Disciplina Engenharia de Software I

Resolução da lista de exercícios no. 1 - Introdução à Engenharia de Software

Prof. Tiago Garcia de Senna Carneiro

ENYA LUÍSA GOMES DOS SANTOS 19.2.4201

1.

Engenharia de software é a área que busca sistematizar o desenvolvimento de software, com um conjunto de princípios, métodos e técnicas, com a finalidade de garantir uma boa qualidade, desempenho, facilidade de manutenção, baixo custo e produtividade de software.

2.

Segundo descreve o guia PMBOK (PMI, 2013), um projeto pode entendido como sendo um esforço temporário que é empreendido com o intuito de se criar um produto, serviço ou resultado exclusivo, tendo início e fim definidos, caracterizando assim a sua natureza temporária.

3.

A arquitetura de software diz o como os softwares são divididos em componentes e como esses componentes se comunicam.

4.

Componentes de software é uma parte de um software que funciona de forma independente e pode ser usada por outro softwares, se assemelha muito com uma biblioteca, ou seja, se uma pessoa cria um componente para executar uma tarefa, outro programador que queira executar a mesma tarefa em seu software pode reutilizar esse componente já criado adicionando-o em seu código. Componentes de software são focados em reutilização.

"Por exemplo, se um programador criar um componente para acessar um cliente em um banco de dados corporativo, nenhum outro programador terá que escrever tal funcionalidade novamente."

O desenvolvimento baseado em componentes permite que o sistema final seja tratado como vários “minissistemas”, diminuindo sua complexidade e

permitindo que cada componente empregado seja focado em apenas uma funcionalidade ou um conjunto de funcionalidades semelhantes.

5.

Monolítica: Toda a modularização utilizada é executada em uma mesma máquina. Assim, os módulos compartilham recursos de processamento, memória, bancos de dados e arquivos.

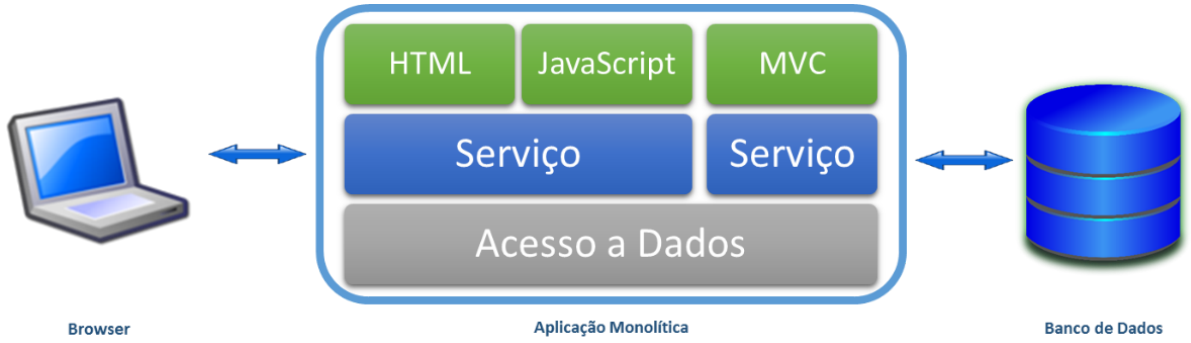


Figura - Representação da arquitetura monolítica.

FONTE: Micro Serviços: Qual a diferença para a arquitetura Monolítica? | OPUS

Desvantagens:

- **Manutenção:** a aplicação se torna cada vez maior de acordo com o seu tamanho, o código será cada vez mais difícil de entender e o desafio de fazer alterações rápidas e ter que subir para o servidor só cresce;
- **Alterações:** para cada alteração feita, é necessário realizar um novo *deploy* de toda a aplicação;
- **Linha de código**: uma linha de código que subiu errada pode quebrar todo o sistema e ele ficar totalmente inoperante;
- **Linguagens de programação:** não há flexibilidade em linguagens de programação. Aquela que for escolhida no início do projeto terá que ser seguida, sempre. Se o desenvolvimento de uma nova funcionalidade exigir outra linguagem de programação, existem duas possibilidades: ou todo o código é alterado ou a arquitetura do sistema precisará ser trocada.

