Matheus Peixoto Ribeiro Vieira - 22.1.4104

Lista 01 - Engenharia de Software

1 - Qual sua definição para o termo “Engenharia de Software”?

Engenharia de Software é um conjunto de boas práticas e normas para que a idealização, criação, desenvolvimento, manutenção e atualização de um programa de computador sejam feitas da melhor maneira possível, acelerando o processo e evitando erros que poderiam ocorrer caso o planejamento do sistema não fosse bem feito.

2 - O que é um projeto segundo o PMBOK (Project Management Body of Knowledge)?

Um projeto é uma atividade que tem como objetivo a criação de um produto, um serviço ou obter um resultado tendo em mente que o mesmo deve possuir um começo e um final.

Dessa forma, o projeto deve ter um tempo finito para a sua concepção, todavia os seus resultados devem perdurar.

Durante a sua execução, ele deve ser progressivamente elaborado, permitindo que ele tenha melhorias e seja refinado a cada passo

3 - O que é arquitetura de software?

A arquitetura pode ser entendida como o conjunto de decisões que são feitas para o processo de desenvolvimento, englobando o padrão de desenvolvimento que será utilizado, como o software que está sendo produzido irá se relacionar com outros, qual a linguagem de programação utilizada, o banco de dados, etc.

Dessa forma, a arquitetura é a responsável por definir como todas as camadas do software estão sendo conectadas e utilizadas.

4 - O que é componente de software? O que é desenvolvimento baseado em componentes?

Um componente de software é uma parte do software que pode ser utilizada por outros, logo, ele é independente e pode funcionar sozinho, o que permite o encapsulamento e reuso do código de forma mais fácil, pois ele tem funções bem definidas e uma boa documentação.

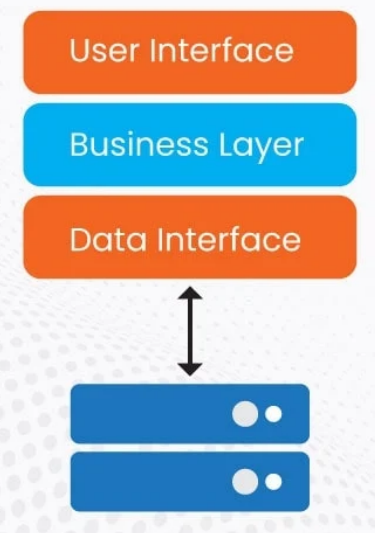
O desenvolvimento baseado em componentes tem como ideia integrar, corretamente, componentes que já existem e que serão desenvolvidos a fim de criar um software, acelerando o processo de desenvolvimento enquanto facilita a manutenção por ser facilmente modificado.

5 - Originalmente o UNIX possuía uma arquitetura monolítica, o Windows possui uma arquitetura cliente-servidor, a pilha de protocolos Ethernet possui uma arquitetura em camadas. Explique como essas arquiteturas são organizadas e como funcionam, utilize figuras. Que vantagens e desvantagens cada uma dessas arquiteturas possui?

- Monolítica:

A arquitetura monolítica é uma arquitetura onde todo o programa está sendo executado em único processo e todos os seus componentes estão sendo conectados dentro deste mesmo programa. Desta forma, quando deve-se realizar uma modificação em um arquivo, todo o código deve ser recompilado para que, então, possa ocorrer o seu deploy.

Em geral, o sistema é dividido em três partes: Os dados que serão salvos, a interface para o usuário e a lógica de negócios que fará com que tudo se interligue, mas estando todos em um mesmo processo, como podemos ver na imagem a seguir.



Como desvantagem, podemos citar o fato de que, caso um procedimento falhe, todo o sistema poderá não funcionar corretamente. Ele é mais difícil de crescer e o seu código é muito grande, pois como tudo concentra-se no mesmo lugar, é necessário conhecer bem onde está mexendo. Ademais, um dos principais problemas é a necessidade de recompilar todo o código caso uma pequena mudança seja feita.

