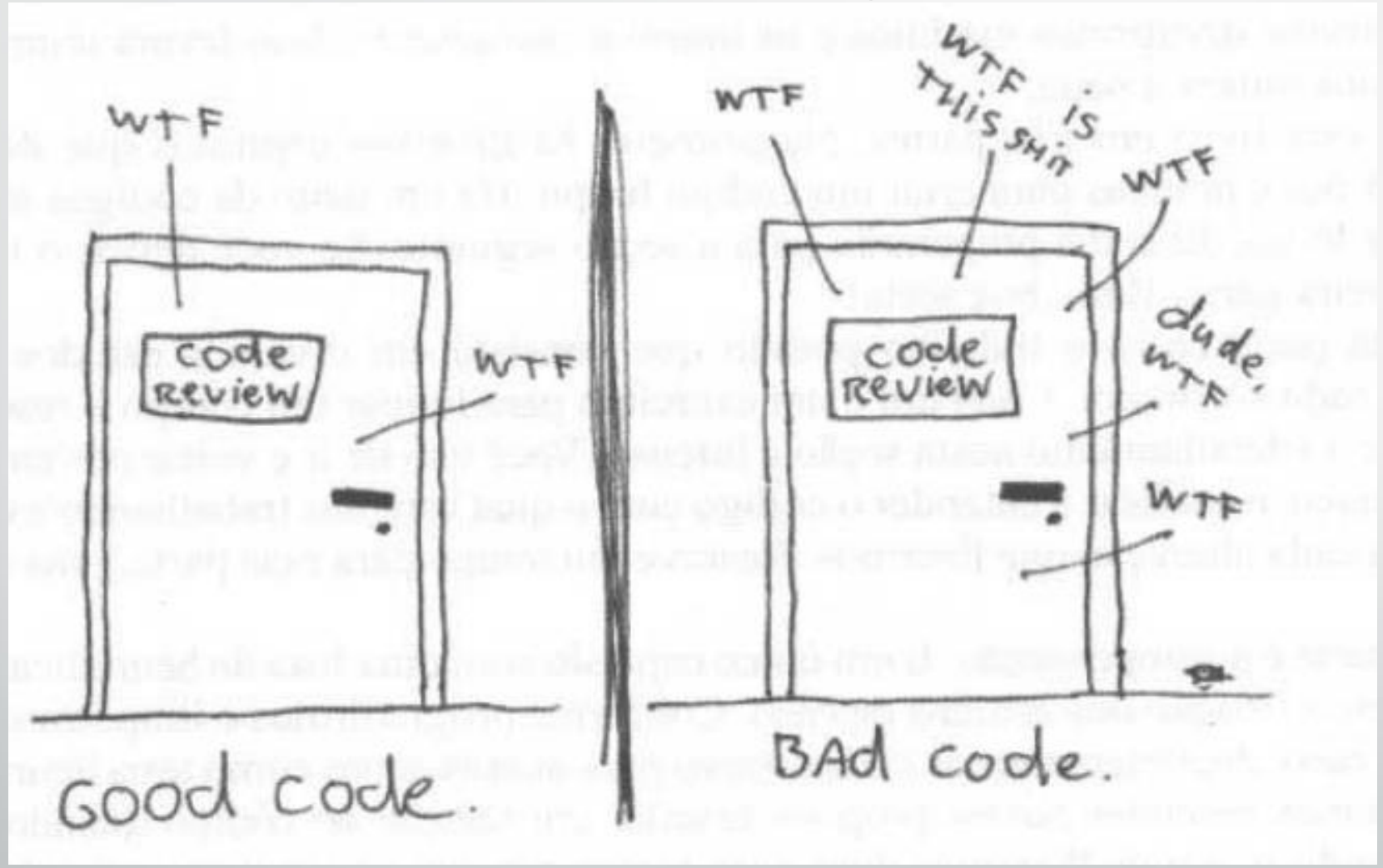
Questões Objetivas	Questões Discursiva
STF 2013	TJDFT 2015
STJ 2015	
TRE PI 2016	
TRE MT 2015	
TCE SC 2016	
TCE PA 2016	
Esse tema tem estado presente em todos os editais da banca CESPE de 2015 para cá.	

Clean Code

- Conceitos Gerais.
- Nomes Significativos.
- Funções.
- Comentários
- Objeto e Estrutura de dados.
- Tratamento de erros.
- Análise discursiva CESPE.



Introdução


WTF
code review
WTF
Good code.
WTF
WTF
WTF is this shit
WTF
code review
dude, WTF
WTF
BAd code.

Motivação para escrever código ruim

- Cronogramas apertados.
- Pressão do gerente.
- Foco na entrega do produto (comprometendo qualidade do produto).
- Desmotivação com o projeto atual
- **Resultado futuro:** A equipe fica estagnada (reparando brechas em códigos mal escritos)

Código limpo (Analogia quadro)

"Simples e direto. Pode ser lido como uma conversa".
Grady Booch

"Um código limpo sempre parece que foi escrito por alguém que se importava".
Michael Feathers

"Você sabe quando está criando um código limpo, quando cada rotina que você lê faz o que se esperava".
Ward Cunningham

Regra do Escoteiro

- Além de escrever código limpo, prezar para sempre **mantê-lo** limpo.
- “Deixe a área do acampamento mais limpa do que como você a encontrou”.

