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

O termo engenharia de software traz os conceitos de engenharia (criação, construção, análise, desenvolvimento e manutenção) para a produção de softwares com o objetivo de produzi-los de maneira econômica, que seja confiável e que trabalhe em máquinas reais. Os fundamentos científicos envolvem o uso de modelos abstratos e precisos que permitem ao engenheiro especificar, projetar, implementar e manter sistemas de software, avaliando e garantindo sua qualidade. Além disso, deve oferecer mecanismos para se planejar e gerenciar o processo de desenvolvimento.

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

O PMBOK conceitua um projeto como um esforço temporário, ou seja, finito. Tem, portanto, início e fim bem determinados e empreendidos para se alcançar um objetivo exclusivo, ou seja, um resultado específico que o torna único.

Os projetos são executados por pessoas e com limitações de recursos e são planejados, executados e controlados ao longo de seu ciclo de vida. De forma simples, é possível afirmar que os projetos se diferenciam dos processos e das operações, porque não são contínuos e repetitivos pois possuem caráter único.

Para que se tenha uma dimensão melhor da importância dos projetos, basta compreender que, para que qualquer organização alcance seus objetivos, ela precisará de esforços organizados. Isso é válido desde a construção de uma nova fábrica até a ampliação de uma unidade operacional, por exemplo.

Ele pode ser melhor compreendido por meio dos processos que o compõem, organizados em cinco grupos:

• Iniciação;

• Planejamento;

• Execução;

• Monitoramento e controle;

• Encerramento.

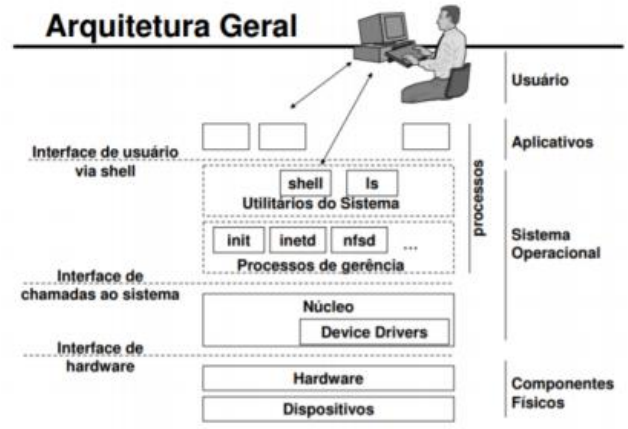
3. O que é arquitetura de software?

Arquitetura de software é um conceito abstrato que trata da relação entre o mapeamento dos componentes de um software e os detalhes que são levados em conta na hora de implementar esses elementos na forma de código.

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

Componentes são unidades independentes de um software com o qual interagimos através de interfaces, estas são: Interface Requires e Interface Provides (API). Desenvolvimento baseado em componentes (DBC) é o desenvolvimento de um sistema com foco no desenvolvimento dos componentes independentes que irão compor o software quando interligados e que permitem uma fácil manutenção e alteração de parte do software.

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?



As vantagens do UNIX:

● Memória virtual: É possível utilizar uma grande quantidade de programas, usando

pouca memória física.

● Caixa de ferramenta: O sistema operacional fornece uma grande coleção de pequenos

utilitários e comandos que fazem tarefas específicas de forma eficiente.

● Personalização: O Unix consegue unir diversos utilitários e comandos em um número

ilimitado de configurações, para realizar uma série de tarefas complexas.

● Portabilidade: Está disponível para diversos tipos de computadores.

As desvantagens são:

● Interface: A interface do sistema operacional é baseada em linha de comando, sendo

ela feita para ser usada por programadores e usuários experientes.

● Comandos especiais: Geralmente usam nomes complexos, e não informam o usuário

do que estão fazendo.

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

explique.

Arquitetura de Micros serviços projeta o sistema de forma a decompor as aplicações em propósitos, isto é, esta organização divide um monólito. Cada micro serviço é responsável por uma funcionalidade no sistema, fazendo essa atividade com eficiência.

● Alguns princípios:

○ Alta coesão: esse princípio indica que o serviço deve ter uma única

responsabilidade no sistema.

○ Autônomos: um micro serviço deve ser autônomo, isto é, independente de outros serviços.

