



WBA0883_v1.0

Qualidade de software com *Clean Code* e técnicas de usabilidade



Integrando *Clean Code* e técnicas de usabilidade para ampliar a qualidade

Arquitetura de Clean Code

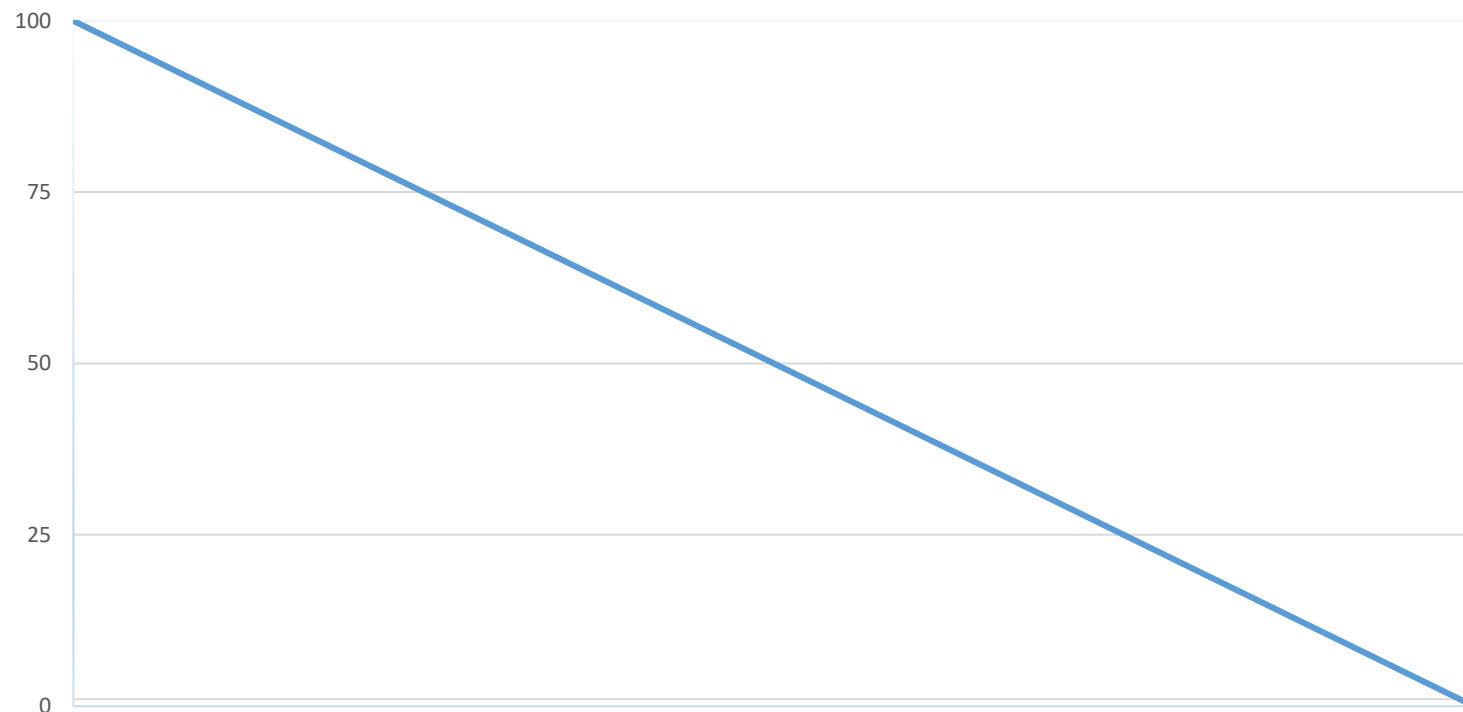
Técnicas de Clean Code

Bloco 1

Stella Marys Dornelas Lamounier



Figura 1 - Custo de um código confuso



Fonte: adaptada de Neves (2019).



➤ 3 Rs da arquitetura de software – *Clean Code*

Figura 2 - Arquitetura *Clean Code*



Fonte: elaborada pela autora.



➤ Indentação em *Clean Code*

Código sem Indentação

```
var SearchPage = function() {  
  this.searchField = $('#search');  
  this.imLuckButton = $('#search-button');  
  this.get = function() {  
    browser.get('search');  
  };  
  this.search = function(text) {  
    this.searchField.sendKeys(text);  
    this.imLuckButton.click();  
  };  
};  
module.exports = new SearchPage();
```

Código com indentação

```
var SearchPage = function() {  
  
  this.searchField = $('#search');  
  this.imLuckButton =  
    $('#search-button');  
  
  this.get = function() {  
    browser.get('search');  
  };  
  
  this.search = function(text) {  
    this.searchField.sendKeys(text);  
    this.imLuckButton.click();  
  };  
};  
module.exports = new SearchPage();
```

