

TÓPICO 05 - COMENTÁRIOS

Clean Code - Professor Ramon Venson - SATC 2025

Comentários

Comentários são utilizados desde os primórdios da programação para documentar o código.

É uma forma de comunicação que é usada exclusivamente entre programadores.

```
01. #include <iostream>
02.
03. using namespace std;
04.
05. int main()
06. {
07.     /*
08.         Este eh programa feito em C++ que imprime uma mensagem
09.         de texto na tela para que o usuário possa ler.
10.     */
11.
12.     cout << "C++ EH A MELHOR LINGUAGEM"; // Imprime a mensagem
13.
14.     return 0; // Fim do programa
15. }
```

"Comentários são sempre fracassos."

Clean Code

Um comentário pode ser uma ferramenta útil para explicar o que está acontecendo em um trecho de código, mas deve ser usado com moderação.

O comentário existe para explicar aquilo que não conseguimos fazer com a linguagem de programação.



Comentários compensam código ruim

Quanto mais bem escrito o código,
menor a necessidade de
comentários.

Ao invés de:

```
// Verifica se o funcionário é elegível para benefícios completos  
if ((employee.flags & HOURLY_FLAG) && (employee.age < 65))
```

Use:

```
if (employee.isEligibleForFullBenefits())
```

Licenças de Software

Por questões legais, muitos projetos precisam incluir informações sobre a licença de software em cada um dos arquivos do projeto.

Felizmente, a maioria dos editores de código possui plugins que adicionam essas informações automaticamente ou escondem enquanto o usuário trabalha no arquivo.

Onde for possível, é recomendado que os termos da licença sejam apresentados num documento externo.

```

/*****
/*  main.h                                     */
/*****
/*      This file is part of:                 */
/*      GODOT ENGINE                         */
/*      https://godotengine.org               */
/*****
/* Copyright (c) 2014-present Godot Engine contributors (see AUTHORS.md). */
/* Copyright (c) 2007-2014 Juan Linietsky, Ariel Manzur.                  */
/*                                                                           */
/* Permission is hereby granted, free of charge, to any person obtaining */
/* a copy of this software and associated documentation files (the        */
/* "Software"), to deal in the Software without restriction, including     */
/* without limitation the rights to use, copy, modify, merge, publish,    */
/* distribute, sublicense, and/or sell copies of the Software, and to     */
/* permit persons to whom the Software is furnished to do so, subject to  */
/* the following conditions:                                               */
/*                                                                           */
/* The above copyright notice and this permission notice shall be         */
/* included in all copies or substantial portions of the Software.        */
/*                                                                           */
/* THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,       */
/* EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF    */
/* MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.*/
/* IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY   */
/* CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,  */
/* TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE     */
/* SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.                */
*****/
```


Explicação da Intenção

Um comentário que explica a razão de um trecho de código ao invés de descrever como ele funciona, por vezes, é útil:

```
// os espaços em branco são substituídos por %20 para evitar erros de URL  
String url = rawUrl.replaceAll("\\s+", "%20");
```