Vantagens:

- **Mais simples de desenvolver:** a organização fica concentrada em um único sistema;

- **Simples de testar:** é possível testar a aplicação de ponta a ponta em um único lugar;
- **Simples de fazer o *deploy* para o servidor:** a alteração é simplesmente feita e pronto;
- **Simples de escalar:** como é só uma aplicação, se for preciso adicionar mais itens, é simplesmente ir adicionando o que for necessário.

Camadas: Os componentes de uma Layered Architecture são organizados em camadas horizontais onde cada uma desempenha um papel específico na aplicação.

Embora o padrão não especifique a quantidade e os tipos de camadas, a maioria segue a seguinte organização: apresentação, negócios, persistência e banco de dados. Em alguns casos, a camada de negócios e persistência são integradas quando a lógica de persistência é encapsulada nos componentes de negócio (exemplo com o uso de ORMs como NHibernate e Entity Framework).

Além disto, é comum que aplicações pequenas possuam em torno de 3 camadas e aplicações mais complexas em torno de 5 ou mais camadas.

Cada camada dentro da arquitetura possui um papel e uma responsabilidade dentro da aplicação. Por exemplo, a camada de apresentação lida com aspectos relacionados à interface do usuário, enquanto a camada de negócios lida com regras de negócios dentro de um determinado contexto.

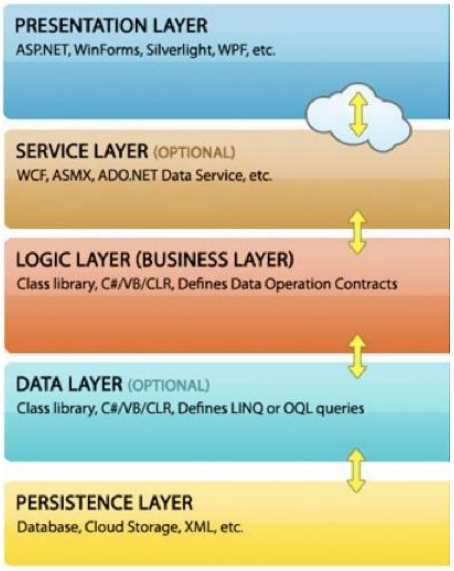


FIGURA - Exemplo de organização em camadas

FONTE: Arquitetura em Camadas - Padrões de Projeto

Portanto, cada camada fornece uma abstração sobre o trabalho necessário para atender uma necessidade de negócios. Consequentemente, cada camada não precisa saber os detalhes do que acontece nas camadas mais distantes na estrutura.

Em resumo, uma das principais características deste padrão de projeto está na separação de responsabilidades. Componentes dentro de uma camada lidam apenas com as particularidades daquela camada. E este modelo de organização facilita o processo de construção, testes e manutenção dos seus elementos.

Cliente-servidor: A tecnologia cliente/servidor é uma arquitetura na qual o processamento da informação é dividido em módulos ou processos distintos. Um processo é responsável pela manutenção da informação (servidores) e outros responsáveis pela obtenção dos dados (os clientes).

Os processos cliente enviam pedidos para o processo servidor, e este por sua vez processa e envia os resultados dos pedidos.

