Universidade Federal de Pernambuco :: Centro de Informática
Sistemas de Informação :: Engenharia de Software
Prof. Vinicius Cardoso Garcia

INSTRUÇÕES: Leia o enunciado com atenção e cuidado, e responda a questão do exercício.

- Esta avaliação tem 25 questões objetivas, com 5 alternativas, para um total de 10 pontos com **sua resolução sendo individual e sem consulta**.
- Adicionalmente, esta avliação também tem 1 questão EXTRA **não obrigatória**.
- Organize o tempo, a prova tem duração de **até 1 hora e 40 minutos**.
- Por favor, mantenha-se focado na prova. Não é permitido **abrir qualquer aba** diferente do formulário do EE Final durante a realização da avaliação a não ser, em caráter **EXTRAORDINÁRIO**, o uso do Bloco de Notas, vim, gedit ou similares/equivalentes.
- Não é permitido **nada em cima da mesa ou no colo**. Guardem os celulares no bolso ou na mochila (ou equivalente) e a mesma deve estar no chão.
- Responda todas as questões no **formulário de respostas**.
- Dúvidas podem ser expostas, **publicamente**, durante **os primeiros 30 minutos**.
- Não é permitido sair da sala (i.e. ir ao sanitário) **durante a realização da prova**. Certifique-se de atender a essa necessidade antes do início da avaliação.
- **Entender o enunciado faz parte da avaliação**.

Bem-vindo à seção de questões objetivas da prova. Por favor, leia atentamente cada enunciado e escolha a opção que melhor responde à pergunta proposta. Lembre-se de que esta seção contém 20 questões, cada uma valendo 0,5 pontos, totalizando 10 pontos.

É fundamental que você mantenha o foco e a concentração durante toda a resolução das questões. Não hesite em reler os enunciados e as alternativas antes de selecionar sua resposta.

1. Qual das seguintes opções melhor descreve uma preocupação ética relacionada ao impacto social da tecnologia no desenvolvimento de software?

(a) A escolha de linguagens de programação baseada na popularidade em vez da eficiência.
(b) O uso de algoritmos de aprendizado de máquina que podem perpetuar vieses existentes em dados históricos.
(c) A preferência por modelos de desenvolvimento ágil em detrimento de modelos de cascata.
(d) A seleção de plataformas de hospedagem com base em considerações de custo.
(e) O uso de padrões de projeto para garantir a reutilização do código.

2. Qual das seguintes características é mais associada a bancos de dados não relacionais?

(a) Estrutura rígida de tabelas com linhas e colunas
(b) Suporte a transações complexas e junções de tabelas
(c) Flexibilidade na estrutura de dados, como documentos JSON
(d) Uso exclusivo de linguagem SQL para consultas
(e) Estrita aderência ao modelo ACID (Atomicidade, Consistência, Isolamento,
(f) Durabilidade)

3. O que é integração de sistemas e por que é importante em Engenharia de Software?

(a) Integração de sistemas é o processo de conectar diferentes sistemas e aplicações para que possam compartilhar dados e funcionalidades. É importante em Engenharia de Software porque permite a criação de soluções mais completas e eficientes, combinando os recursos de sistemas existentes.
(b) Integração de sistemas é a prática de testar sistemas individualmente antes de integrá-los em um ambiente de produção. É importante em Engenharia de Software porque garante a qualidade e confiabilidade das soluções desenvolvidas.

Boa sorte!

(c) Integração de sistemas é o processo de documentar os requisitos de diferentes sistemas e aplicações para garantir que eles possam funcionar juntos de forma harmoniosa. É importante em Engenharia de Software porque facilita a comunicação entre equipes de desenvolvimento.
(d) Integração de sistemas é a atividade de criar interfaces de usuário consistentes em diferentes sistemas e aplicações. É importante em Engenharia de Software porque melhora a experiência do usuário final ao fornecer uma interface unificada para acessar funcionalidades distribuídas.
(e) Integração de sistemas é a prática de garantir que todos os sistemas e aplicações estejam protegidos contra ameaças de segurança. É importante em Engenharia de Software porque protege os dados e recursos corporativos contra acesso não autorizado.