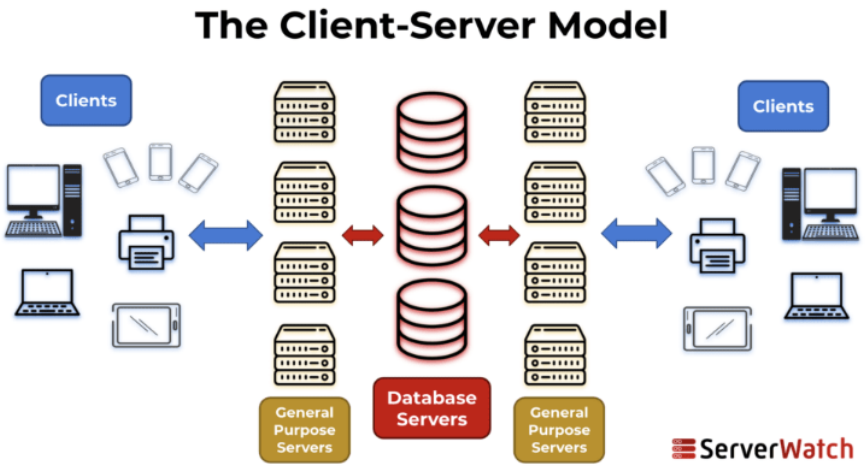
Entre suas vantagens temos a facilidade de iniciar um novo projeto e de publicá-lo, facilitar o entendimento da equipe sobre o que está acontecendo, pois utiliza-se uma única tecnologia.

Cliente-Servidor:

A arquitetura de cliente-servidor baseia-se na existência de uma máquina principal chamada de host, ou servidor, que recebe solicitações, realiza o tratamento das mesmas e devolve uma resposta para quem solicitou.

Por outro lado, o cliente é quem realiza o envio de mensagens e é quem obtém a resposta, não necessitando de realizar grandes processamentos, uma vez que já foi feito.

Em geral, a conexão entre as diferentes estações de trabalho é realizada através da internet ou até mesmo dentro de uma rede privada.



A imagem acima mostra, de forma simplificada, o funcionamento de uma arquitetura de cliente-servidor, onde os diferentes tipos de clientes realizam solicitações para os servidores, que realizam acesso a um banco de dados e retorna as informações obtidas e processadas para o cliente.

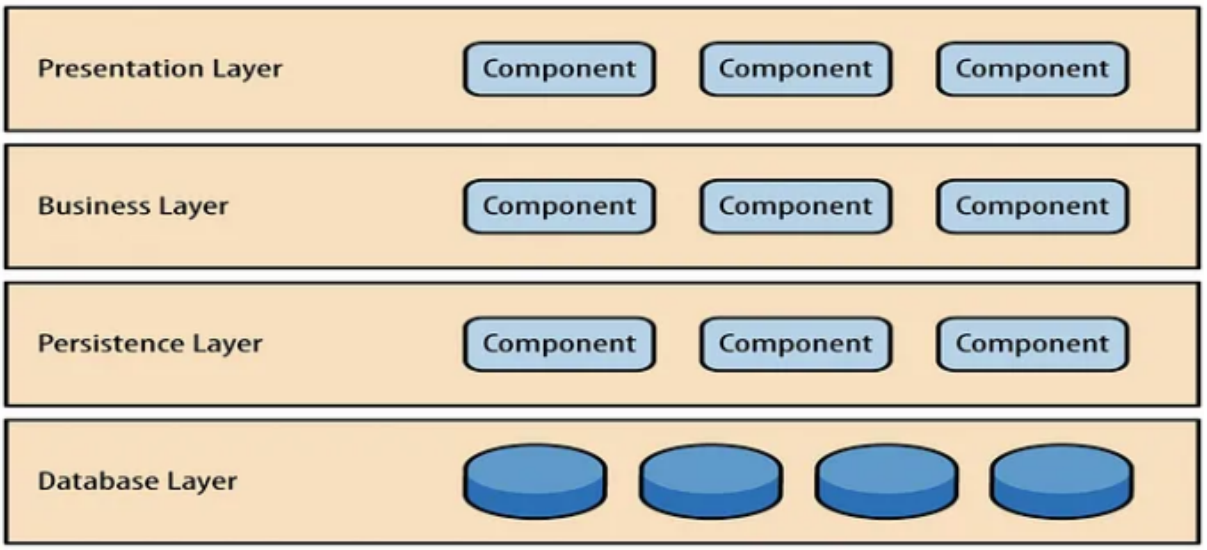
Entre as vantagens temos o controle e acesso central dos dados, a eficiência do sistema juntamente com a facilidade de ser escalado, permite que diferentes plataformas realizem requisições e compartilhem recursos.