○ Resiliência: ser capaz de validar os dados recebidos (mesmo que estes

estejam corrompidos) e tratar a perda ou falha na comunicação com outro serviço da cadeia, sem quebrar o fluxo da aplicação.

○ Observável: poder consultar o status de um sistema em tempo real.

○ Automatização: automatizar algumas tarefas como envio de resultados de forma contínua

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?

Em um framework é definida a arquitetura de uma aplicação, como a divisão de classes e objetos, as responsabilidades chave de cada e como essas classes e objetos se relacionam. Dessa forma, é possível inferir que o framework lida com o reuso de design e controla o fluxo da aplicação. Enquanto que em uma biblioteca de funções são definidas funções em uma determinada

linguagem de programação, para serem utilizadas em um programa, portanto, uma biblioteca lida com o reuso de código e o programador que a utiliza define o fluxo da aplicação.

Assim a diferença entre um framework e uma biblioteca de funções se encontra na forma como lidam o fluxo de uma aplicação, em uma biblioteca o programador tem independência para controlá-lo, já em um framework, esse fica dependente das diretrizes do framework.

Alguns exemplos de softwares que utilizam a arquitetura de framework seriam: Facebook, Discord e Airbnb, eles utilizam o framework React Native em suas aplicações mobile, por outro lado, softwares como Google, Instagram e Wikipedia utilizam a biblioteca de funções JavaScript e jQuery em suas respectivas aplicações web.

Assim é possível concluir que em situações que necessitam de uma maior flexibilidade, o ideal é utilizar a biblioteca de funções, visto que o programador tem maior controle, já em situações que necessitam treinar novos membros de equipe e criar aplicações em larga escala o ideal é utilizar uma framework, devido a documentação para o treinamento e implementação prévia das normas de design.

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

É um intermediário de software que permite que dois aplicativos se comuniquem.

Quando você usa um aplicativo no seu celular, ele se conecta à Internet e envia dados para um servidor. O servidor recupera esses dados, interpreta-os, executa as ações necessárias e os envia de volta ao seu telefone. O aplicativo interpreta esses dados e apresenta as informações desejadas de forma legível. Isso é o que é uma API - tudo isso acontece via API.

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

a) Fraco acoplamento: O fraco acoplamento pode ser definido como a baixa

interdependência entre os componentes de um software.

b) Alta coesão: A alta coesão é definida como a capacidade de um componente dentro do software de realizar apenas um papel determinado pelo programador e de forma independente, ou seja, esse componente não realiza ou depende dos papéis desempenhados por outros componentes e não necessita de outros componentes para desempenhar sua função.

10. O desenvolvedor de software é aconselhado a sempre separar a interface de um

programa (API) da sua implementação. Por que?

Separar as aplicações front-end e back-end (API) é uma grande vantagem, pois é importante para proteger o armazenamento de dados, possibilitando apenas a troca de informações. O processo de desenvolvimento da aplicação ocorre com mais facilidade, já que não há dificuldades para acoplar recursos uma vez que o código possui toda sua estrutura organizada e desacoplada de interfaces. A API permite que o banco de dados de diferentes servidores seja acessado pela aplicação, se tornando importante para o desenvolvimento em grandes aplicações. Portanto, sua utilização resulta em uma garantia de mais praticidade e confiabilidade.

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

importante?

Reuso de código (software) é uma estratégia em que o desenvolvimento de software é baseado no reuso de software existente. As vantagens de reutilizar um código são: redução dos custos de desenvolvimento, confiança aumentada, redução do risco de processo, uso eficaz de especialistas, conformidade com padrões e desenvolvimento acelerado. Mesmo tendo suas vantagens, é possível encontrar problemas em: maiores custos de manutenção, falta de ferramentas de suporte, criação - manutenção e uso de uma biblioteca de componentes e como encontrar, compreender e adaptar melhor os códigos reusáveis.

12. Quais são as fases no desenvolvimento de um projeto de software? Quais atividades

são realizadas em cada fase?

Para o desenvolvimento de um projeto de software é necessário seguir 4 fases, sendo elas: Fase de diagnóstico, levantamento e análise de requisitos, de desenvolvimento e a etapa de implantação.

● Fase de diagnóstico: Nesta fase o time de desenvolvimento é responsável por conhecer o cliente a fundo. Dessa forma, é essencial que o problema seja extremamente detalhado e explicado para que o software atenda todas as necessidades.

