ARQUITETURA DE SOFTWARE

João Choma Neto
joao.choma@unicesumar.edu.br
https://github.com/JoaoChoma/arquitetura-software
Unicesumar – Maringá

Teste de Aceitação

TESTE DE ACEITAÇÃO

- O teste de aceitação, também conhecido como teste de aceitação do usuário, se concentra na validação do software pelos usuários finais ou pelos stakeholders que representam os interesses dos usuários.

TESTE DE ACEITAÇÃO

- Usuário não é o cliente

- Um usuário de sistema é uma pessoa ou entidade que interage com um sistema de software ou computador para realizar tarefas, executar funções ou obter informações.
- Os usuários de sistemas podem variar amplamente em suas funções, necessidades e níveis de acesso, dependendo do tipo de sistema e do contexto em que estão operando.

TESTE DE ACEITAÇÃO

- Usuário não é o cliente

- CLIENTE refere-se à pessoa, equipe, departamento, organização ou entidade que solicita, encomenda ou financia o desenvolvimento de um sistema de software.
- Cliente são os responsáveis por definir os requisitos, objetivos e necessidades do sistema.

TESTE DE ACEITAÇÃO DO USUÁRIO

- Usuário não é o cliente

- **Usuário Final**: Os usuários de sistema podem ser classificados como "usuários finais", o que significa que são os destinatários finais do software ou do sistema.
- Por exemplo, em um software de processamento de texto, os usuários finais seriam escritores e editores que usam o software para criar documentos.

TESTE DE ACEITAÇÃO DO USUÁRIO

- Usuário não é o cliente

- **Usuários Internos e Externos**: Os usuários de sistema podem incluir pessoas dentro da organização (usuários internos) ou fora dela (usuários externos).
- Por exemplo, os funcionários de uma empresa podem ser usuários internos de um sistema interno, enquanto os clientes que acessam um site de comércio eletrônico são usuários externos.

TESTE DE ACEITAÇÃO DO USUÁRIO

- Usuário não é o cliente

- **Níveis de Acesso**: Os usuários de sistema podem ter diferentes níveis de acesso e permissões, dependendo de sua função e responsabilidades.
- Os usuários podem ter acesso completo a todas as funcionalidades do sistema, ou ter acesso limitado a recursos específicos.

TESTE DE ACEITAÇÃO DO CLIENTE

- Representantes dos clientes ou stakeholders avaliam o software com base nos requisitos do projeto e nas metas de negócios.

COMO FAZER?

PARTICIPANTES

- Os participantes do teste de aceitação são geralmente os usuários finais ou representantes dos usuários. Eles são as pessoas que usarão o software no ambiente real.

ESCOPO

- O teste de aceitação aborda principalmente os requisitos de negócios e as necessidades dos usuários finais. Ele se concentra em verificar se o software atende a esses requisitos de maneira satisfatória.

AMBIENTE DE TESTE

- O teste de aceitação é realizado em um ambiente que se assemelha ao ambiente de produção
- A ideia é garantir que o software funcione da mesma forma que quando estiver em uso real.

Teste de Usuário

TESTE DE USUÁRIO

- O teste de usuário, também conhecido como teste de usuário final ou teste de usuário real, é uma fase no processo de desenvolvimento de software, na qual os usuários finais ou representantes dos usuários participam da avaliação do software antes do lançamento

TESTE DE USUÁRIO

- O principal objetivo do teste de usuário é coletar feedback direto dos usuários reais para garantir que o software atenda às suas necessidades, expectativas e requisitos.

COMO FAZER?

PARTICIPANTES

- Os participantes do teste de usuário são os usuários finais reais para os quais o software está sendo desenvolvido, ou representantes desses usuários.
- Podem incluir clientes, funcionários da empresa, membros da comunidade, ou qualquer pessoa que represente o público-alvo do software.