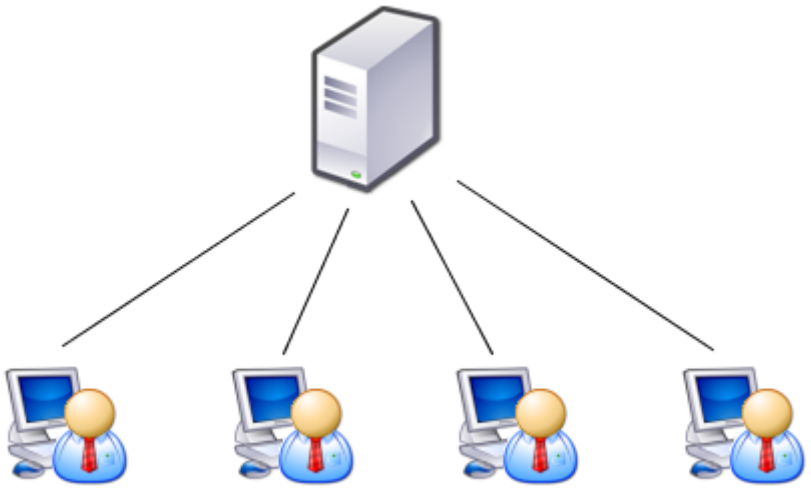


FIGURA - Representação da arquitetura cliente-servidor

FONTE: COMO FUNCIONA A ARQUITETURA CLIENTE SERVIDOR | ARQUITETURA CLIENTE/SERVIDOR

Vantagem: desse modelo é que os servidores podem ser distribuídos em rede. A funcionalidade geral (por exemplo, um serviço de impressão) pode estar disponível para todos os clientes e não precisa ser implementada por todos os serviços.

Desvantagens: cada serviço é um único ponto de falha e, portanto, é suscetível a ataques de negação de serviço ou a falhas no servidor. O desempenho pode ser imprevisível porque depende da rede e também do sistema. Podem surgir problemas de gerenciamento se os servidores forem de propriedade de organizações diferentes.

6.

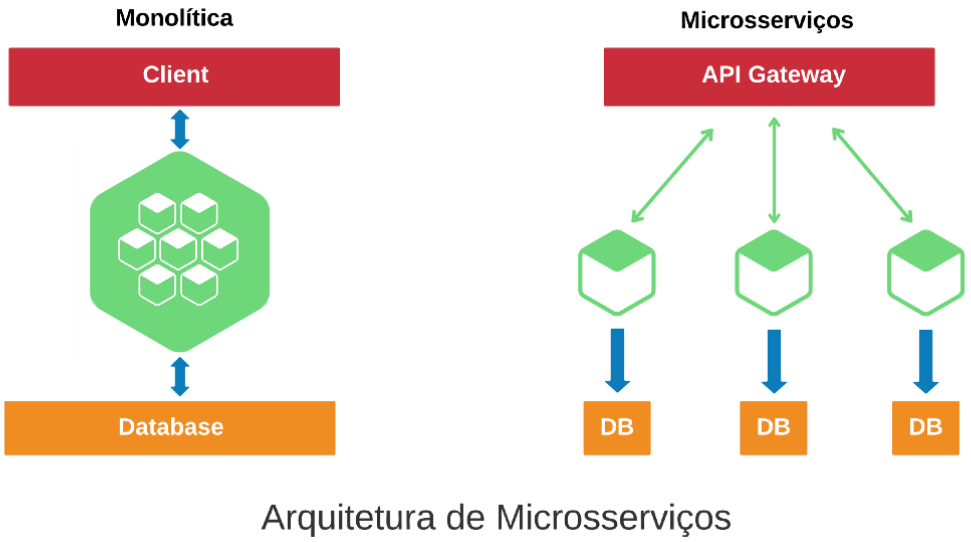


Figura 1 - Arquitetura monolítica e arquitetura de microsserviços
FONTE: https://medium.com/@marcelomg21/arquitetura-de-microsservi%C3%A7os-bc38d03fbf64

A abordagem tradicional para o desenvolvimento de aplicações enfatiza as construções monolíticas. Nelas, todas as partes implantáveis ficam contidas na própria aplicação, em um único bloco.

Já os microsserviços são uma arquitetura e uma abordagem para escrever programas de software. Com eles, as aplicações são desmembradas em componentes mínimos e independentes. Diferentemente da abordagem tradicional monolítica em que toda a aplicação é criada como um único bloco, os microsserviços são componentes separados que trabalham juntos para realizar as mesmas tarefas. Cada um dos componentes ou processos é um microsserviço. Essa abordagem de desenvolvimento de software valoriza a granularidade, a leveza e a capacidade de compartilhar processos semelhantes entre várias aplicações.

7.

Uma biblioteca se trata de um ou mais conjuntos de códigos que fornece funções padrões para resolver determinados problemas e que podem ser utilizados e executados dentro de um código.