Q1 - [CESPE STF 2013] O desenvolvimento de sistemas mediante a utilização de CLEAN CODE baseia-se em um ciclo curto de repetições, em que o responsável pela codificação descreve testes automatizados que definem uma funcionalidade elicitada. Após se definir o teste, desenvolve-se o código que será validado pela equipe de teste e, posteriormente, refatorado.

Q1 - [CESPE STF 2013] O desenvolvimento de sistemas mediante a utilização de CLEAN CODE baseia-se em um ciclo curto de repetições, em que o responsável pela codificação descreve testes automatizados que definem uma funcionalidade elicitada. Após se definir o teste, desenvolve-se o código que será validado pela equipe de teste e, posteriormente, refatorado. ERRADO.

Nomes Significativos

- Nomes são utilizados em: variáveis, funções, métodos, classes, etc.
- “Diga o que você quer expressar. Expresse o que você quer dizer.” – Robert C Martin

Dicas para bons nomes

- **Use nomes que revelem seu propósito**: “Se um nome requer um comentário, então ele não revela seu propósito”.
  - Ex: int d;
  - Ex2: int diasLetivo;
  - Ex3: int diasMensais;
  - “Um código bem escrito é como uma prosa”.

- **Evitar informações dúbias**: "Usar conceitos similares para montar palavra é *informação.* Usar formatos inconsistentes para palavras leva a uma má interpretação".
  - Ex: utilizar letra 'l' juntamente com 1.
  - Ex2: utilizar letra 'o' juntamente com 0.

- **Use nomes passíveis de busca:** "O tamanho de um nome de variável deve ser proporcional ao tamanho do escopo".
  - Essa é uma característica que pode facilitar ou dificultar futuras alterações em um determinado nome de variável.
  - Ex: Imagine que foi declarado uma variável global para capturar o usuário do sistema a partir do momento que ele se logar. O nome da variável foi definida por **'u'.**
  - Quais as implicações disso???

Nomes de classes e métodos

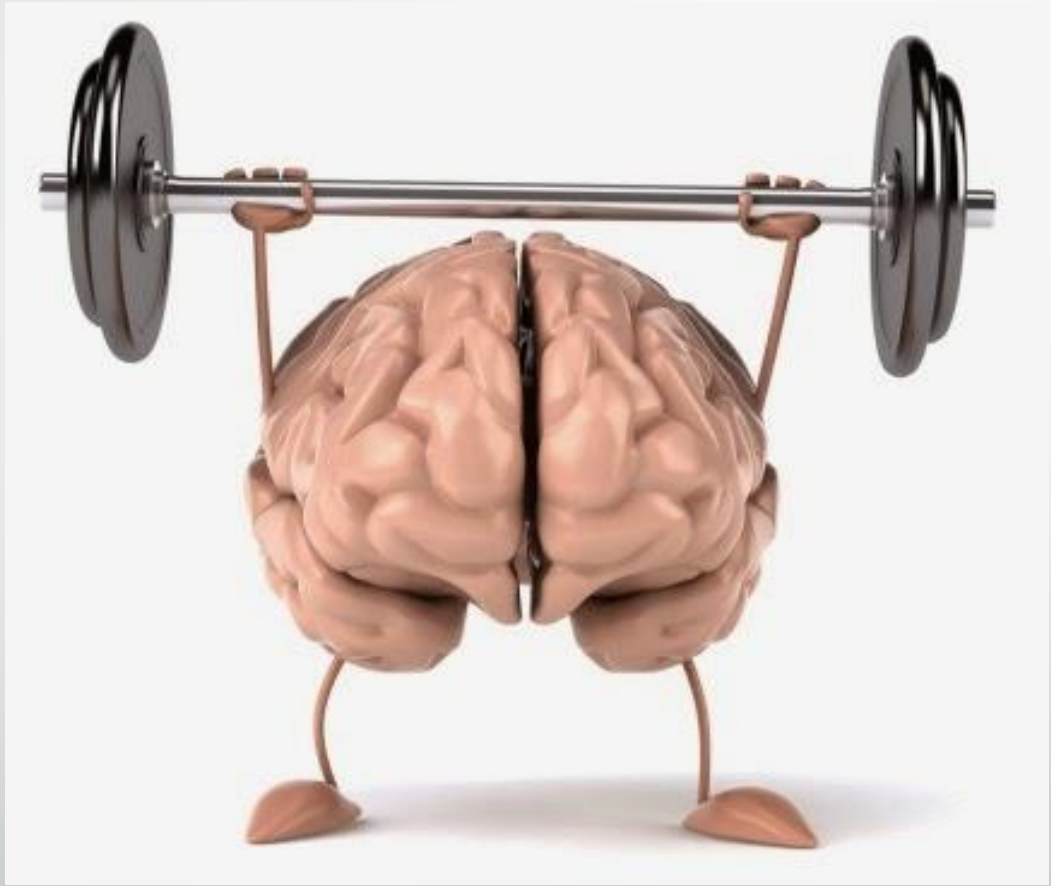
- **Métodos:** Nomes de métodos devem ter verbos (que denotam qualquer tipo de ação).
  - Ex: cobrarCliente(), excluirConta(), comprarLivro().
- **Classe**: Nomes de classe e/ou objeto devem ser nomeados com ***substantivos.***
  - Ex: Cliente, Conta, Livro, etc.