● Levantamento e análise de requisitos: Nesta fase de Levantamento e Análise de requisitos, deve-se pensar em alternativas de solução. Pois, as alternativas devem sempre se basear em um diagnóstico previamente feito na fase anterior (daí a importância de um diagnóstico bem feito). Além disso, esse momento exige muito cuidado e atenção por parte do time de desenvolvimento. Ou seja, é extremamente importante que nenhum detalhe ou requisito seja esquecido ou deixado de lado. Caso contrário, o desenvolvimento pode ser

interrompido e o produto final pode sofrer com alterações inesperadas ou prazos alterados. Após o levantamento de requisitos, é feita a análise. Tal análise consiste na definição de atividades, alocações de membros, definição de prazos, entre outros. E por fim, é feita uma breve validação. Essa validação é feita avaliando-se a eficiência e relevância de todos os requisitos levantados.

● Fase de Desenvolvimento: É aqui o momento em que o código será realmente desenvolvido, criado. Os grupos se organizam, as tarefas são organizadas, e os responsáveis iniciam o desenvolvimento do software. Neste momento, é muito comum ouvirmos falar sobre uma grande gama de “métodos ágeis” de desenvolvimento de software. Estes métodos seriam nada mais do que processos e etapas que, caso seguidas, agilizam e otimizam o desenvolvimento. E com isso, o produto final é entregue muito mais rápido e de maneira muito mais satisfatória.

● Etapa de Implantação: E por fim, na última etapa é realizada a fase de implantação do software. Nela, o código é instalado no ambiente do cliente (sistema operacional, servidor específico, entre outros). Além disso, são criados Manuais do Sistema, que auxiliam o entendimento do produto final. Estes materiais facilitam um possível treinamento para as pessoas que usarão futuramente o novo software. Além disso, trazem mais clareza e confiança ao cliente no momento do uso, fornecendo uma autonomia extremamente importante. E caso necessário, também é realizada uma migração/importação de dados e informações antigas para o novo software.

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

A verificação de software pode ser realizada por uma equipe de desenvolvimento de software e consiste em analisar se o que foi implementado está correto.

A validação de software (também conhecida por assurance ou quality assurance) pode ser feita apenas pelos clientes e usuários e consiste em analisar se o software desenvolvido é aquele que se deseja, com o comportamento esperado e alta qualidade.

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.

a) Teste unitário: testes de funções e métodos (menores unidades de código em linguagens imperativas e programação orientada a objetos, respectivamente), da maneira mais isolada possível, e analisando se, para um conjunto determinado de entradas, as funções ou métodos em questão retornam os valores corretos e esperados;

b) Teste funcional: testa se o uso coordenado dos métodos ou funções que compõem um determinado módulo ou classe atinge o objetivo determinado. Muitas vezes utilizado em módulos ou classes que possuem uma funcionalidade completa e cujas funções ou métodos não podem ser testados isoladamente (Ex.: API para a construção de um servidor, que envolve métodos de abrir e fechar uma conexão, enviar e receber dados, etc.);

c) Teste de integração: teste que cruza várias interfaces ou módulos, analisando a integração destes, e se estão comunicando de forma correta;

d/e) Teste sistêmico/Teste de aceitação: utiliza um programa ou sistema como um todo e analisa se ele atinge as necessidades do cliente.

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

a) Teste caixa branca: testes abertos em que se tem acesso ao código. Este tipo de teste é evitado pois, se um software já estiver funcionando na visão de um cliente, não existe razão para testes internos serem realizados, embora seja difícil implementar métodos e funções (nível mais baixo) sem saber se estão funcionando como o esperado;

b) Teste caixa preta: testes de software já empacotados e implantados no ambiente de produção (ou num ambiente que simula tal), em que não se tem acesso ao código. Este tipo de teste é mais difícil de ser projetado e não se preocupa com a implementação. Existem ferramentas para realizar este tipo de teste, como o Appium (que gera testes para Android) e o Selenium (que gera testes para web);

c) Teste caixa cinza: nível intermediário entre o teste caixa branca e o teste caixa preta. Em testes funcionais, em que um módulo ou classe está sendo testado, não é necessário ter acesso à implementação, basta ter acesso à API ou interface e ser capaz de utilizar este componente. No entanto, ainda é necessário utilizar a mesma linguagem e tecnologia para esse tipo de teste.