Já um fremework se trata de uma estrutura genérica que fornece uma arquitetura padrão, um esqueleto com a qual podemos desenvolver um software. Dentro de um framework pode haver uma coleção de padrões, API's e até mesmo bibliotecas.

8.

Em resumo, é um conjunto de padrão que permite a construção de aplicativos e a sua utilização de maneira não tão evidente para os usuários. Também podemos dizer que uma API é um código pronto, de algum programa ou ferramenta que pode ser implementado no seu projeto.

9.

(a) **Fraco Acoplamento**

Segundo a engenharia de software, acoplamento é a medida de interdependência entre os componentes de um sistema.

Quando um sistema possui entre seus componentes uma relação de interdependência fraca, significa que a dependência entre seus componentes é baixa, ou seja, estão acoplados, mas fracamente acoplados. Isso chamamos de Fraco Acoplamento.

(b) **Alta Coesão**

Coesão está ligada ao princípio da responsabilidade única.

Um componente com Alta Coesão é um componente que possui apenas uma única responsabilidade, que possui em seu conteúdo/suas funções, apenas aquilo que realmente deve fazer.

Ele não assume responsabilidades de outros componentes e não as mistura com as suas, nem delega suas responsabilidades a outros componentes, e não depende dos outros componentes para realizar suas funções conforme suas responsabilidades.

10.

11.

O reuso de código se resume na reutilização de um código desenvolvido anteriormente em um outro software.

Vantagens:

- Aumento da produção com a redução no esforço do desenvolvimento.
- Redução dos custos e do prazo de entrega.
- Aumento da qualidade do produto final, pois as chances das soluções utilizadas no reuso derem erradas são baixas, devido ao fato de já que já foram utilizadas e testadas várias vezes.
- Padronização dos produtos desenvolvidos.

12.

1 - Fase de requisitos: Levantar os requisitos mínimos, estudar a viabilidade e definir o modelo a ser utilizado;

2 - Fase de projeto: Envolve atividades de concepção, especificação, design da interface, prototipação, design da arquitetura;

3 - Fase de implementação: Tradução para uma linguagem de programação das funcionalidades definidas durante as fases anteriores;

4 - Fase de testes: Realização de testes no que foi desenvolvido de acordo com os requisitos;

5 - Fase de produção: Implantação em produção do produto final;

13.

Validação: Avalia um sistema ou componente para determinar se esse satisfaz os requisitos para ele especificados.

Verificação: Avalia se um sistema ou componente para determinar se os produtos de uma dada atividade de desenvolvimento satisfazem as condições impostas no início desta atividade.

Basicamente a verificação se preocupa com o desenvolvimento do software e se ele está sendo implementado corretamente, já a validação se preocupa em verificar se aquilo que foi desenvolvido conforme as especificações.

14.

(a) **teste unitário**

É a fase de teste que tem como finalidade testar individualmente as funcionalidades do software em questão, garantindo que todas as funcionalidades do software sejam testadas pelo menos uma vez.

Ou seja, testar partes menores de forma isolada, essa parte é frequentemente referida como unidade. Esses testes geralmente são desempenhados cedo no processo de desenvolvimento.

(b) **teste funcional**

Os testes funcionais, também conhecidos como testes de caixa-preta, é uma validação de software na qual determinada funcionalidade é verificada, sem levar em conta a estrutura do código-fonte, os detalhes de implementação ou os cenários de execução. Nos testes de caixa-preta, o foco é apenas nas entradas e saídas do sistema, sem se preocupar com a estrutura interna do programa.

(c) **teste de integração**

O objetivo do próximo nível de teste é verificar se as unidades combinadas trabalham bem como um grupo. Testes de integração visam detectar as falhas na interação entre as unidades dentro dos módulos.

(d) **teste sistêmico**

É realizado após o sistema estar pronto, sendo avaliados todos os componentes e funcionalidades. O objetivo do teste de sistema é exercitar todo o software, assegurando que todos os elementos que compõem o sistema estão de acordo com as especificações dos requisitos, incluindo todos os itens de hardware e software que compõem a regra de negócio.