Casos extraordinários

- Utilize nomes a partir do domínio de solução
  - Diz respeito a vivência do dia-a-dia do programador.
  - Ex: AccountVisitor (variável referindo-se a conta de um visitante, mas que também enfatiza o nome de um dos padrões de projeto GOF).
- Utilize nomes a partir do domínio do problema.
  - Diz respeito ao domínio do negócio, e consequentemente à vivência do usuário.
  - Força o programador a entender cada vez mais sobre o domínio do negócio.

Conclusões

- Geralmente, poucos dão importância à utilização de bons nomes.
- Preocupam-se em desenvolver rapidamente o sistema.
- Ou aprender as tecnologias do momento.
- A consequência disso num futuro próximo é apenas uma: “Apagar incêndios”.



Q1 – [CESPE TRE PI 2016] A segurança do código é vital, por isso os programadores devem deixar o código o mais obscuro possível.

Q2 – [CESPE TRE PI 2016] Os nomes das variáveis devem ser simplificados, de forma a não criar códigos gordos (fat codes) — por exemplo, o uso de x para o nome de uma variável é mais apropriado que MediadosAlunosAprovados.

Q1 – [CESPE TRE PI 2016] A segurança do código é vital, por isso os programadores devem deixar o código o mais obscuro possível. ERRADO.

Q2 – [CESPE TRE PI 2016] Os nomes das variáveis devem ser simplificados, de forma a não criar códigos gordos (fat codes) — por exemplo, o uso de x para o nome de uma variável é mais apropriado que MediadosAlunosAprovados. ERRADO.

Q3 – [CESPE TRE MT 2015] Os nomes utilizados devem ser pronunciáveis e devem ter sentido conhecido.

Q4 – [CESPE TRE MT 2015] Os nomes de classes devem ser verbos no infinitivo e os de métodos devem ser substantivos.

Q3 – [CESPE TRE MT 2015] Os nomes utilizados devem ser pronunciáveis e devem ter sentido conhecido. CERTO

Q4 – [CESPE TRE MT 2015] Os nomes de classes devem ser verbos no infinitivo e os de métodos devem ser substantivos. ERRADO

Q5 – [CESPE TRE PI 2016] Se um valor deve ser utilizado em múltiplos locais do código, é imperativo atribuir esse valor a uma variável ou a uma constante com nome amigável.

Q6 – [CESPE TRE PI 2016] Para customizar o código, deve-se utilizar o mesmo termo para duas diferentes ideias.

Q5 – [CESPE TRE PI 2016] Se um valor deve ser utilizado em múltiplos locais do código, é imperativo atribuir esse valor a uma variável ou a uma constante com nome amigável. CERTO.

Q6 – [CESPE TRE PI 2016] Para customizar o código, deve-se utilizar o mesmo termo para duas diferentes ideias. ERRADO.

Q7 – [CESPE TRE PI 2016] As classes devem possuir nome amigável oriundo de verbos, escolhidos no infinitivo, e não no gerúndio.

Q8 – [CESPE TRE MT 2015] Os nomes de funções e de métodos devem ser longos e descritivos.

Q7 – [CESPE TRE PI 2016] As classes devem possuir nome amigável oriundo de verbos, escolhidos no infinitivo, e não no gerúndio. ERRADO.

Q8 – [CESPE TRE MT 2015] Os nomes de funções e de métodos devem ser longos e descritivos. ERRADO.

Q9 – [CESPE STF2013] Os nomes de classes devem conter verbos, ao passo que os métodos devem ser indicados por substantivos.

Q9 – [CESPE STF2013] Os nomes de classes devem conter verbos, ao passo que os métodos devem ser indicados por substantivos. ERRADO.

Gabarito

Q1 - ERRADO

Q2 - ERRADO

Q3 - CERTO

Q4 - ERRADO

Q5 - CERTO

Q6 - ERRADO

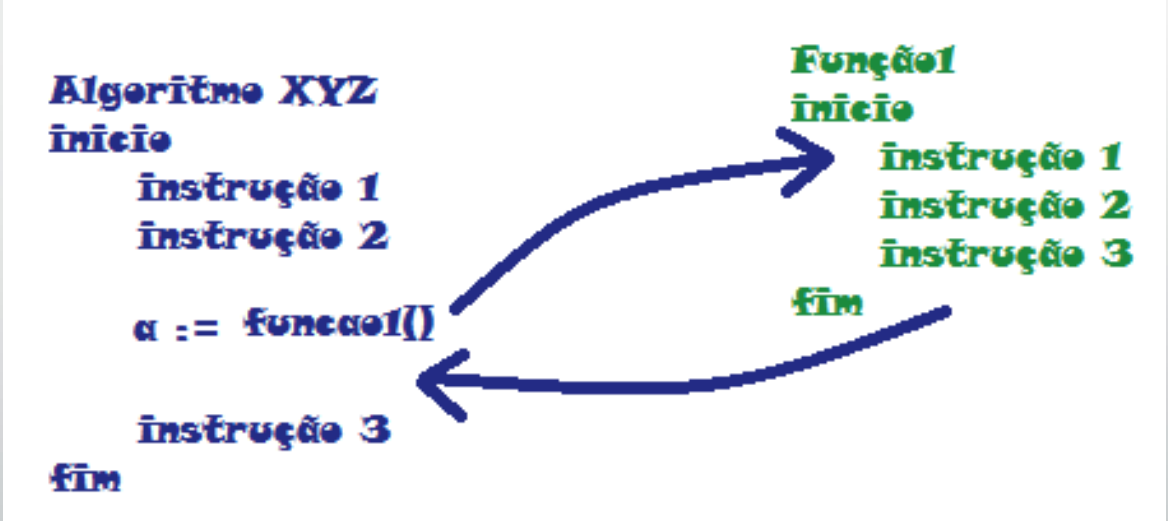
Q7 - ERRADO

Q8 - ERRADO

Q9 - ERRADO

Funções

- Começaram a ser utilizadas desde a abordagem procedural de programação
- Tendo como um dos principais objetivos, a reutilização de código