4. O que é computação em nuvem e por que é relevante em Engenharia de Software?

(a) Computação em nuvem é um modelo de entrega de serviços de computação através da internet, oferecendo recursos como armazenamento, processamento e aplicativos sob demanda. É relevante em Engenharia de Software porque permite aos desenvolvedores acessar facilmente recursos de infraestrutura para desenvolver, testar e implantar aplicativos de forma rápida e escalável.
(b) Computação em nuvem é um modelo de entrega de serviços de computação que requer o uso de hardware específico, como servidores dedicados, para funcionar. É relevante em Engenharia de Software porque oferece uma maneira mais econômica de implantar aplicativos em comparação com servidores físicos tradicionais.
(c) Computação em nuvem é um modelo de entrega de serviços de computação que requer a instalação de software especializado nos dispositivos dos usuários finais para acessar os recursos na nuvem. É relevante em Engenharia de Software porque simplifica o processo de implantação de aplicativos para os usuários finais.
(d) Computação em nuvem é um modelo de entrega de serviços de computação que utiliza redes de computadores locais para fornecer recursos de computação sob demanda. É relevante em Engenharia de Software porque oferece uma maneira mais segura de armazenar e acessar dados sensíveis.
(e) Computação em nuvem é um modelo de entrega de serviços de computação que utiliza hardware virtualizado para fornecer recursos de computação sob demanda. É relevante em Engenharia de Software porque permite aos desenvolvedores criar e implantar aplicativos independentemente da infraestrutura subjacente.

5. Qual das seguintes afirmações melhor descreve o impacto do DevOps na Engenharia de Software?

(a) DevOps foca exclusivamente na automação de processos de desenvolvimento, sem interação com operações de TI.
(b) DevOps promove a separação entre equipes de desenvolvimento e operações de TI para aumentar a eficiência.
(c) DevOps enfatiza a colaboração e comunicação entre as equipes de desenvolvimento e operações de TI, visando melhorar a agilidade e a qualidade do desenvolvimento de software.
(d) DevOps elimina a necessidade de testes de software, confiando inteiramente em operações de TI para garantir a qualidade.
(e) DevOps é uma prática que se aplica apenas ao desenvolvimento de softwares de grande escala, sendo irrelevante para projetos menores.

6. Qual é uma abordagem eficaz para implementar refatoração em um projeto de software existente sem comprometer a entrega contínua de novas funcionalidades?

(a) Suspender o desenvolvimento de novas funcionalidades até que a refatoração esteja completa, para evitar conflitos de código.
(b) Incorporar tarefas de refatoração como parte do processo de desenvolvimento regular, intercalando-as com a implementação de novas funcionalidades.

Boa sorte!

(c) Delegar a refatoração exclusivamente para uma equipe separada, mantendo a equipe principal focada no desenvolvimento de novas funcionalidades.
(d) Realizar a refatoração apenas após o lançamento de todas as funcionalidades planejadas, para minimizar interrupções no ciclo de desenvolvimento.
(e) Utilizar ferramentas automatizadas para realizar toda a refatoração de uma vez, acelerando o processo e reduzindo a necessidade de intervenção manual.

7. Como a implementação de uma arquitetura de microsserviços pode impactar a estratégia de testes em um projeto de desenvolvimento de software?

(a) Em uma arquitetura de microsserviços, não é necessário realizar testes, pois cada serviço é independente e autocontido.
(b) Os testes tornam-se mais desafiadores, pois é necessário validar cada microsserviço individualmente, além de testar a integração e comunicação entre eles.
(c) A arquitetura de microsserviços simplifica os testes, reduzindo-os apenas a verificações de usabilidade e desempenho.
(d) Em uma arquitetura de microsserviços, os testes são centralizados e executados apenas no final do processo de desenvolvimento para economizar tempo.
(e) A estratégia de teste em microsserviços foca exclusivamente em testes de segurança, ignorando outros aspectos como funcionalidade e desempenho.

8. No contexto do desenvolvimento ágil de software, qual é a melhor prática para integrar a garantia de qualidade (QA) de maneira contínua e eficaz ao longo do ciclo de vida do projeto?

(a) Delegar todas as atividades de QA para uma fase posterior do desenvolvimento, a fim de acelerar as fases iniciais.
(b) Implementar QA como um processo contínuo e integrado, envolvendo revisões frequentes de código, testes automatizados e feedback regular das partes interessadas.
(c) Focar exclusivamente em testes manuais durante as reuniões de revisão do sprint para validar a qualidade.
(d) Limitar a atividade de QA apenas aos desenvolvedores, sem envolver testadores ou analistas de QA no processo.
(e) Realizar a garantia de qualidade apenas nas fases de lançamento, para evitar retrabalho constante.

9. Em um cenário de desenvolvimento de software, quando a integração de um framework de automação de testes é mais benéfica?

(a) Apenas durante as fases iniciais do desenvolvimento para estabelecer padrões de código.
(b) Somente após o lançamento do produto, para realizar manutenção e testes de regressão.
(c) Ao longo de todo o ciclo de vida do desenvolvimento, desde as fases iniciais até a manutenção pós-lançamento, para contínua verificação e garantia de qualidade.
(d) Exclusivamente durante as fases de testes de aceitação do usuário para validar os requisitos finais do produto.
(e) Apenas em projetos grandes, pois em projetos pequenos, a automação de testes não é custo-efetiva.

