Código Limpo:

Habilidades Práticas do Agile Software

...

Capítulo 4: Comentários

Este material foi desenvolvido pelo grupo PET/ADS do IFSP São Carlos

Comentários: um mal necessário

As três faces dos comentários de código:

- A face boa, que faz mais do que qualquer outra explicação poderia.
- A face ruim, que justifica por meio de texto um código longo e confuso.
- A pior face, que é antiquada e está desatualizada com relação ao código.

Fracassos

Utilize comentários apenas em situações que um código de qualidade não é capaz de se auto-explicar.

Melhore sua capacidade de expressão a partir do código-fonte, pois não é viável manter a duplicação código-comentário ao longo da vida de um programa.

Manter um comentário atualizado, relevante e preciso é mais caro do que construir um código inteligível. A única fonte da verdade deve ser o código

Comentários compensam um código ruim?

Qual o motivo de um comentário?

- Confusão
- Desorganização
- Autoconsciência

NÃO!

É melhor organizar seu código!

“É melhor escrever um comentário.”

Explique-se no código!

A expressão condicional no código a seguir parece precisar de um comentário, mas uma refatoração ajuda a explicar diretamente a intenção:

```
public static void main(String[] args) {
//checa se o empregado é elegível a todos os os benefícios
   if((employee.flags && HOURLY_FLAG)&& employee.age >= 65){...}
}
```

```
if(employee.isElegibleForFullBenefits()){...}
```

Comentários Bons: Comentários Legislativos

Muitos dos programas precisam expressar licenças de uso, autoria e outras informações similares.

Nesses casos:

- Não coloque todo o documento legal, aponte para a documentação completa.
- Não coloque informações desnecessárias, como a razão da legislação ser adotada.
- Evite amontoar diferentes informações legais juntas, separando os comentários.

Comentários Bons: Comentários Informativos

O exemplo a seguir descreve um bom comentário, pois ele informa sobre algo que é inerentemente complexo de ser entendido direto no código:

```
public static void main(String[] args) {
   //formato igual a kk:mm:ss EEE, MMM dd, aaaa
   Pattern timeMatcher = Pattern.compile("\\d*:\\d*:\\d* \\w*, \\w* \\d*, \\d*");
   ...
}
```

Esse comentário ajuda a explicar as Strings aceitas pela implementação. Mesmo assim, o comentário poderia ser trocado por uma função ou classe específica para isso.

Comentários Bons: Explicação da Intenção

O motivo das escolhas em um código devem ficar bem claros.

Exemplo:

Classificar os objetos de uma classe como prioritários com relação aos demais é uma ação que pode ser explicada em um comentário.

Comentários Bons: Esclarecimento

O uso de códigos de bibliotecas de terceiros causam dúvidas:

- Pergunta: Por que usou StringBuffer ao invés de StringBuilder?
- Resposta: // O programa será utilizado em rotinas multi-thread

O comentário explica uma decisão do *porquê* foi feita uma escolha no código. Entretanto, assegure-se que isso não poderia ser feito diretamente no próprio código.

"Um comentário bom é um comentário que não foi feito!"

Comentários Bons: Alerta sobre as consequências

Se o código contém uma limitação explícita, pode ser útil esclarecer o motivo.

Atente-se para as características a seguir ao fazer um comentário:

- Precisão: Escreva com nitidez e concisão os motivos da limitação;
- Seriedade: Mesmo piadas contidas podem tirar o foco e rebaixar a validade do comentário;
- Sucintez: Deixe longas explicações para a documentação, não no código.

Comentários Bons: TODOs (a fazer)

Representam comentários sobre tarefas que ainda estão por fazer no seu código.

Motivos válidos para deixar uma tarefa para trás:

- Não podem ser executadas no momento
- Lembretes para futuras atualizações
- Pedidos para terceiros revisarem o código

O TODO é um comentário eliminável, portanto conclua-os o mais rápido possível. **TODOs não são motivos para um código ruim**

Comentários Bons: Destaques e APIs públicas

Comentários de destaque devem ser feitos em porções cruciais de código cujas alterações devem ser feitas cautelosamente.

O uso dos comentários para a documentação de métodos de APIs públicas são mandatórios, pois detalham aos outros o funcionamento do código que será invocado.

A inclusão de comentários sobre a sobrescrita de métodos de APIs públicas podem evitar grandes problemas!