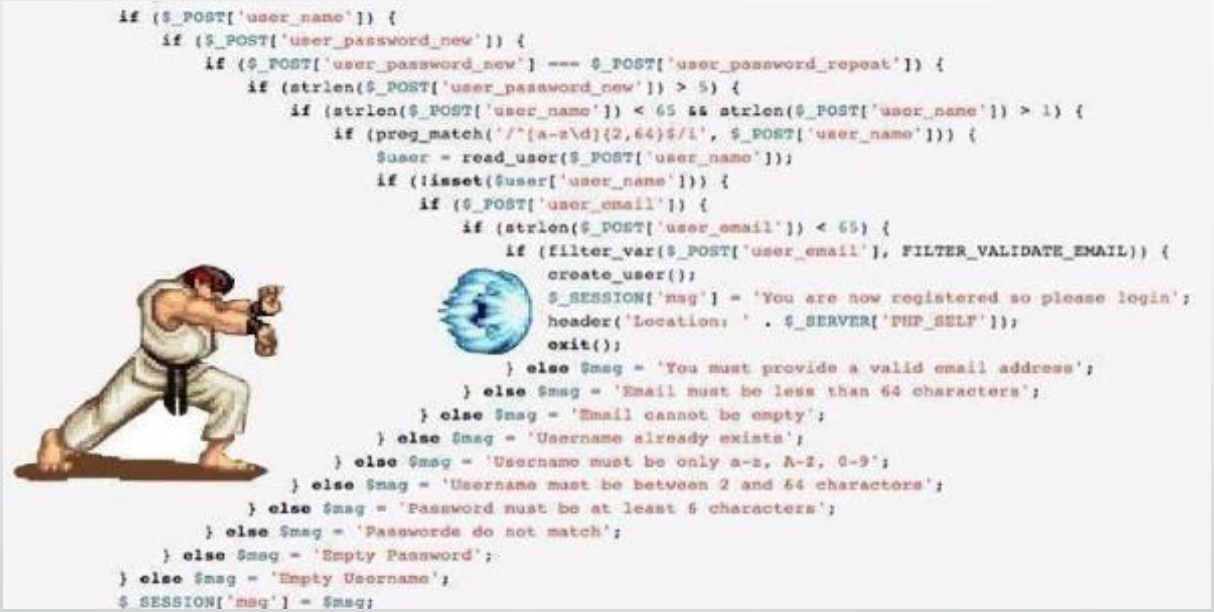


Características de uma boa função

- **Pequena**: Antigamente um parâmetro de tamanho de função, era se ela coubesse nos monitores da época (que eram bem limitados).

- Em uma visita a Kent Beck, Robert constatou que suas funções **tinha de 2 a 4 linhas.**

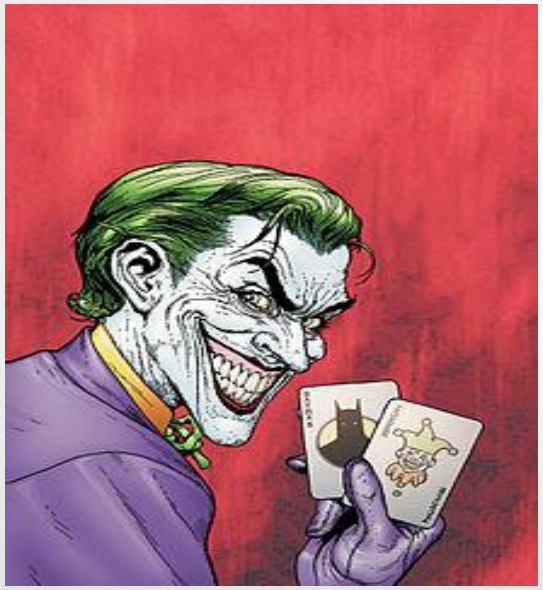
- **Endentação**: Instruções como if, else e while devem ter apenas uma linha (preferência a chamada de uma outra função).



```
if ($_POST['user_name']) {
    if ($_POST['user_password_new']) {
        if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
            if (strlen($_POST['user_password_new']) > 5) {
                if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                    if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                        $user = read_user($_POST['user_name']);
                        if (!isset($user['user_name'])) {
                            if ($_POST['user_email']) {
                                if (strlen($_POST['user_email']) < 65) {
                                    if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                        create_user();
                                        $_SESSION['msg'] = 'You are now registered so please login';
                                        header('Location: ' . $_SERVER['PHP_SELF']);
                                        exit();
                                    } else $msg = 'You must provide a valid email address';
                                } else $msg = 'Email must be less than 64 characters';
                            } else $msg = 'Email cannot be empty';
                        } else $msg = 'Username already exists';
                    } else $msg = 'Username must be only a-z, A-Z, 0-9';
                } else $msg = 'Username must be between 2 and 64 characters';
            } else $msg = 'Password must be at least 6 characters';
        } else $msg = 'Passwords do not match';
    } else $msg = 'Empty Password';
} else $msg = 'Empty Username';
$_SESSION['msg'] = $msg;
```