Alerta de Consequências

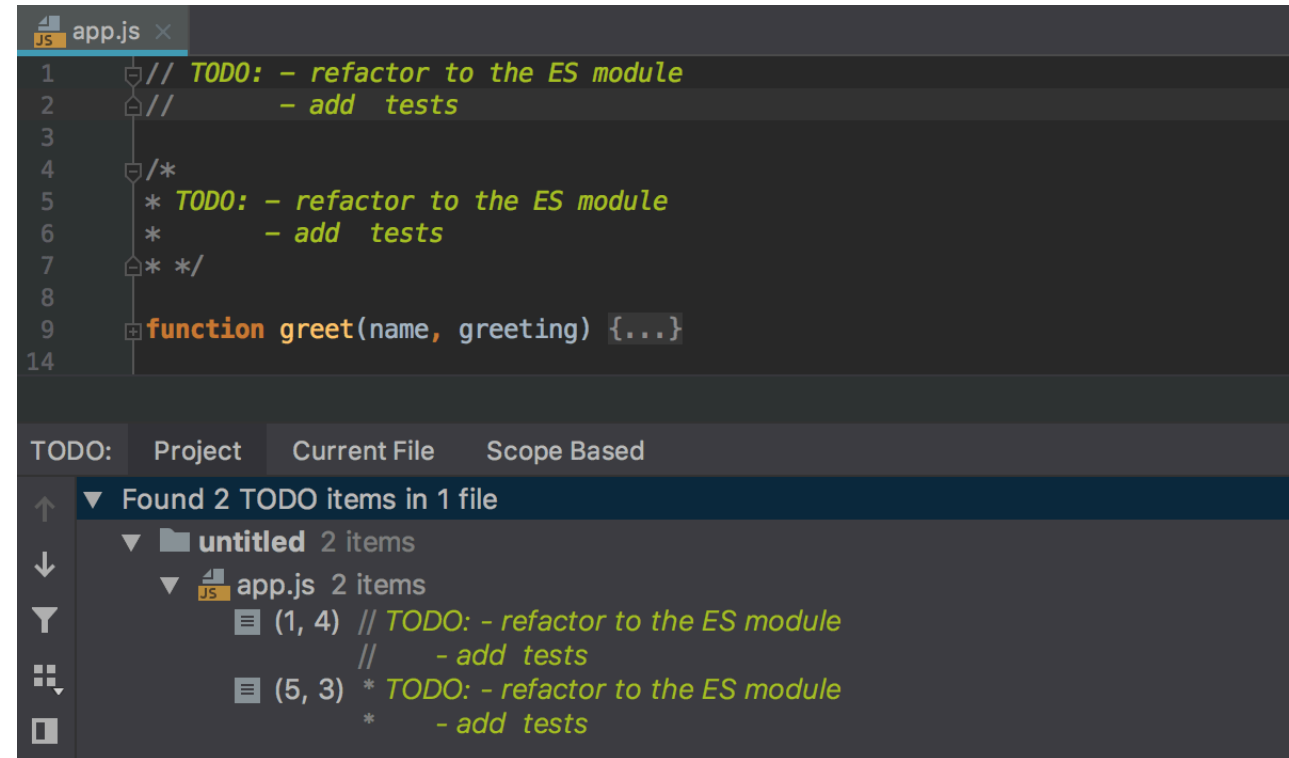
Partes críticas de um código também podem ser comentadas para alertar sobre as consequências de uma mudança.

```
// Executar esse teste apenas se tiver tempo disponível
/*
public void _testWithReallyBigFile() {
    // ...
}
*/
```

Comentários TODO

Comentários que indicam que uma tarefa ainda precisa ser feita geralmente são marcados com `TODO` .

Essa prática é comumente usada para indicar que uma tarefa ainda não foi implementada.



The screenshot shows a code editor with a file named `app.js`. The code contains two TODO comments: a line comment on lines 1-2 and a block comment on lines 4-7. Both comments indicate the need to refactor to the ES module and add tests. Below the code editor, a TODO list panel is visible, showing the search results for the current file. The panel indicates that 2 TODO items were found in 1 file. The items are listed with their line numbers and the text of the comments.

```
1 // TODO: - refactor to the ES module
2 //   - add tests
3
4 /*
5  * TODO: - refactor to the ES module
6  *   - add tests
7  */
8
9 function greet(name, greeting) {...}
14
```

TODO: Project Current File Scope Based

Found 2 TODO items in 1 file

- untitled 2 items
 - app.js 2 items
 - (1, 4) // TODO: - refactor to the ES module
// - add tests
 - (5, 3) * TODO: - refactor to the ES module
* - add tests

Documentação de API

Códigos que serão distribuídos devem ter documentação de API. Geralmente utiliza-se ferramentas como Javadoc ou Doxygen para gerar documentação de API, que deve ser mantida atualizada e devem ser incluídas no código.