Formatação em *Clean Code*

Código ruim

```
const DIAS_DA_SEMANA = 7;
const diasdomes = 30;

const musicas = ['Maior
Abandonado', 'Sutilmente', 'Los
Hermanos'];
const Artists = ['Barao
Vermelho', 'Skank', 'Último
Romance'];

function eraseDatabase() {}
function restore_database() {}

class animal {}
class Cachorro {}
```

Código bom

```
const DIAS_DA_SEMANA = 7;
const DIAS_DO_MES = 30;

const MUSICAS = ['Maior
Abandonado', 'Sutilmente',
'Los Hermanos'];
const ARTISTAS = ['Barao
Vermelho', 'Skank', 'Último
Romance'];

function eraseDatabase() {}
function
restore_Database(){}

class Animal {}
class Cachorro {}
```


Como é feita a leitura inicial de um código-fonte?

A leitura de um código deve ser tão simples quanto a leitura de um jornal.

Figura 3 - Principais jornais – técnicas de leitura



Fonte: <https://diariodocomercio.com.br/politica/comissao-mista-rejeita-mp-da-publicacao-de-balanco-nos-jornais-impresos>. Acesso em: 28 out. 2021.



Formatação vertical

- Blocos de códigos devem ter em média 200 linhas de código.
- Utilizar o espaço para separação de conceitos.
- Declaração de variáveis sempre próxima de onde será utilizada.
- Quando dependentes, as funções devem ficar verticalmente próximas uma das outras.





Formatação horizontal

- Linhas entre 100 até 200 caracteres.
- Separação do código por espaços em branco.
- Não se deve separar com espaços em branco funções com parênteses.
- As variáveis devem ser seguidas umas das outras.

```
int TamanhoNome = nome.length ( ) ;  
totalCaracteres += TamanhoNome;
```





Funções

- As funções devem fazer apenas uma coisa.
- Elas devem realizar apenas uma ação.

Código ruim

```
function emailClients(clients) {  
  clients.forEach((client) => {  
    const clientRecord =  
      database.lookup(client);  
    if (clientRecord.isActive()) {  
      email(client);  
    }  
  });  
}
```

Código bom

```
function  
emailActiveClients(clients) {  
  clients  
    .filter(isActiveClient)  
    .forEach(email);  
}  
function  
isActiveClient(client) {  
  const clientRecord =  
    database.lookup(client);  
  return  
    clientRecord.isActive();  
}
```

➤ Tratamento de exceções em *Clean Code*

- *Try* é responsável pela transações que podem ser canceladas a qualquer momento.
- O *catch* é responsável pela definição de escopo na execução.

Código ruim

```
try {  
    FuncaoDeExcecao();  
} catch (error) {  
    console.log(error);  
}
```

Código bom

```
try {  
    FuncaoDeExcecao();  
} catch (error) {  
    // Uma opção (mais chamativa que  
    console.log):  
    console.error(error);  
    // Outra opção:  
    NotificarUsuario(error);  
    // Outra opção:  
    ReportarParaOServico(error);  
    // OU as três!  
}
```

➤ Separação e construção de sistemas

- Efetuar todos os testes.
- Sem duplicação de código.
- Expressar o propósito do programador.
- Minimizar o número de classes e métodos.

Código ruim

```
public int Size()
{
    // Size implementation
}
public bool IsEmpty()
{
    // IsEmpty implementation
}
}
```

Código bom

```
public bool IsEmpty()
{
    return Size() == 0;
}
```



Integrando *Clean Code* e técnicas de usabilidade para ampliar a qualidade

Clean Architecture

Bloco 2

Stella Marys Dornelas Lamouonier





Clean Architecture

- Utiliza o conceito de “Regra de Independência”.
- A dependência do código-fonte só pode apontar para o interior do aplicativo.
- Deve ser independente de *frameworks*.
- Facilmente testável.
- Independente do banco de dados.
- Independente da interface gráfica.