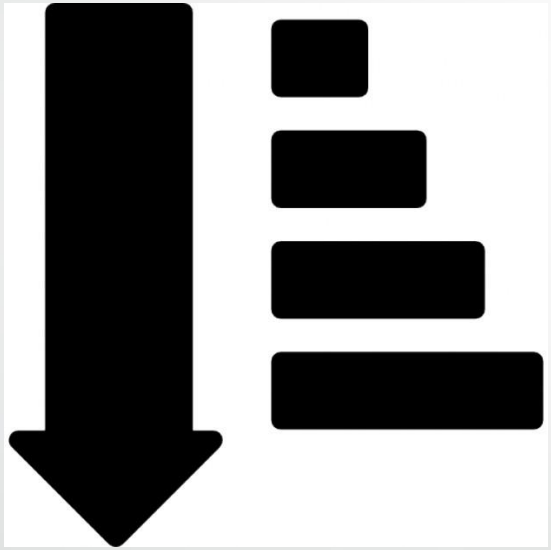
- O nível de endentação deve ser, de **1 ou no máximo 2 linhas**.

- **Faça apenas uma coisa**: Às vezes, cria-se uma função genérica que possa ser reutilizada em muitos lugares e que resolva mais de um problema.



- Funções ‘coringas’ além de dificultar a legibilidade do código, dificulta também a cobertura de testes no código.

- **Regra decrescente**: Dá ênfase à legibilidade do código, sobretudo numa codificação sequencial lógica.

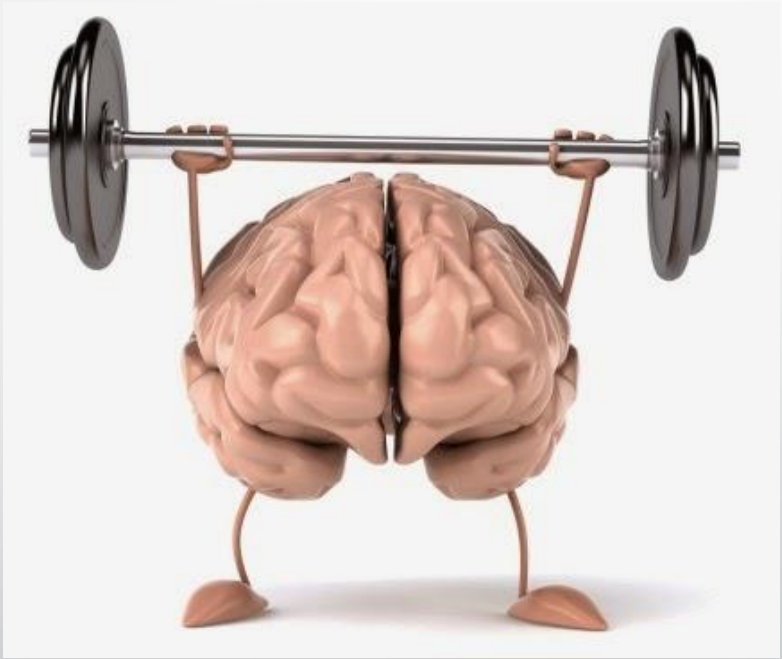


- Analogia a uma prosa ou poesia.

- **Nomes descritivos**: Nomes extensos e descritivos são melhores que nomes pequenos e enigmáticos.
- Esse princípio se encaixa com todos os assuntos que comentamos na lição de Nomes Significativos.
- “É difícil superestimar o valor de bons nomes”.

Parâmetro de Funções

- A quantidade ideal de parâmetros é **zero**. Principalmente, por causa de testes.
- Sempre que necessário, evitar 3 ou mais parâmetros.
  - Função Mônades (1 parâmetro): Validação de CNPJ de uma empresa.
  - Função Diáde (2 parâmetros): Muito utilizado em funções para desenho de primitivas em canvas. Ex: Point p = new Point(x,y);
  - Função Triáde (3 parâmetros): Um exemplo seria o método assertEquals que captura o estado da interação atual, o valor esperado e o valor atual, respectivamente.



Q1 – [CESPE TRE MT 2015] Os parâmetros devem ser aglutinados em funções, e cada função deve ter, no máximo, três parâmetros.

Q2 – [CESPE TRE MT 2015] O comando return deve ser evitado, ao passo que continue e break devem ser priorizados, assim como o goto.

Q1 – [CESPE TRE MT 2015] Os parâmetros devem ser aglutinados em funções, e cada função deve ter, no máximo, três parâmetros. ERRADO.

Q2 – [CESPE TRE MT 2015] O comando return deve ser evitado, ao passo que continue e break devem ser priorizados, assim como o goto. ERRADO.

Q3 – [CESPE TCE SC 2016] De acordo com as diretivas do Clean Code, o número de argumentos de uma função não deve ser igual ou superior a três, devido a sua influência no entendimento da função.

Q4 – [CESPE TCE PA 2016] As práticas de clean code recomendam que as funções tenham, no máximo, vinte linhas, e até dois níveis de indentação.