Não é necessário gerar documentação de API se o código não for distribuído.

```
-- BOA PRÁTICA #1: COMENTÁRIOS

-- CÓDIGO PARA ALTERAR VALORES DENTRO DO BANCO DE DADOS
-- Autor: Marcus Cavalcanti e João Lira
-- Data Criação: 11/04/2022
-- Última Atualização: 30/08/2022
-- Descrição: ....

-- Comentário #1: Início do Código
-- Dois Parâmetros: Id que significa o id do valor, e novo_valor será o novo valor atribuído
CREATE OR ALTER PROCEDURE altera_valor(@id INT, @novo_valor FLOAT)
AS
BEGIN
    -- Comentário #2: Início da Transação de Atualização (Update)
    BEGIN TRANSACTION

    -- A última alteração no código foi feita nessa linha
    UPDATE Carro
    SET valor = @novo_valor
    WHERE id_carro = @id

    -- Comentário #3: Efetivando a alteração no banco através de um COMMIT
    COMMIT
```

Murmúrios

Comentários que adicionam a experiência pessoal do programador e que não auxiliam na compreensão do código devem ser suprimidos.

```
-- BOA PRÁTICA #1: COMENTÁRIOS

-- CÓDIGO PARA ALTERAR VALORES DENTRO DO BANCO DE DADOS
-- Autor: Marcus Cavalcanti e João Lira
-- Data Criação: 11/04/2022
-- Última Atualização: 30/08/2022
-- Descrição: ....

-- Comentário #1: Início do Código
-- Dois Parâmetros: Id que significa o id do valor, e novo_valor será o novo valor atribuído
CREATE OR ALTER PROCEDURE altera_valor(@id INT, @novo_valor FLOAT)
AS
BEGIN
    -- Comentário #2: Início da Transação de Atualização (Update)
    BEGIN TRANSACTION

    -- A última alteração no código foi feita nessa linha
    UPDATE Carro
    SET valor = @novo_valor
    WHERE id_carro = @id

    -- Comentário #3: Efetivando a alteração no banco através de um COMMIT
    COMMIT
```

Comentários Redundantes

Em ambientes de aprendizagem, é comum que os alunos adicionem comentários redundantes para ajudar a entender o código.

Pra desenvolvedores mais experientes, não é necessário comentar o código, pois ele deve ser autoexplicativo.

Comentários Enganosos

Poucas coisas são mais destastosas do que comentários que documentam funcionalidades do código que não existem ou que não funcionam.

Um comentário enganoso deve ser alterado ou removido assim que possível.

```
-- BOA PRÁTICA #1: COMENTÁRIOS

-- CÓDIGO PARA ALTERAR VALORES DENTRO DO BANCO DE DADOS
-- Autor: Marcus Cavalcanti e João Lira
-- Data Criação: 11/04/2022
-- Última Atualização: 30/08/2022
-- Descrição: ....

-- Comentário #1: Início do Código
-- Dois Parâmetros: Id que significa o id do valor, e novo_valor será o novo valor atribuído
CREATE OR ALTER PROCEDURE altera_valor(@id INT, @novo_valor FLOAT)
AS
BEGIN
    -- Comentário #2: Início da Transação de Atualização (Update)
    BEGIN TRANSACTION

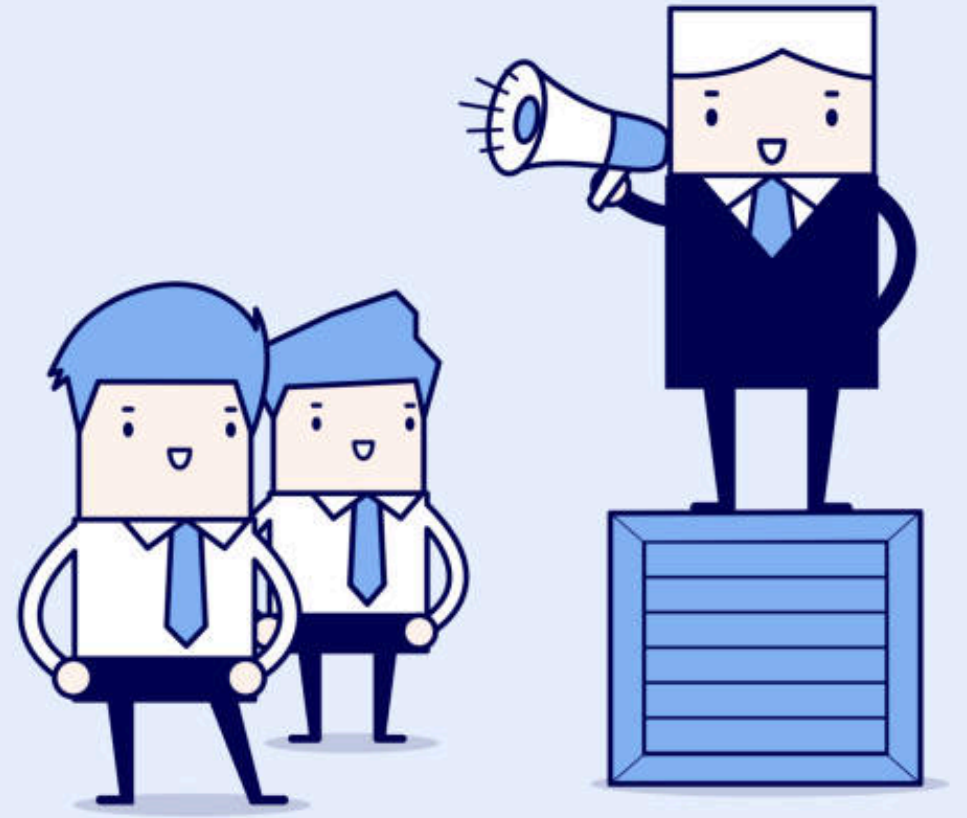
    -- A última alteração no código foi feita nessa linha
    UPDATE Carro
    SET valor = @novo_valor
    WHERE id_carro = @id

    -- Comentário #3: Efetivando a alteração no banco através de um COMMIT
    COMMIT
```

