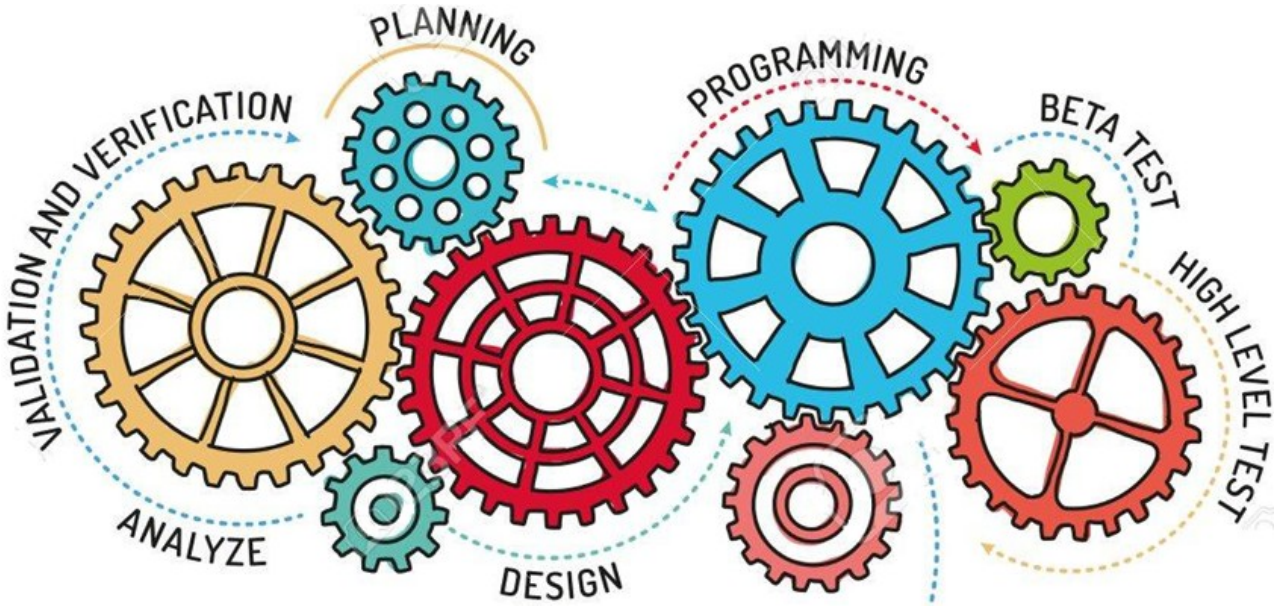
Engenharia de Software

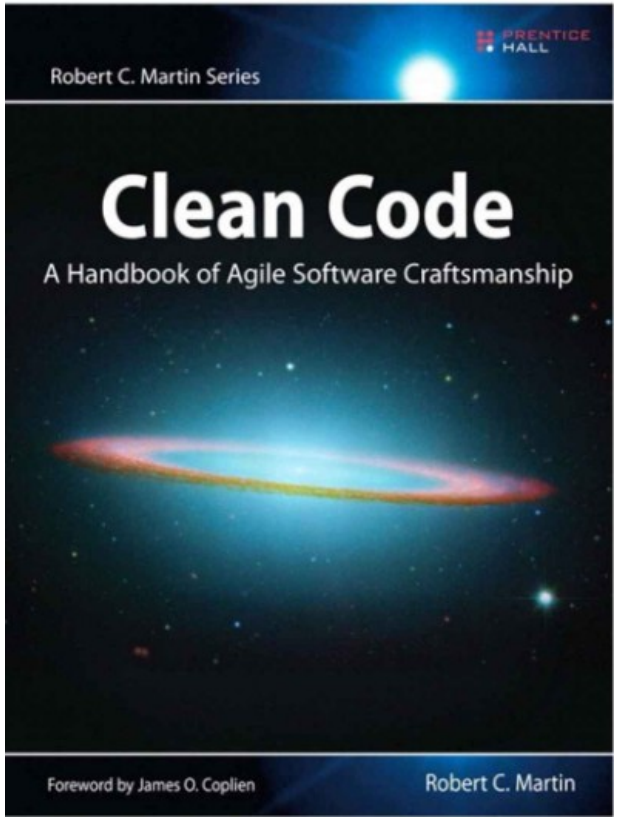


Prof. Me. Clênio Silva
e-mail: clenio.silva@uniube.br

Uniube

Boas práticas no Desenvolvimento de projetos de software

- **Clean code**
- **Clean Architecture**



Boas práticas no Desenvolvimento de projetos de software

- Ter como principio a escrita de código limpo é muito importante para a continuidade do projeto;
- Outros desenvolvedores terão facilidade no entendimento do código;
- Um código bem desacoplado, bem nomeado e padronizado, tende-se a ser mais fácil de implementar novas *features* e também a resolução de *bugs*;
- Lembre-se, um código ruim pode funcionar. Mas se ele não for limpo, pode acabar com uma empresa de desenvolvimento.

10 Boas práticas de Clean Code

1) Funções devem fazer apenas uma coisa:

   a) Um código dividido em pequenas funções é mais agradável aos olhos, mais simples de ler. O nome dessas funções deve dizer exatamente o que fazem.

2) Evite efeitos colaterais:

   a) Um efeito colateral é quando uma função não faz o que diz. Altera coisas fora do escopo dela como variáveis globais ou muda dados que não fazem sentido com o nome da função

3) Use nomes significativos:

   a) É muito importante dar bons nomes a variáveis, funções, classes e tudo o que há no código. Isso separa um código que se explica de um código nebuloso.

10 Boas práticas de Clean Code

4) Evite comentários:

   a) O código deve explicar a si mesmo, comentários geralmente compensam um código sujo.

5) Remova código inutilizado:

   a) Trechos comentados ou funções que não são mais chamadas só poluem a base de código.

6) Selecione uma palavra por conceito:

   a) É confuso ter funções como *fetch*, *retrieve*, *get* para o mesmo propósito em partes diferentes do código. O ideal é escolher uma e padronizar todo o código assim.

10 Boas práticas de Clean Code

7) Evite repetição:

   a) Código repetido se amontoa, cria a necessidade de modificar a mesma coisa em vários lugares em caso de mudança, e cria novas oportunidades para omissão de erros. ***Don't Repeat Yourself***.

8) Evite funções com mais de 2 parâmetros:

   a) Quando uma função recebe 3 ou mais parâmetros é hora de considerar criar um objeto com sua própria classe/tipo para passar esses parâmetros.

9) Evite mapeamento mental:

   a) Relaciona a 3º prática descrita. Use nomes significativos. Um programador nomeando uma variável como u sabe que ela é referente a um usuário. Mas outro desenvolvedor que pegar o código para dar manutenção terá dificuldade.