10. Durante a fase de análise e especificação de requisitos em um projeto de software, qual é a melhor maneira de garantir que os requisitos identificados estão alinhados com os objetivos de negócios do cliente e são viáveis dentro das limitações tecnológicas e de recursos?

(a) Desenvolver um protótipo avançado do sistema antes de finalizar os requisitos.
(b) Realizar uma análise de risco detalhada para cada requisito proposto.

Boa sorte!

(c) Envolver todas as partes interessadas, incluindo a equipe de desenvolvimento, o cliente e os usuários finais, em uma revisão e validação conjunta dos requisitos.
(d) Focar apenas nos requisitos técnicos, assumindo que estes automaticamente atendem aos objetivos de negócios.
(e) Delegar a validação dos requisitos exclusivamente aos usuários finais, sem a interferência da equipe de desenvolvimento.

11. No contexto do planejamento de projetos de software, por que é importante realizar estimativas precisas de tempo e custo, e como essas estimativas devem ser utilizadas durante o ciclo de vida do projeto?

(a) As estimativas precisas são importantes apenas na fase inicial para definir o orçamento e o cronograma, e não precisam ser ajustadas ao longo do projeto.
(b) Estimativas precisas são cruciais para garantir que o software nunca exceda o orçamento inicial, independentemente das mudanças no escopo do projeto.
(c) Estimativas de tempo e custo são importantes para planejamento e alocação de recursos, e devem ser regularmente ajustadas com base no progresso e mudanças no escopo do projeto.
(d) O principal objetivo das estimativas é garantir que todas as funcionalidades sejam desenvolvidas, independentemente do tempo e custo necessários.
(e) Estimativas de tempo e custo são feitas apenas para cumprir requisitos contratuais e não têm impacto significativo no gerenciamento do projeto.

12. Considerando as práticas do Desenvolvimento Orientado ao Comportamento (BDD), qual é a principal finalidade de se utilizar a linguagem de definição de comportamento, como Gherkin, na elaboração de cenários de teste?

(a) Fornecer um meio de realizar testes de desempenho e escalabilidade de forma mais eficiente.
(b) Permitir a automação de testes utilizando linguagens de programação específicas.
(c) Facilitar a comunicação entre os membros técnicos e não técnicos da equipe, proporcionando uma descrição clara dos comportamentos esperados do sistema.
(d) Aumentar a complexidade dos testes para cobrir casos mais avançados e técnicos.
(e) Servir exclusivamente como documentação técnica para futuras referências de desenvolvimento.

13. Qual dos seguintes comandos do Git cria uma cópia local de um repositório remoto?

(a) git commit
(b) git clone
(c) git branch
(d) git checkout
(e) git merge

14. Selecione a alternativa que melhor explica o papel dos contêineres Docker no desenvolvimento de serviços e discuta como essa tecnologia facilita o processo de implantação e escalabilidade de aplicativos distribuídos.

(a) Os contêineres Docker são uma tecnologia de virtualização que permite empacotar e executar aplicativos e suas dependências em ambientes isolados chamados contêineres. No desenvolvimento de serviços, os contêineres Docker fornecem uma maneira consistente e portável de empacotar e distribuir serviços, juntamente com todas as suas dependências, garantindo que o ambiente de execução seja consistente em diferentes plataformas. Além disso, os contêineres Docker facilitam a implantação de aplicativos distribuídos, permitindo que eles sejam implantados rapidamente e escalados conforme necessário, devido à sua leveza e eficiência na alocação de recursos.

Boa sorte!

(b) Os contêineres Docker são uma tecnologia de gerenciamento de configuração que permite a criação e implantação automatizadas de aplicativos em ambientes distribuídos. No desenvolvimento de serviços, os contêineres Docker simplificam o processo de empacotamento e distribuição de serviços, garantindo que todos os componentes necessários estejam presentes no ambiente de execução. Além disso, os contêineres Docker oferecem uma solução escalável para implantação de aplicativos distribuídos, permitindo que novas instâncias de serviços sejam criadas e dimensionadas conforme necessário, sem comprometer a estabilidade do sistema.

(c) Os contêineres Docker são uma tecnologia de orquestração que facilita a automação do ciclo de vida de desenvolvimento, implantação e gerenciamento de serviços distribuídos. No desenvolvimento de serviços, os contêineres Docker simplificam o processo de desenvolvimento, permitindo que os desenvolvedores criem ambientes de desenvolvimento consistentes em suas máquinas locais. Além disso, os contêineres Docker facilitam a implantação de aplicativos distribuídos, permitindo que eles sejam implantados em qualquer ambiente de nuvem compatível com Docker e escalados automaticamente conforme a demanda.