Q3 – [CESPE TCE SC 2016] De acordo com as diretivas do Clean Code, o número de argumentos de uma função não deve ser igual ou superior a três, devido a sua influência no entendimento da função. CERTO

Q4 – [CESPE TCE PA 2016] As práticas de clean code recomendam que as funções tenham, no máximo, vinte linhas, e até dois níveis de indentação. CERTO.

Q5 – [CESPE TRE MT 2015] Os nomes de funções e de métodos devem ser longos e descritivos.

Q6 – [CESPE STJ 2015] No contexto de clean code, as funções devem ter tamanho reduzido.

Q5 – [CESPE TRE MT 2015] Os nomes de funções e de métodos devem ser longos e descritivos. **ERRADO.**

Q6 – [CESPE STJ 2015] No contexto de clean code, as funções devem ter tamanho reduzido. CERTO.

Gabarito

Q1 - ERRADO

Q2 - ERRADO

Q3 - CERTO

Q4 – CERTO

Q5 – ERRADO

Q6 – CERTO

<!-- Comentários !-->

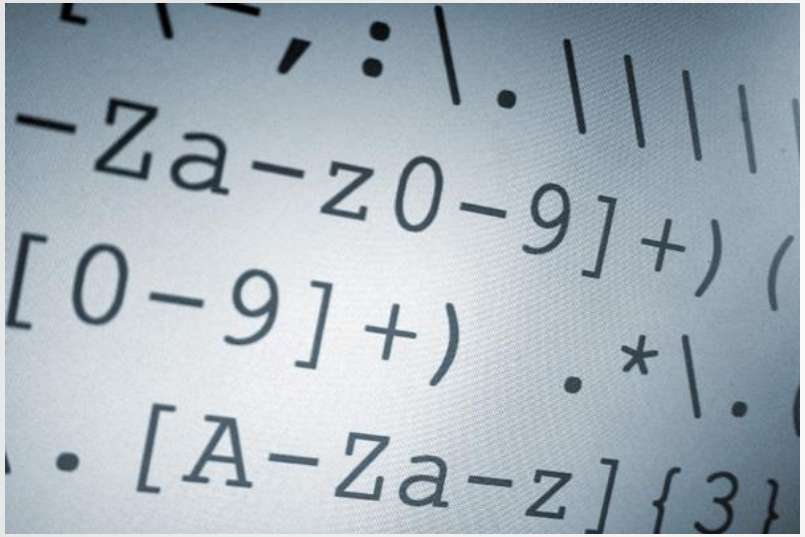
- “Não insira comentários num código ruim, reescreva-o”.
- “Nada pode ser útil quanto um comentário bem colocado”.
- “O uso adequado de comentários é compensar nosso fracasso em nos expressar no código”.

Comentários Bons

- **Comentários legais**: Informações sobre direitos autorais do código são considerado informações relevantes pra se colocar no início do arquivo.

```
// Direitos autorais (C) 2003,2004,2005 por Object Mentor, Inc. Todos
os direitos reservados.
// Distribuido sob os termos da versão 2 ou posterior da Licenca
Publica Geral da GNU.
```

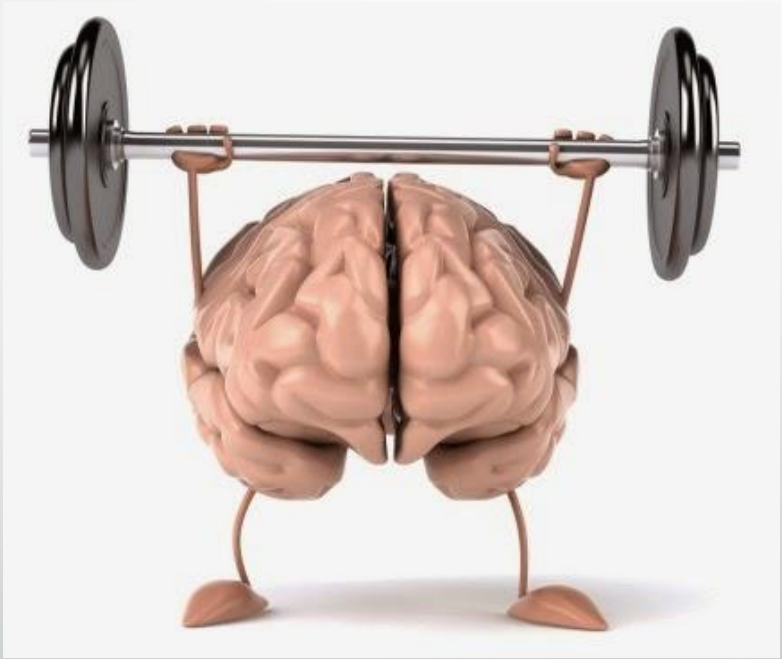
- **Comentários informativos**: Imagina uma função que contém uma expressão regular para validar tal informação.



- **Explicação da intenção:** Esse tipo de comentário é bem importante, principalmente se um outro programador vai dar manutenção num código que ele não conhece muito bem sobre sua linguagem de domínio.

- **Comentário TODO:** É interessante colocar comentários de algo que ainda é pra ser feito dentro do código.

- **Alerta sobre consequências: :** É útil alertar outros programadores sobre as implicações de testar ou executar determinada parte do software.
  - Explicação de alguma linha ou parâmetro que esteja comentado.