10 Boas práticas de Clean Code

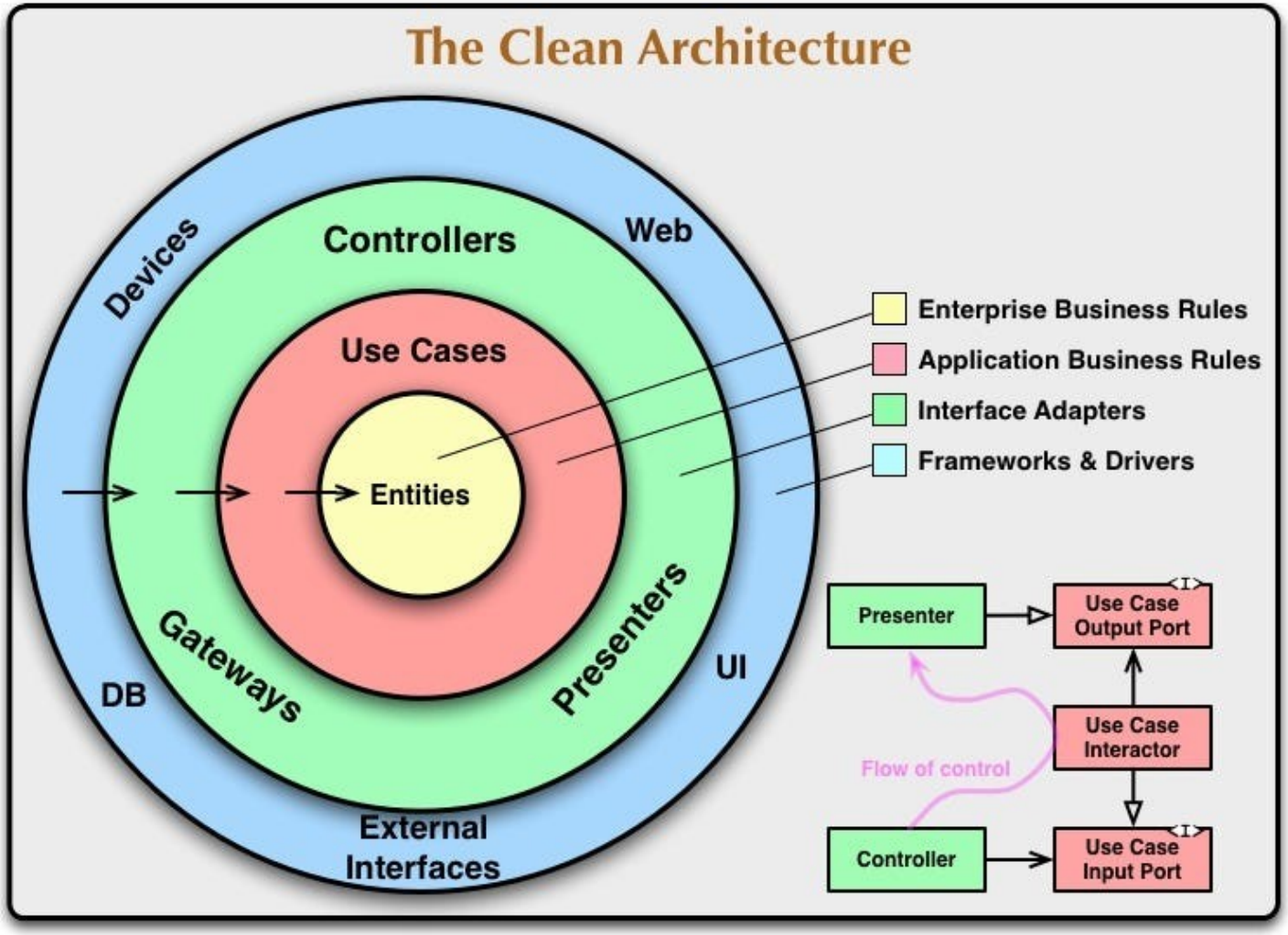
10) Espaçamento vertical entre conceitos:

a) Cada linha ou grupo de linhas representa um pensamento completo. Esses pensamentos devem ficar separados por linhas em branco.

Clean Architecture

- Clean Architecture tenta fornecer uma metodologia a ser usada na codificação, a fim de facilitar o desenvolvimento códigos, permitir uma melhor manutenção, atualização e menos dependências.
- Um objetivo importante da Clean Architecture é fornecer aos desenvolvedores uma maneira de organizar o código de forma que encapsule a lógica de negócios, mas mantenha-o separado do mecanismo de entrega.

Clean Architecture



Clean Architecture

As vantagens de utilizar uma arquitetura em camadas são muitas, porém podemos pontuar algumas:

- Testável. As regras de negócios podem ser testadas sem a interface do usuário, banco de dados, servidor ou qualquer outro elemento externo.
- Independente da interface do usuário. A interface do usuário pode mudar facilmente, sem alterar o restante do sistema. Uma UI da Web pode ser substituída por uma UI do console, por exemplo, sem alterar as regras de negócios.
- Independente de banco de dados. Você pode trocar o Oracle ou SQL Server, por Mongo, BigTable, CouchDB ou qualquer outro. Suas regras de negócios não estão vinculadas ao banco de dados.
- Independente de qualquer agente externo. Na verdade, suas regras de negócios simplesmente não sabem nada sobre o mundo exterior, não estão ligadas a nenhum Framework

Clean Architecture

A separação de camadas poupará o desenvolvedor de muitos problemas futuros com a manutenção do software, a regra de dependência bem aplicada deixará seu sistema completamente testável. Quando um framework, um banco de dados, ou uma API se tornar obsoleta a substituição de uma camada não será uma dor de cabeça, além de garantir a integridade do core do projeto.

Clean Architecture