Comentários Imperativos

Com exceção de documentação de API, comentários não devem ser obrigatórios.

O uso de comentários deve ser moderado pelo bom senso do programador.



Comentários Longos

Comentários devem ser objetivos e diretos.

Um bloco de comentário que tende a crescer com o tempo é um forte indicativo de que ele não é bem vindo em um código.

Exemplo:

```
/*  
Esta variável name é usada para armazenar o nome do ambiente.  
O ambiente deve ser configurado da seguinte forma:  
- development  
- test  
- production  
Para alterar o ambiente, é necessário ir até as configurações de ambiente do sistema  
e alterar o valor da variável de ambiente.  
O valor padrão é development.  
Em caso de problemas em alterar o valor da variável de ambiente, é necessário reiniciar o sistema.  
*/  
String name = getEnviromentVariable("name");  
String statusCode = getEnviromentVariable("status_code");  
String statusMessage = getEnviromentVariable("status_message");
```

Comentários Ruidosos

Comentários excessivos adicionam ruído ao código, que impedem de entender funcionalidades ou nomenclaturas simples.

```
name = "Pankaj" # employee name
id = 100 # employee id

# This function adds the two numbers
def add(x, y):
    return x + y
```

Substitua comentários por funções

Quando existe a necessidade de comentar um trecho de código, a primeira ação é avaliar a possibilidade de deixar a função mais clara ou construir uma nova função.