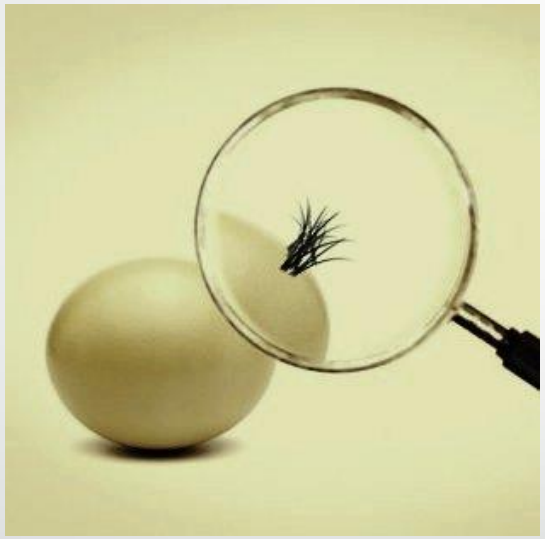
Q1 – [CESPE TRE MT 2015] De acordo com as práticas de clean code, comentários em um código-fonte servem para compensar um código mal escrito, devendo, portanto, ser evitados.

Q1 – [CESPE TRE MT 2015] De acordo com as práticas de clean code, comentários em um código-fonte servem para compensar um código mal escrito, devendo, portanto, ser evitados. ERRADO.

Comentários Ruins

- **Comentários redundantes**: Quando um código já é autoexplicativo, qualquer tipo de comentário é desnecessário.

- **Comentários enganadores**: Acontece muito quando é atualizado partes de um código, mas se esquecem de atualizar também os comentários.



- **Comentários longos**: Usar o próprio código pra controlar o seu versionamento.

```
 * Changes (from 11-Oct-2001)
 * --------------------------
 * 11-Oct-2001 : Re-organised the class and moved it to new package
 *               com.jrefinery.date (DG);
 * 05-Nov-2001 : Added a getDescription() method, and eliminated NotableDate
 *               class (DG);
 * 12-Nov-2001 : IBD requires setDescription() method, now that NotableDate
 *               class is gone (DG);  Changed getPreviousDayOfWeek(),
 *               getFollowingDayOfWeek() and getNearestDayOfWeek() to correct
 *               bugs (DG);
 * 05-Dec-2001 : Fixed bug in SpreadsheetDate class (DG);
 * 29-May-2002 : Moved the month constants into a separate interface
 *               (MonthConstants) (DG);
 * 27-Aug-2002 : Fixed bug in addMonths() method, thanks to N???levka Petr (DG);
 * 03-Oct-2002 : Fixed errors reported by Checkstyle (DG);
 * 13-Mar-2003 : Implemented Serializable (DG);
 * 29-May-2003 : Fixed bug in addMonths method (DG);
 * 04-Sep-2003 : Implemented Comparable.  Updated the isInRange javadocs (DG);
 * 05-Jan-2005 : Fixed bug in addYears() method (1096282) (DG);
```

Q2 – [CESPE TRE MT 2015] O uso de comentários é uma das técnicas de código limpo que, em conjunto com a refatoração de códigos, permite aumentar a produtividade de desenvolvimento de códigos.

Q2 – [CESPE TRE MT 2015] O uso de comentários é uma das técnicas de código limpo que, em conjunto com a refatoração de códigos, permite aumentar a produtividade de desenvolvimento de códigos.
CERTO.

Gabarito

Q1 - ERRADO

Q2 - CERTO

Objeto e Estrutura de dados

- Esse capítulo enfatiza muito sobre o encapsulamento de dados.

- “Há um motivo para declararmos nossas variáveis como privadas. Não queremos que ninguém dependa delas”.

Objeto X Estrutura de dados

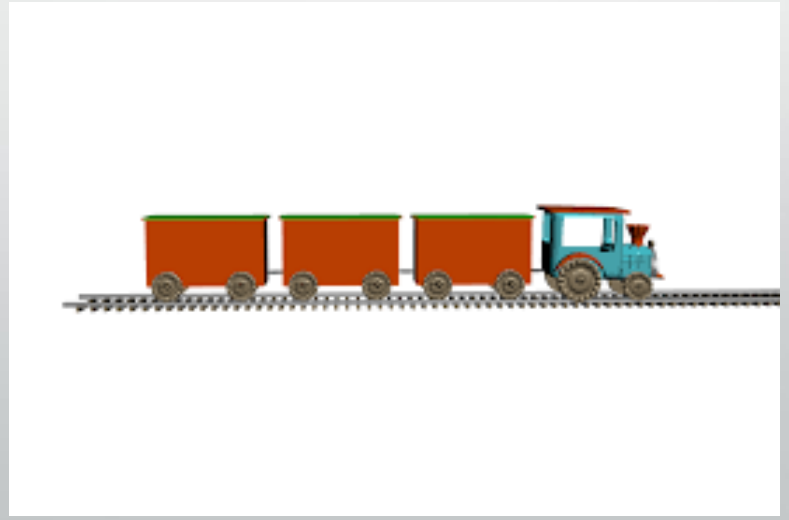
- Código procedural dificulta a adição de novas estruturas de dados, pois todas as **funções** precisam ser **modificadas**.
- Código orientação à objeto dificulta a adição de novas funções, pois todas as **classes** tem que mudar.
- Ex: Para adicionar novos tipos de dados, a abordagem utilizando **objeto** é melhor.
- No entanto, para adicionar novas funções, o mais adequado é utilizar **estrutura de dados**.