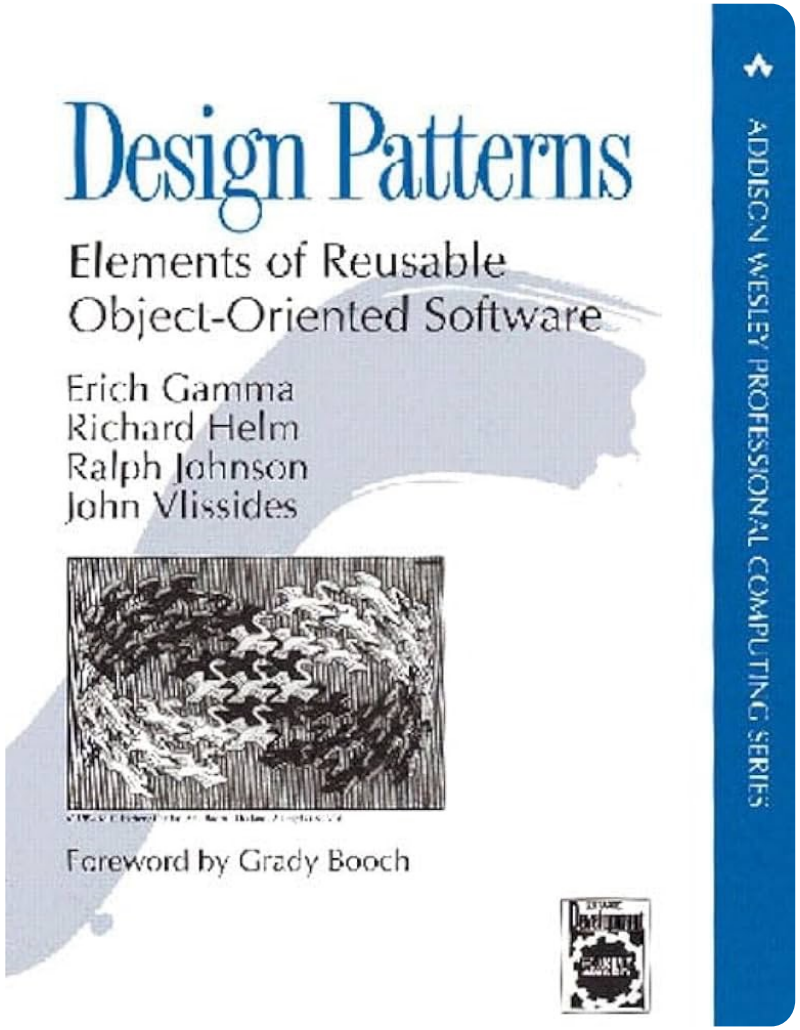
AMBIENTE DE TESTE

- Os participantes são solicitados a realizar tarefas específicas usando o software, seguindo cenários de uso definidos.
- Isso ajuda a simular situações da vida real em que o software será usado.

FEEDBACK

- Os participantes são encorajados a fornecer feedback qualitativo sobre sua experiência. Isso pode incluir comentários, sugestões e observações sobre quaisquer problemas encontrados.

PADRÃO DE PROJETO



CATÁLOGO

- Referência ao livro "Design Patterns: Elements of Reusable Object-Oriented Software", escrito por Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides

CATÁLOGO

- Este livro, muitas vezes abreviado como o livro "GoF", foi publicado em 1994 e é considerado uma referência fundamental no campo da engenharia de software.

CATÁLOGO

- Os padrões de projeto são comumente agrupados em três categorias principais:
  - Criacionais
  - Estruturais
  - Comportamentais

PADRÕES CRIACIONAIS

- Os padrões criacionais estão relacionados à criação de objetos de maneira adequada para a situação em que se encontram
- Mecanismos de criação de objetos, tentando criar objetos de uma maneira que seja adequada às circunstâncias

PADRÕES ESTRUTURAIS

- Os padrões estruturais se preocupam com a composição de classes e objetos
- Como as classes e objetos são compostos para formar estruturas maiores
- Soluções fornecem maneiras de simplificar as estruturas

PADRÕES COMPORTAMENTAIS

- Os padrões comportamentais estão preocupados com a comunicação entre objetos
- Como os objetos colaboram entre si para alcançar funcionalidades específicas

SINGLETON

- O padrão Singleton é usado quando você precisa garantir que uma classe tenha apenas uma instância e fornece um ponto global de acesso a essa instância

JÁ VIMOS


Estratégia... Strategy... Stratégie
NASCIMENTO

STRATEGY

- STRATEGY é um padrão comportamental que permite definir:
  - uma família de algoritmos,
  - encapsular cada um deles como um objeto e
  - torná-los intercambiáveis.

INTERCAMBIÁVEL

- Um objeto intercambiável, no contexto de padrões de projeto como o Strategy, refere-se à capacidade de substituir um objeto por outro do mesmo tipo sem que o código cliente (que usa o objeto) precise ser alterado
- Essa substituição pode ocorrer em tempo de execução

OBJETIVO DO PADRÃO STRATEGY

- Permitir a seleção de um algoritmo em tempo de execução, dependendo do contexto em que é necessário.
- Encapsular comportamento que pode variar e separá-lo do código que permanece inalterado.

FUNCIONAMENTO

- Imagine que você está construindo um aplicativo de mapas e precisa de diferentes algoritmos de roteamento: um para **pedestres**, outro para **automóveis** e um terceiro para **bicicletas**

FUNCIONAMENTO

- Cada algoritmo pode ser encapsulado dentro de sua própria classe que implementa uma interface RoteamentoStrategy
- O contexto, que neste caso poderia ser o próprio aplicativo de mapas, mantém uma referência a um objeto RoteamentoStrategy que é utilizado para calcular o roteamento

FUNCIONAMENTO

- O usuário decide qual estratégia de roteamento deseja (por exemplo, selecionando um modo de transporte)
- O aplicativo ajusta a estratégia de roteamento conforme necessário durante a execução

STRATEGY - VANTAGENS

- **Flexibilidade:** Facilmente intercambiável em tempo de execução
- **Encapsulamento:** Separa a implementação de um algoritmo das classes que o utilizam
- **Extensibilidade:** Novos comportamentos podem ser adicionados sem alterar o contexto

STRATEGY - DESVANTAGENS

- **Complexidade:** Pode aumentar a complexidade do código devido à criação de múltiplas classes.
- **Quantidade de Objetos:** Pode aumentar o número de objetos no sistema.

ARQUITETURA DE SOFTWARE

João Choma Neto
joao.choma@unicesumar.edu.br
https://github.com/JoaoChoma/arquitetura-software
Unicesumar – Maringá