Certified Tech Developer
The Ultimate Degree

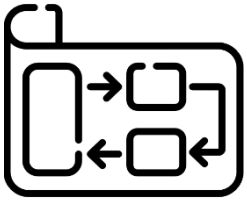
Padrões de Projeto

Os padrões de projeto são soluções comprovadas e documentadas para problemas comuns de desenvolvimento de software. Eles também são usados na automação de software.

Padrões de projeto em automatização

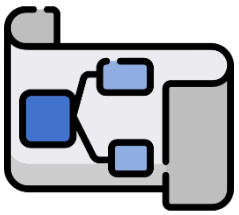
Screenplay:

- Este padrão tem uma abordagem de desenvolvimento orientada pelo comportamento (*Behaviour Driven Development* - BDD). É uma estratégia de desenvolvimento que se concentra na prevenção de defeitos ao invés de localizá-los em um ambiente controlado.
- Apresenta um alto desacoplamento da interface do usuário.
- Propõe a trabalhar em termos de tarefas e não de interface de usuário.
- Cumpre os princípios SOLID.

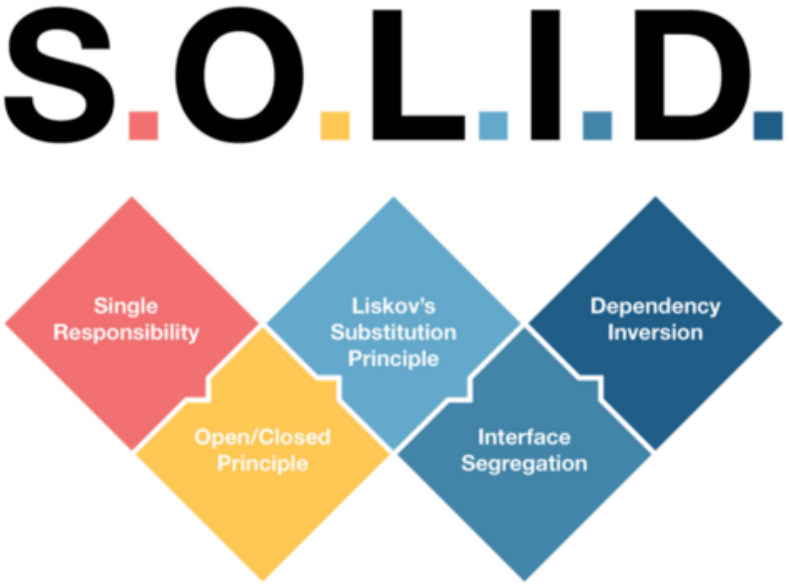


Page Object:

- É um padrão de projeto que representa os componentes web ou a página web em uma classe.
- É utilizado em testes automatizados para evitar código duplicado e melhorar a manutenção dos mesmos.
- Não cumpre com os princípios SOLID.



O que são os princípios SOLID?



SOLID é um acrônimo introduzido por Robert C. Martin para definir os cinco princípios básicos da programação orientada a objetos: *Single responsibility*, *Open-closed*, *Liskovs substitution*, *Interface segregation* e *Dependency inversion*. SOLID tem bastante relação com os padrões de projeto.

S: Princípio da responsabilidade única

Como o próprio nome indica, estabelece que uma classe, componente ou microsserviço deve ser responsável apenas por uma coisa (o tão aclamado termo *decoupled* em inglês). Se, por outro lado, uma classe tem várias responsabilidades, isso implica que uma mudança em uma responsabilidade causará uma mudança em outra.

O: Princípio do aberto/fechado

Estabelece que as entidades de software (classes, módulos e funções) devem ser abertas para sua extensão, mas fechadas para sua modificação.

L: Princípio de substituição de Liskov

Estabelece que uma subclasse deve ser substituível por sua superclasse. Se ao fazer isso o programa falhar, estaremos violando este princípio. Obedecer a este princípio confirmará que nosso programa tem uma hierarquia de classes fácil de entender e um código reutilizável.

I: Princípio de segregação de interfaces

Este princípio estabelece que os clientes não devem ser forçados a depender de interfaces que não usam. Em outras palavras, quando um cliente depende de uma classe que implementa uma interface cuja funcionalidade ele não usa, mas que outros clientes usam, ele será afetado pelas mudanças que outros forçam nessa interface.

D: Princípio de inversão de dependências

Estabelece que as dependências devem estar nas abstrações, não nas concreções. Quer dizer:

- Os módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações.
- As abstrações não devem depender de detalhes. Os detalhes devem depender de abstrações.

Em algum momento, nosso programa ou aplicativo será composto por vários módulos. Quando isso acontecer, devemos usar injeção de dependência: ela nos permitirá controlar as funcionalidades de um local específico ao invés de tê-las espalhadas por todo o programa.

Os princípios SOLID são guias, boas práticas, que podem ser aplicadas no desenvolvimento de software para eliminar projetos ruins. Eles fazem com que o programador tenha que refatorar o código-fonte até que seja legível e extensível. Em outras palavras, eles podem ajudar a escrever um código melhor: mais limpo, mais fácil de manter e escalável.

Alguns dos objetivos para considerar esses princípios para o código são:

- Criar um software eficaz que faça o trabalho e seja robusto e estável.
- Escrever um código limpo e flexível diante das mudanças: que pode ser facilmente modificado de acordo com a necessidade, que seja reutilizável e fácil de manter.
- Permitir escalabilidade: que aceita ser expandido com novas funcionalidades de forma ágil.

Resumindo, desenvolver um **software de qualidade.**