(e) **teste de aceitação**

Esse teste busca verificar se o software atende às expectativas do ponto de vista do cliente e dos usuários finais.

15.

(a) **teste caixa branca**

Possui esse nome porque o testador tem acesso à estrutura interna da aplicação.

Como dito, possui acesso ao código fonte, conhecendo a estrutura interna do produto. Sendo analisados e possibilitando que sejam escolhidas

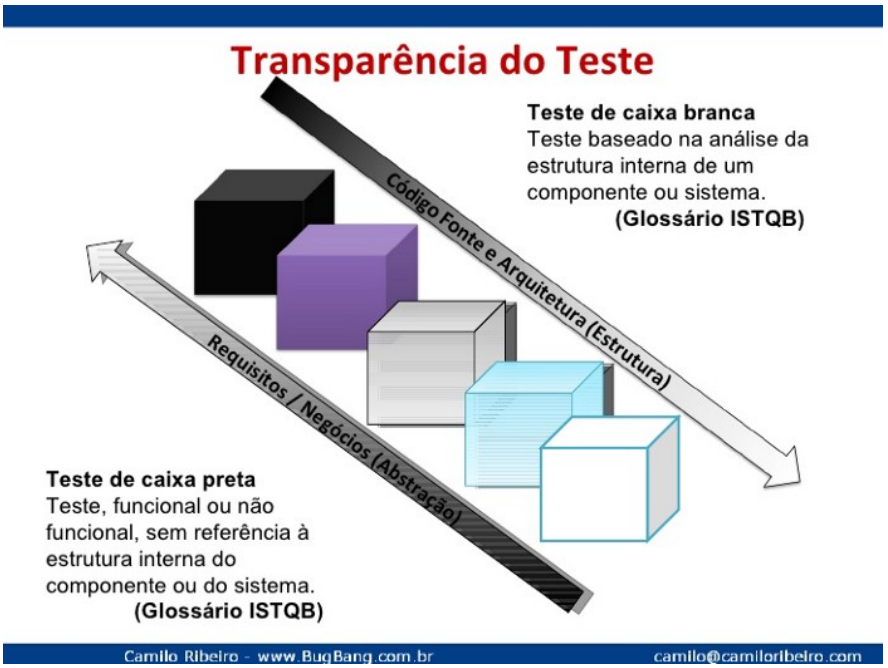
partes específicas de um componente para serem avaliados, permitindo uma busca precisa do comportamento da estrutura.

(b) **teste caixa preta**

Não há acesso ao código fonte e desconhece a estrutura interna do sistema. Baseia-se nos requisitos básicos do software, sendo o foco nos requisitos da aplicação, ou seja, nas ações que deve desempenhar.

(c) **teste caixa cinza**

A ideia geral é combinar esses dois outros tipos para utilizar os pontos fortes de cada um, minimizando suas limitações ou fraquezas. O teste da caixa cinza consiste basicamente em testes profissionais, nos quais os testadores compreendem algumas das maneiras pelas quais o software funciona, mas eles não entendem tudo sobre ele.



FONTE: Você sabe o que é Teste Caixa Branca e Teste Caixa Preta?

REFERÊNCIAS:

Arquitetura de Microsserviços. Refere-se a um estilo de arquitetura… | by Marcelo M. Gonçalves

O que é arquitetura de microsserviços?

Qual a diferença entre Framework e Biblioteca?

Framework x Biblioteca x Toolkit - O que são e qual a diferença?

Acoplamento e Coesão no projeto de Software

Reutilização de Software: Técnicas e Ferramentas

Testes Funcionais de Software

Principais técnicas de testes funcionais | Blog TreinaWeb

Conceito: Teste de Aceitação.

Teste De Aceitação: O Que E Por Que + Tipos A Conhecer

O que é o teste de caixa cinza?

Os Níveis dos Testes de Software – T&M Testes de Software

Ciclo de vida do desenvolvimento de Software

Arquitetura Monolítica e Microsserviços

COMO FUNCIONA A ARQUITETURA CLIENTE SERVIDOR | ARQUITETURA CLIENTE/SERVIDOR

Arquitetura em Camadas - Padrões de Projeto