Comentários Ruins: Murmúrios

"Meu código não é claro o suficiente, tanto que nem eu entendi, preciso escrever como eu penso que isso deveria funcionar..."

```
//isso troca um com o outro
void t(int *a, int *b) {
  //guarda o primeiro
  int c = *a;
  //coloca o segundo no primeiro
  *a = *b;
  //devolve o que foi guardado
  *b = c;
}
```

```
void troca (int *primeiro, int *segundo) {
   int auxiliar = *primeiro;
   *primeiro = *segundo;
   *segundo = auxiliar;
}
```

Observe o poder da refatoração

Comentários Ruins: Comentários Redundantes

É necessário escrever o funcionamento de um trecho de código? Não se ele estiver bem escrito.

Explicar o funcionamento de um código bom é um atestado para duvidarem da sua capacidade de codificação.

Essas explicações excessivas também atrapalham a leitura do seu código.

Comentários Ruins: Comentários Enganadores

É o comentário que faz entender o código de uma maneira, quando na verdade ele se comporta de outra totalmente diferente.

Isso gera confusão entre os possíveis utilizadores dos seus códigos!

Não faça comentários assim, a menos que você possa prever todos os tipos de comportamento para tal trecho de código.

Comentários Ruins: Comentários Imperativos

Não existe sentido em comentar por obrigação algo, gerar um Javadoc para cada uma das funções é inviável.

Em especial, não se preocupe em documentar com Javadoc as funções privadas.

Se bons nomes foram escolhidos, comentários desse tipo são totalmente desprezíveis.

Comentários Ruins: Comentários Longos

Antes dos sistemas de controle de versionamento (VCS), longos comentários detalhando alterações eram necessários, diferentemente de hoje.

Lembre-se: O tamanho importa! Seu comentário não é uma redação do Enem, portanto textos grandes devem ser reservados para a documentação final do código

Comentários Ruins: Créditos e Autoria

Muitos programadores anteriores aos VCS costumam colocar em seus códigos comentários com seus nomes creditando a autoria por determinado trecho.

Isso é um comportamento totalmente desnecessário hoje em dia, Já que o VCS armazenam os autores e as atualizações que ocorrem nos códigos.

Comentários Ruins: Comentários Ruidosos

Toda estrutura de código é óbvia para quem programa, não há necessidade em dizer “Isso é um construtor” ou “Isso percorre o array”.

Uma piada interna, um pensamento próprio, uma reclamação direta ou uma briga entre colegas não são coisas para estar em seu código, nem na documentação.

Guarde esse tipo de interação para momentos de relação interpessoal.

Comentários Ruins: Assustadoramente Ruidosos

Javadocs podem tornar-se ruídos se não usados com cautela. Veja um exemplo:

```
/** Nome. */
private String name;
/** Versão. */
private String version;
/** Nome da licença. */
private String licenseName;
/** Versão. */
private String info;
```

Notou algo estranho?

Comentários Ruins: Marcadores de posição

São usados quando precisamos agrupar e recordar o lugar onde colocamos as coisas:

```
//////variáveis////////////////////////////////////

////ações/////////////////////////////////////////
```

Mesmo se necessários por um tempo, acabamos nos acostumando com eles e ignoramos a presença no código.

Se um comentário é ignorado é porque não deveria estar ali.

Comentários Ruins: Troque-os por funções e variáveis

Se o código está muito complicado ou ficou grande, extraia em métodos e variáveis:

```
public static void main(String[] args) {
   /** verifica se a procura é igual ao curso de medicina */
   if(numVagas > procura || procura > 8000 || numVagas < 40 …){...}
}
```

```
public static void main(String[] args) {
   if(isEqualToMed()){...}
}
```

Comentários Ruins: Comentários de Chaves de Fechamento

Observe o código a seguir:

```
while(int j > 10) {
  . . .
  for(int k : int[] nums) {
    . . .
    while(int n > j + k) {
      . . .
    }//while depois for: olha se os valores são compatíveis
  }//for que compara nos números
}//while depois do primeiro while, para cada idade faz 10 vezes
```

Funções devem ser atômicas, ou seja, fazer bem uma coisa e apenas isso. Se existem vários loops na sua função é sinal que ela precisa ser particionada.

Comentários Ruins: Comentários HTML

A maioria dos Javadocs utiliza porções de códigos HTML para que o comentário seja mostrado em páginas web. Como se sabe, os códigos HTML são feios por si só.