➤ *Clean Architecture* - Regra de Dependência

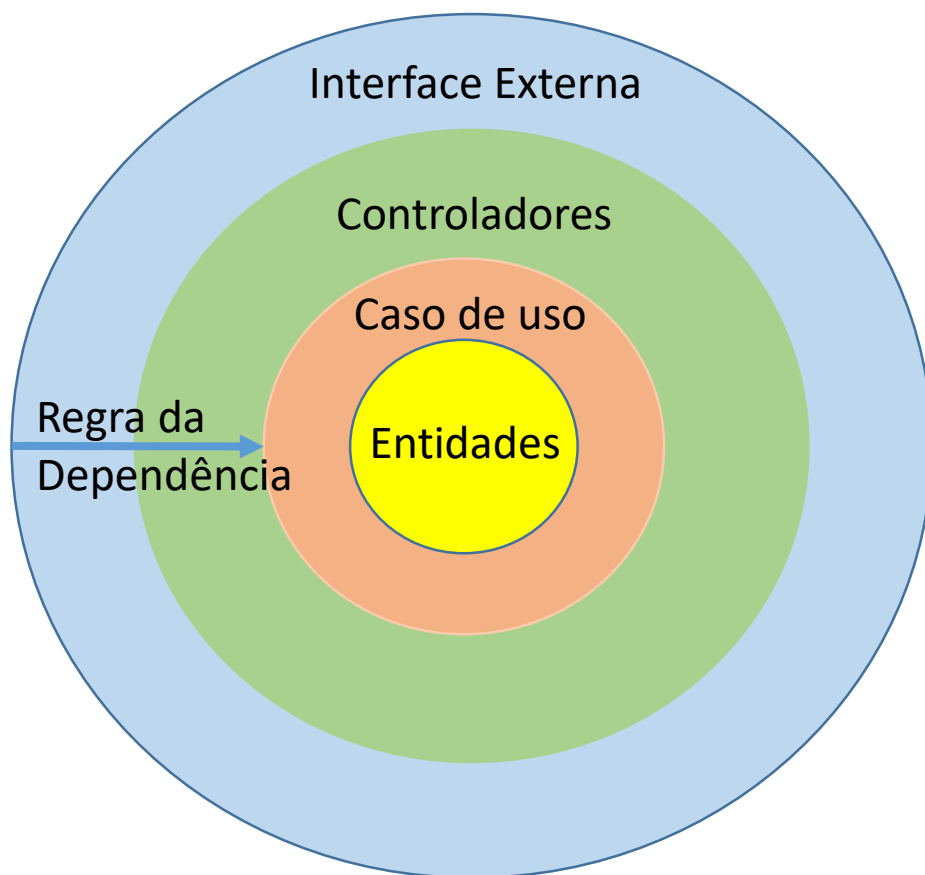
Em uma Arquitetura Limpa, as classes de uma camada não devem conhecer nenhuma classe de uma camada mais externa, ou seja, a dependência do código-fonte só pode apontar para o interior do aplicativo.









Clean Architecture

Figura 4 - Arquitetura Limpa



-  Regras de negócio empresariais.
-  Regras de negócio do projeto.
-  Adaptadores de interface.
-  *Frameworks* e drives.

Fonte: adaptada de Martin (2012).



Clean Architecture - Aplicação

Figura 5 - Aplicação *Clean Architecture*

```
// Casos de uso:

public class AdicionarProdutoNoCarrinhoPeloCliente
{
    public AdicionarProdutoNoCarrinhoPeloCliente(Produtos produto) {}
}

public class AdicionarProdutoAoInventarioPeloAdmin
{
    public AdicionarProdutoAoInventarioPeloAdmin(Produtos produto) {}
}

// Entidades:

public class Clientes()
{
    public int ClientId { get; set; }
    public string ClienteNome { get; set; }
}

public class Produtos()
{
    public int ProdutoId { get; set; }
    public string ProdutoNome { get; set; }
}

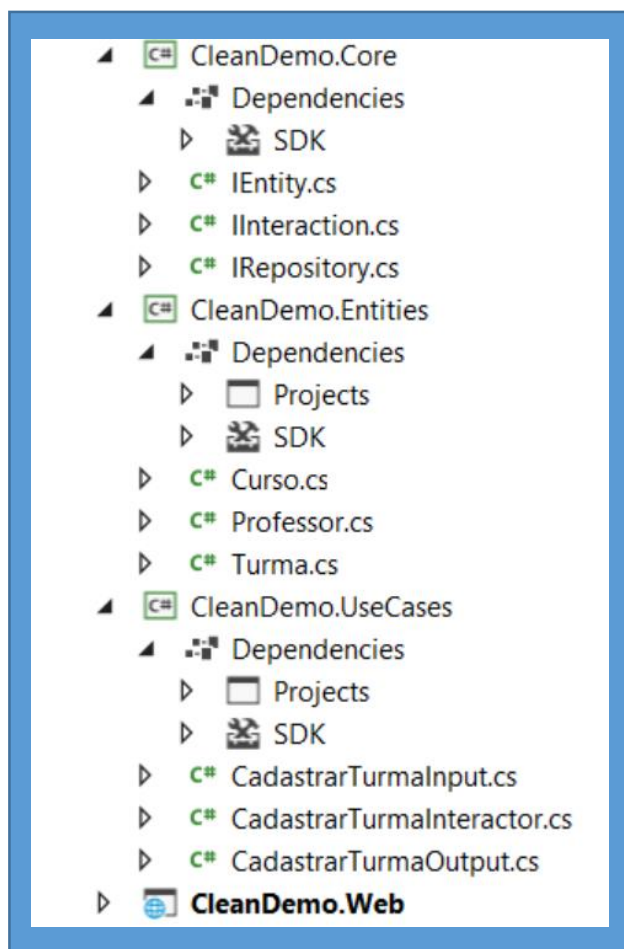
public class Pedidos()
{
    public int PedidoId { get; set; }
    public int ClientId { get; set; }
    public DateTime DataCriacao { get; set; }
}
```

Fonte: Macoratti (2021, [s.p.]).