Por outro lado, temos como desvantagem o custo para uma implementação inicial, uma solicitação maior de profissionais bem treinados, o risco de ataques de negação de serviço, o que acabaria com o funcionamento do programa, etc.

Arquitetura em camadas:

Organiza as classes e componentes de um programa em n camadas, onde cada uma possui a sua função e elas são interdependentes entre si, ou seja, os componentes se comunicam, mas não dependem uns dos outros.

As camadas são organizadas em uma forma hierárquica onde uma camada pode acessar os métodos somente da camada diretamente inferior a ela na hierarquia.



Supondo um sistema em camadas divididos como na imagem acima, a camada de apresentação seria responsável pela interação com o usuário, a lógica de negócios contém o funcionamento do programa, a camada de persistência lida com o mapeamento dos dados e o banco de dados armazena as informações.

Dessa forma, caso seja solicitado um dado, a partir de uma interação do usuário com um componente da camada de apresentação, as solicitações iriam de camada em camada até o banco de dados para recuperar o que foi solicitado.

Como vantagem do uso dessa arquitetura temos o fato de que uma grande modificação em qualquer uma das camadas não deve refletir nas outras. Ademais, os testes são fáceis de serem realizados uma vez que cada parte do programa pode ser testada separadamente.

Já como desvantagens temos o fato de que mudanças significativas em uma camada não são tão simples já que o sistema é singular. Ademais, quanto maior for o sistema e maior for a quantidade de camadas, mais hardware ele necessitará para conseguir lidar com todos os requests dos usuários.

6 - Quais são os princípios da arquitetura de Microsserviços? Apresente uma figura e a explique.

A arquitetura de microserviços tem como objetivo a separação de um sistema grande que possui diferentes módulos, mas que estão em um único processo, em

diferentes processos que se comunicam entre si, mas não compartilham endereços de memória.

Dessa forma, o isolamento de processos garante que cada parte do sistema é independente e que uma mudança não causará problemas em diferentes outros processos.

Ademais, caso um módulo do sistema esteja sendo muito mais executado que outros, é possível separá-lo dos demais e ter um hardware exclusivo para lidar com ele, liberando recursos para as outras partes do sistema funcionarem sem problemas.

Com essa separação, garantimos outra característica dos microserviços, o isolamento de falhas permitindo o software a continuar executando mesmo que uma parte dele esteja com um problema.

7 - Qual a principal diferença entre as seguintes arquiteturas de software: biblioteca de funções e framework (arcabouço)? Dê exemplo de software largamente conhecidos que possuam essas arquiteturas. Quando uma arquitetura é preferível à outra?

Bibliotecas são conjuntos de funções e classes para a realização de tarefas específicas em um código e que podem ser usadas no projeto de um código. Dessa forma, não é necessário escrever do zero funções complexas que terão o objetivo procurado uma vez que elas já estão feitas.