Caso haja necessidade de algo do tipo, refaça os comentários em HTML, geralmente eles são extraídos da pior maneira possível.

Comentários Ruins: Informações não locais

Se a informação não é do próprio código ela não deve estar no código, a mantenha nos meios externos.

Se o comentário não está ligado a algo diretamente relacionado com o código ele não deve estar lá.

Por exemplo: Se referir a uma porta de conexão → isso é parte das configurações do sistema, não parte do código (a menos que isso seja parte da lógica).

Comentários Ruins: Informações excessivas

Por mais correlata que a informação seja ao código, analisar sua relevância é imprescindível.

Adicionar detalhes históricos ou detalhes irrelevantes são práticas a serem evitadas:

```
public static void main(String[] args) {
//O algoritmo do menor caminho é um algoritmo que encontra a menor
//possibilidade para conectar dois pontos passando pela menor
//quantidade de pontos possíveis, nesta implementação usei “Nós”
para //determinar o intervalo de dois pontos que desejo encontrar,
origem é //o nó de onde deve partir e o destino é o nó onde deve se
chegar, a //função retorna o tamanho do menor caminho porque preciso do
tamanho //do menor caminho para encontrar qual o menor espaço de tempo
que uma //goteira vai escorrer pela prateleira em 5 minutos. . .
   int goteira = menorCaminhoEntreNos(Nos, Origem, Destino)
}
```

Comentários Ruins: Conexões misteriosas

A conexão entre um comentário e o que ele descreve deve ser óbvia, se seu comentário precisa de um comentário algo está errado.

Observe se é possível entender o código a seguir:

```
public static void main(String[] args) {
//Começa com um array grande o bastante para conter todos os pixels,
//mais os bytes de filtragem e 200 bytes extras para as informações
de //cabeçalho
   this.pngBytes = new byte[((this.width + 1)* this.height * 3) + 200];
}
```

Comentários Ruins: cabeçalhos para as funções

Funções curtas não requerem muita explicação. Um nome bem selecionado para uma função pequena que faça apenas uma coisa costuma ser melhor do que um comentário.

Exemplo:

```
void troca (int *primeiro, int *segundo) {
   int auxiliar = *primeiro;
   *primeiro = *segundo;
   *segundo = auxiliar;
}
```

Comentários Ruins: Documentação em códigos não públicos

Não gere documentação excessiva e desnecessária em códigos que não serão utilizados pelo público.

Documentação é burocracia explanatória, se o código não for público, por que investir em burocracia extensa?

Sugestão: pense melhor em nomes de variáveis e funções que sejam mais bem explicativas para os poucos que terão acesso ao código.

Comentários: Evite-os

Todo comentário é um comentário ruim em potencial.

Foque na refatoração, criando funções pequenas e escolhendo nomes significativos.

Comentários que são necessários geralmente descrevem o porquê ou o porquê não.

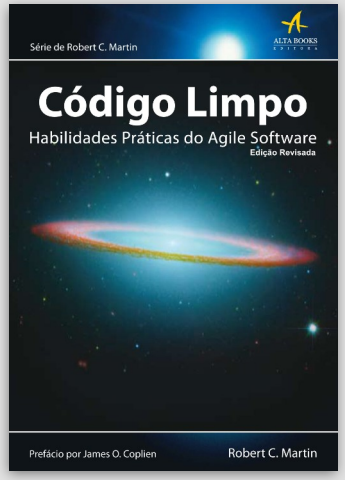
Se o seu código está muito confuso, reinicie, refatore, repense e evite comentários.

Sobre o autor

Robert C. “Uncle Bob” Martin é desenvolvedor e consultor de software desde 1990. Ele é o fundador e o presidente da Object Mentor, Inc., uma equipe de consultores experientes que orientam seus clientes no mundo todo em C++, Java, C#, Ruby, OO, Padrões de Projeto, UML, Metodologias Agile e eXtreme Programming.

Este conjunto de slides foi elaborado a partir da obra:

MARTIN, Robert. **Código Limpo:** Habilidades Práticas do Agile Software. 1. ed. Rio de Janeiro: Alta Books, 2009.



A equipe

Luana Monteiro,
Autora

Aluna e ex-bolsista do PET/ADS
LinkedIn

Lucas Oliveira,
Revisor

Professor de Computação, é tutor do PET/ADS desde janeiro de 2023.
LinkedIn