Clean Architecture - Aplicação

Figura 6 - Estrutura *Clean Architecture* – Sistema Escolar



Fonte: Fonseca (2021, [s.p.]).

Integrando *Clean Code* e técnicas de usabilidade para ampliar a qualidade

Técnicas e Testes automatizados de Usabilidade

Bloco 3

Stella Marys Dornelas Lamouonier



► Testes Automatizados de Unidade

Figura 7 - Logomarca Loop11



Fonte: <https://www.loop11.com>.
Acesso em: 22 jul. 2012.

Figura 8 - Fivesecondtest



Fonte: <https://www.psdmockups.com/original-five-second-test/>. Acesso em: 22 jul. 2012.



➤ Testes Automatizados de Unidade

Projeto Prático

Criar um teste automatizado de Interface utilizando a ferramenta Fivesecondtest para criar testes de 5 segundos por meio da utilização de um protótipo.



► Testes Automatizados de Unidade

Projeto Prático

Criar um teste automatizado de Usabilidade utilizando a ferramenta Loop 11 para criação de 5 tarefas de testes de Usabilidade.



Teoria em Prática

Bloco 4

Stella Marys Dornelas Lamounier



➤ Reflita sobre a seguinte situação

O *Clean Code* foi criado para sanar, principalmente, problemas com manutenção de código-fonte, ou seja, códigos mal escritos podem até funcionar, mas com o passar do tempo seu prejuízo é evidente.

É verdade que profissionais passam muito mais tempo tentando entender códigos já existentes do que escrevendo novos códigos.

Um bom código-fonte deve ter princípios, objetivos e clareza, e esta é a função do *Clean Code*.

Portanto, lembre-se: um sistema jamais está finalizado, alguma hora ele vai precisar de uma nova funcionalidade ou manutenção.



➤ Reflita sobre a seguinte situação

Como profissional da área, você aborda os demais colegas da empresa de desenvolvimento em que trabalha e utiliza essas técnicas para que o código seja facilmente interpretado por todos os membros da equipe e demais profissionais que futuramente utilizarão o código, seja para leitura quanto para alterações. Quais as técnicas de *Clean Code* que você mostraria aos colegas?



► Norte para a resolução...

Algumas boas práticas podem ser aplicadas de maneira simples nas empresas de desenvolvimento sem alterar toda sua metodologia, como:

- Na criação de nomes.
- Comentários.
- Criação de funções e classes.
- Criação de métodos.
- Refatoração.
- Testável.

Aplicando boas práticas nas funcionalidades mencionadas, ao final, quem ganha é a empresa que cria sistemas mais fáceis de alterar e com pouca manutenção.



Dica do(a) Professor(a)

Bloco 5

Stella Marys Dornelas Lamounier





Dicas de Ferramentas

- Testes automatizados de usabilidade devem fazer parte das atividades de testes de empresas de desenvolvimento.
- Existem no mercado inúmeras ferramentas que podem auxiliar os testadores com eficácia e eficiência em seus projetos de softwares, como:
 - *CrazyEgg.*
 - *Card Sorting.*
 - *ScreenCast O' Matic.*
 - *Robot Replay.*





Referências

DIÁRIO DO COMÉRCIO. Comissão mista rejeita MP da publicação de balanço nos jornais impresso. 2019. Disponível em:

<https://diariodocomercio.com.br/politica/comissao-mista-rejeita-mp-da-publicacao-de-balanco-nos-jornais-impressos/>. Acesso em: 7 jul. 2021.

FONSECA, E. Introdução a Clean Architecture. **Imasters**, 8 out. 2018. Disponível em: <https://imasters.com.br/back-end/introducao-clean-architecture>. Acesso em: 7 jul. 2021.

MARTIN. R. C. The Clean Architecture. **The Clean Code Blog**, 2012. Disponível em: <https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html>. Acesso em: 6 jul. 2021.

MACORATTI. ASP .NET Core - Clean Architecture. 2010. Disponível em: http://www.macoratti.net/20/10/aspc_cleanarq1.htm. Acesso em: 7 jul. 2021.

NEVES, P. Y. Clean Code - Nomes significativos. **Tech Azi**, 2019. Disponível em: <http://tech.azi.com.br/clean-code-parte-1-nomes-significativos/>. Acesso em: 7 jul. 2021.

Bons estudos!