Por outro lado, um framework são ferramentas que auxiliam na construção de um software, mas a diferença é que ele é utilizado para a criação do mesmo, e não somente em partes específicas. Dessa forma, ele auxilia e acelera o processo de desenvolvimento de algumas partes do código, como a interface gráfica, diminuindo a quantidade de linhas necessárias para se construir algo.

Em suma, pode-se perceber que a diferença está no fato de que um framework permite a criação de um sistema completo enquanto a biblioteca realiza tarefas mais específicas.

Um framework muito conhecido e utilizado é o react, que é a base para o Facebook, Instagram, Netflix, Uber, etc.

Um exemplo de biblioteca muito conhecida é o numpy, onde diversos softwares para análises de dados a utilizam

8 - Defina o conceito de API – (Application Programming Interface).

As API's são como contratos entre softwares, onde um realiza uma requisição para consultar alguma informação enquanto o outro provê os dados solicitados, dessa forma, isto é feito sendo que um não precise se importar com a forma como o outro foi desenvolvido.

Dessa forma, o software que solicita preocupa-se somente com chamada das funções necessárias, sem se preocupar com a implementação de como será feito o processamento. Por outro lado, o software que recebe a solicitação é o encarregado para tratar a solicitação, realizar o processamento dos dados necessários e retornar uma resposta, sem se preocupar com a forma de como os dados serão utilizados.

9 - Defina os seguintes conceitos: (a) Fraco Acoplamento e (b) Alta Coesão.

Fraco acoplamento: significa que os diferentes módulos do sistema não são completamente dependentes entre si. Dessa forma, a mudança de código em um componente não exigirá a modificação de outro, a não ser, claro, que ocorra mudanças nos cabeçalhos de funções, por exemplo. Dessa forma, o reuso do código torna-se mais simples e rápido.

Alta Coesão: Significa que o conjunto de funções que um módulo realiza é bem definido e tarefas extras não são realizadas por ele. Dessa forma, o componente realiza somente o que ele deve fazer, delegando outras funcionalidades para diferentes métodos. Assim, o código fica mais fácil de ser compreendido e organizado, facilitando a manutenção e correção, uma vez que seu escopo é bem delimitado.

10 - O desenvolvedor de software é aconselhado a sempre separar a interface de um programa (API) da sua implementação. Por que?

Separar a API da implementação do código garante que mudanças feitas na API não force a necessidade de uma recompilação do código, desde que os cabeçalhos de função permaneçam os mesmos.

Ademais, a separação garante uma camada a mais de segurança, pois como desenvolvedores terão acesso somente às chamadas de funções, possíveis pontos de falhas serão mais difíceis de serem descobertos uma vez que o sistema está bem encapsulado, o que gera um outro ponto positivo, que é o reuso de código, que é uma das grandes vantagens da API, pois toda uma lógica de programação foi realizada apenas uma vez e servirá para todos os programas sem precisar reescrever o que já foi feito.

11 - O que significa reuso de código? Quais as vantagens e desvantagens? Por que é importante?

O reuso do código é a utilização do que já foi escrito e pensando em outras partes do programa, sem a realização de um copia e cola, mas sim com o uso de chamadas de funções e procedimentos.

Entre as suas principais vantagens destaca-se o ganho de produtividade quando torna-se necessário a atualização de uma função que é invocada em diferentes partes do sistema. Dessa forma, uma única mudança irá refletir em todo o programa sem grandes complicações.

Todavia, uma das suas maiores desvantagens é a dificuldade em transformar uma função o mais genérica possível de forma que ela possa ser usada em diferentes partes do programa, pois esta não é uma função muito trivial, em grande parte das vezes.

12 - Quais são as fases no desenvolvimento de um projeto de software? Quais atividades são realizadas em cada fase?

Tomando como base o modelo de desenvolvimento em cascata, temos as seguintes fases:

Levantamento de requisitos: São feitas listas de como o sistema deve funcionar e quais características ele deve ter. Geralmente são feitas a partir de uma conversa com o cliente, onde o mesmo diz o que deve ser feito.