Lei de Demétrio

- “Um módulo não deve enxergar o interior dos objetos que ele manipula”.
- Esconder os dados e expor apenas as operações.
- “Em outras palavras, fale apenas com conhecidos, não com estranhos”.

Acoplamento e vagões de trem

- É necessário, mas com moderação.
- Grau de dependência entre diferentes funções, classes, métodos em um código.
- Alto acoplamento é algo que deve ser evitado, pois fica difícil de testar e ainda por cima, manter um código que tenha alto acoplamento.



Q1- [CESPE TCE PA 2016] No contexto de clean code, o conceito de objetos é semelhante ao de estruturas de dados, devendo os dados e as funções ficar expostos para permitir a inclusão de novos dados e de novas funções.

Q1- [CESPE TCE PA 2016] No contexto de clean code, o conceito de objetos é semelhante ao de estruturas de dados, devendo os dados e as funções ficar expostos para permitir a inclusão de novos dados e de novas funções. **ERRADO.**

Gabarito

Q1 - ERRADO

Tratamento de erros

- “Em suma, as coisas podem dar errado, e quando isso ocorre, nós como programadores, somos responsáveis por certificar que o código faça o que seja preciso fazer”.

- Apesar disso, deve haver uma conexão com código limpo bem clara.

Características

- **Use exceções em vez de retornar código**: Até alguns tempos atrás, existiam linguagens que não suportavam exceções.
  - Para resolver isso, era criado flags personalizada de erros ou retornava-se um código de erro que representava um determinado estado.
  - Porém, hoje recomenda-se utilizar o recurso de exceções que já está presente em bastantes linguagens.
  - Claúsula **throws**; **Try-Catch-Finaly.**

- **Estrutura Try-Catch-Finally**: Uma das abordagens mais utilizadas para tratamento de erros.
  - Bloco **try** são como transações. Só serão efetivadas se o código que estiver dentro do bloco catch se manter em um estado consistente.
  - Se o código não apresentar um estado consistente, o bloco **catch** lançará uma exceção.
  - O bloco **finally** será sempre executado, estando num estado consistente ou não.

- **Forneça exceções com contexto**: Cada exceção lançada deve ter um contexto suficiente para nortear a solução desta exceção.
  - Personalizar as mensagens de erro genéricas em um linguagem
  - Ex: Em Java - **NullPointerException**.
  - Mencione a operação em que ocorreu a falha e que tipo de falha ocorreu.

- **Não retorne null**: “Quando retornamos null, basicamente estamos criando mais trabalho para nós mesmo e jogando problemas em cima de nossos chamadores”.
  - Em tempo de execução se for verificado que determinado objeto está nulo
  - É retornado o famoso **NullPointerException**.
  - A medida correta é não apenas, fazer verificações de null nos objetos, mas sim evitar retornar null para os mesmos.

Clean Code – CESPE (Prova TJDFT 2015)

PROVA DISCURSIVA

- Nesta prova, faça o que se pede, usando, caso deseje, o espaço para rascunho indicado no presente caderno. Em seguida, transcreva o texto para a **FOLHA DE TEXTO DEFINITIVO** da prova discursiva, no local apropriado, pois **não será avaliado fragmento de texto escrito em local indevido**.
- Qualquer fragmento de texto que ultrapassar a extensão máxima de linhas disponibilizadas será desconsiderado.
- Na **folha de texto definitivo**, identifique-se apenas no cabeçalho da primeira página, pois **não será avaliado** texto que tenha qualquer assinatura ou marca identificadora fora do local apropriado.
- Ao domínio do conteúdo, serão atribuídos até **40,00 pontos**, dos quais até **2,00 pontos** serão atribuídos ao quesito apresentação (legibilidade, respeito às margens e indicação de parágrafos) e estrutura textual (organização das ideias em texto estruturado).

Redija um texto dissertativo acerca do *clean code*. Seu texto deverá abordar, necessariamente,

- a escolha de nomenclatura de variáveis, métodos e classes; **[valor: 14,00 pontos]**
- a abrangência e parametrização de funções; **[valor: 12,00 pontos]**
- a necessidade e tipos de comentários. **[valor: 12,00 pontos]**

1 - Um *clean code* é obtido por meio da utilização de várias técnicas, a fim de tornar o código simples, de fácil leitura e sem duplicações. As técnicas utilizadas para se obter um *clean code* incluem a escolha de nomes de variáveis, métodos e classes, que devem informar ao desenvolvedor por que existem, o que fazem e como são usados. As variáveis devem ter nomes significativos e que revelem o objetivo, por exemplo, nomeCliente e saldoAtual; as classes devem ser substantivos, por exemplo, Cliente e ContaCorrente; e os métodos devem ser representados por verbos ou frases verbais, por exemplo, pagarFatura e transferirDinheiro.

2 - As funções devem ser pequenas, com nomes significativos e que façam apenas uma coisa. Devem possuir até dois níveis de identificação, para facilitar o entendimento, e ter, no máximo, dois parâmetros.

3 - Os comentários devem ser realmente úteis e devem ser submetidos à manutenção quando for necessário. Os tipos de comentários mais utilizados são o comentário de esclarecimento, no qual se descreve o que foi realizado; a explicação de intenção, na qual se descreve a intenção que orientou uma decisão; e o aviso de consequência, o qual serve para avisar sobre consequências do que foi realizado.

Obrigado!!


Daqui a
um ano,
você vai desejar
ter começado
hoje.
mundowhatsapp.com