Comentários que explicam o que acontece em um trecho de código são, geralmente, reflexos de um código ruim.

```
-- BOA PRÁTICA #1: COMENTÁRIOS

-- CÓDIGO PARA ALTERAR VALORES DENTRO DO BANCO DE DADOS
-- Autor: Marcus Cavalcanti e João Lira
-- Data Criação: 11/04/2022
-- Última Atualização: 30/08/2022
-- Descrição: ....

-- Comentário #1: Início do Código
-- Dois Parâmetros: Id que significa o id do valor, e novo_valor será o novo valor atribuído
CREATE OR ALTER PROCEDURE altera_valor(@id INT, @novo_valor FLOAT)
AS
BEGIN
    -- Comentário #2: Início da Transação de Atualização (Update)
    BEGIN TRANSACTION

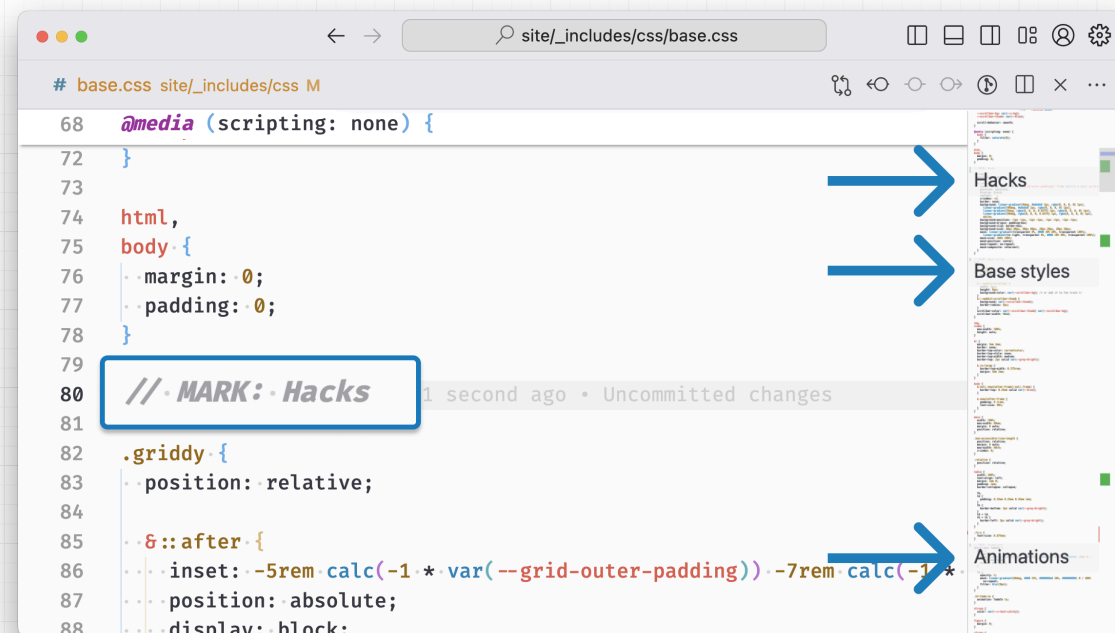
    -- A última alteração no código foi feita nessa linha
    UPDATE Carro
    SET valor = @novo_valor
    WHERE id_carro = @id

    -- Comentário #3: Efetivando a alteração no banco através de um COMMIT
    COMMIT
```

Marcadores

Não use comentários para marcar trechos de código em seções.

Se um código é muito longo, é provável que ele possa ser dividido em arquivos menores.



Exemplo:

```
public class Foo {  
    // ----- ATRIBUTOS -----  
    private String name;  
    // ----- CONSTRUTOR -----  
    public Foo(String name) {  
        this.name = name;  
    }  
    // ----- GETTERS -----  
    public String getName() {  
        return name;  
    }  
    // ----- SETTERS -----  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

Posição dos Comentários

Um bom comentário deve estar posicionado logo acima do código que ele descreve.

Em alguns casos, podemos ter comentários posicionados no final do código, mas devemos tomar cuidado com a legibilidade do código.

```
public String convertDatetimeToString(Date date) {  
    SimpleDateFormat sdf = new SimpleDateFormat();  
    return sdf.format(date);  
} // O padrão do SimpleDateFormat é "dd/MM/yyyy"
```


Comentários vs Controle de Versão

- Autoria
- Data de Criação / Atualização
- Linhas editadas

```
-- BOA PRÁTICA #1: COMENTÁRIOS

-- CÓDIGO PARA ALTERAR VALORES DENTRO DO BANCO DE DADOS
-- Autor: Marcus Cavalcanti e João Lira
-- Data Criação: 11/04/2022
-- Última Atualização: 30/08/2022
-- Descrição: ....

-- Comentário #1: Início do Código
-- Dois Parâmetros: Id que significa o id do valor, e novo_valor será o novo valor atribuído
CREATE OR ALTER PROCEDURE altera_valor(@id INT, @novo_valor FLOAT)
AS
BEGIN
    -- Comentário #2: Início da Transação de Atualização (Update)
    BEGIN TRANSACTION

    -- A última alteração no código foi feita nessa linha
    UPDATE Carro
    SET valor = @novo_valor
    WHERE id_carro = @id

    -- Comentário #3: Efetivando a alteração no banco através de um COMMIT
    COMMIT
```

Exemplo:

```
// Autor: John Doe  
// Data: 2023-07-28  
// Linhas editadas: 1-5, 10-15  
public class Foo {  
    // ...  
    // <- Última linha editada  
}
```

Material de Apoio

- [Daniel Wisky](#)
- [Codex](#)