Análise: Todos os requisitos são analisados e é verificado se há dúvidas, ambiguidades e possíveis problemas para as etapas seguintes em relação aos requisitos que foram levantados.

Projeto: É feito um planejamento de como seguirá as próximas etapas, criando cronogramas, definindo tarefas, montando o time e modelando interfaces.

Codificação: Os desenvolvedores implementam o que foi solicitado nos requisitos e na parte de projeto

Testes: É verificado se o software que foi desenvolvido está de acordo com os requisitos e se ele funciona de forma correta como deveria, sem apresentar bugs e erros críticos.

Implementação: O sistema é implementado para o funcionamento e o cliente pode finalmente ver o projeto como um todo

13 - Qual a diferença entre verificação e validação de software?

Verificação: Garante que um sistema realiza o que foi proposto a fazer; Validação: Garante que o sistema realiza o que o cliente espera que ele faça.

14 - Defina cada um dos seguintes níveis de teste de software: (a) teste unitário, (b) teste funcional, (c) teste de integração, (d) teste sistêmico e (e) teste de aceitação.

Teste unitário: São testes feitos de forma automatizada com pequenas partes do código, de forma que elas são testadas separadamente do sistema, a fim de verificar se o resultado esperado é retornado.

Teste funcional: São testes que verificam se uma aplicação está funcionando da forma adequada, não preocupando-se com a implementação do código, mas sim se ele está atendendo as especificações necessárias.

Teste de integração: Uma maior parte do código é testada para ver se como uma parte maior ele ainda funciona. Dessa forma, podem ser testadas diferentes classes ao mesmo tempo, juntamente com pacotes e serviços distintos.

Teste sistêmico: São testes que simulam uma utilização real do software que está sendo desenvolvido a fim de exercitar grande parte das funcionalidades do sistema.

Teste de aceitação: São testes realizados pelo cliente final, feitos de forma manual, com o mesmo inserindo as informações desejadas no sistema a fim de validar a

implementação realizada. Sendo que eles podem ser divididos em um teste alfa, em um ambiente controlado e com poucos usuários, e um teste beta, com mais usuários em um ambiente não controlado.

15 - Que é: (a) teste caixa branca, (b) teste caixa preta e (c) teste caixa cinza?

Teste caixa-branca: São testes que possuem acesso ao código fonte, levando em conta desvios condicionais, estruturas de repetição, etc. A fim de verificar se diferentes possibilidades estão sendo cobertas.

Teste caixa-preta: São conhecidos também como testes funcionais, tendo como objetivo testar se os requisitos do sistema foram implementados de forma correta, sem se preocupar com a parte de codificação, resumindo-se em testes de entrada e saída.

Teste caixa-cinza: São testes que mesclam características entre os testes de caixa-branca e testes de caixa-preta. geralmente, o seu objetivo é encontrar erros específicos do sistema, feitos a partir de uma perspectiva do usuário a fim de reduzir o tempo entre testes funcionais e não funcionais.

Referências

- https://www.managementstudyguide.com/what-is-project.htm Acesso em 11/10/2023
- https://www.pucrs.br/facin-prov/wp-content/uploads/sites/19/2016/03/tr026.pdf Acesso em 11/10/2023
- https://www.zappts.com.br/arquitetura-monolitica-e-microsservicos/ Acesso em 11/10/2023
- https://www.serverwatch.com/guides/client-server-model/ Acesso em 11/10/2023
- https://www.dio.me/articles/framework-vs-biblioteca-qual-a-diferenca-e-como-escolher-a-melhor-opcao-para-seu-projeto Acesso em 11/10/2023
- https://www.redhat.com/pt-br/topics/api/what-are-application-programming-interfaces Acesso em 11/10/2023
- https://acervolima.com/teste-de-caixa-cinza-teste-de-software/ Acesso em 11/10/2023
- https://engsoftmoderna.info/ Acesso em 11/10/2023