(d) Os contêineres Docker são uma tecnologia de virtualização que permite empacotar e distribuir aplicativos e suas dependências em ambientes isolados chamados contêineres. No desenvolvimento de serviços, os contêineres Docker simplificam o processo de desenvolvimento, permitindo que os desenvolvedores criem ambientes de desenvolvimento consistentes em suas máquinas locais. Além disso, os contêineres Docker facilitam a implantação de aplicativos distribuídos, permitindo que eles sejam implantados rapidamente e escalados conforme necessário, devido à sua eficiência e portabilidade.

(e) Os contêineres Docker são uma tecnologia de virtualização que permite empacotar e distribuir aplicativos e suas dependências em ambientes isolados chamados contêineres. No desenvolvimento de serviços, os contêineres Docker simplificam o processo de empacotamento e distribuição de serviços, garantindo que todos os componentes necessários estejam presentes no ambiente de execução. Além disso, os contêineres Docker facilitam a implantação de aplicativos distribuídos, permitindo que eles sejam implantados em qualquer ambiente de nuvem compatível com Docker e escalados automaticamente conforme a demanda.

15. Selecione qual a alternativa que melhor explica brevemente a diferença entre Web Services baseados em SOAP e API RESTful em termos de arquitetura e protocolos utilizados.

(a) Web Services baseados em SOAP utilizam protocolos de transporte como HTTP, enquanto API RESTful utiliza o protocolo SMTP.

(b) API RESTful é baseada em padrões abertos como XML e WSDL, enquanto Web Services baseados em SOAP usam formatos de dados como JSON.

(c) Web Services baseados em SOAP seguem um estilo de arquitetura mais orientado a mensagem, enquanto API RESTful é centrada em recursos e utiliza métodos HTTP.

(d) Tanto Web Services baseados em SOAP quanto API RESTful dependem fortemente de estado de sessão para garantir a segurança das transações.

(e) API RESTful é mais adequada para integração entre sistemas legados, enquanto Web Services baseados em SOAP são mais utilizados para desenvolvimento de aplicações web modernas.

16. No contexto de aprendizado de máquina aplicado à engenharia de, qual das seguintes tarefas é mais desafiadora e complexa de ser implementada?

(a) Automatização de testes unitários de software.

(b) Classificação de e-mails como spam ou não spam.

(c) Previsão de falhas de software com base em padrões históricos de bugs.

(d) Reconhecimento de padrões em imagens para autenticação biométrica.

(e) Análise de sentimentos em textos de avaliações de usuários.

Boa sorte!

17. Selecione a alternativa que apresenta a melhor discussão acerca dos desafios e considerações éticas associados à migração de sistemas legados para a nuvem. Como esses desafios podem ser superados e quais são as implicações éticas envolvidas nesse processo?

(a) Os desafios incluem a interoperabilidade entre sistemas legados e ambientes de nuvem, preocupações com a segurança e a privacidade dos dados durante a migração e a necessidade de garantir a conformidade com regulamentações governamentais. Esses desafios podem ser superados por meio de uma abordagem gradual de migração, uso de ferramentas de migração e implementação de medidas de segurança robustas. As implicações éticas envolvem questões de privacidade dos dados, transparência no tratamento das informações dos usuários e responsabilidade pelo armazenamento e processamento de dados sensíveis.

(b) Os desafios incluem a migração de dados sensíveis e complexos, a adaptação de sistemas legados para ambientes de nuvem e a integração de sistemas legados com novas tecnologias e arquiteturas. Esses desafios podem ser superados por meio de uma análise cuidadosa dos requisitos de migração, desenvolvimento de estratégias de migração personalizadas e implementação de testes abrangentes para garantir a funcionalidade e a segurança dos sistemas migrados. As implicações éticas envolvem questões de privacidade dos dados, transparência no tratamento das informações dos usuários e responsabilidade pelo armazenamento e processamento de dados sensíveis.

(c) Os desafios incluem a falta de interoperabilidade entre sistemas legados e ambientes de nuvem, dificuldades na migração de dados complexos e a necessidade de treinar pessoal para gerenciar os novos ambientes de nuvem. Esses desafios podem ser superados por meio de uma abordagem cuidadosa de planejamento de migração, uso de ferramentas de automação de migração e colaboração estreita entre equipes de desenvolvimento e operações. As implicações éticas envolvem questões de privacidade dos dados, transparência no tratamento das informações dos usuários e responsabilidade pelo armazenamento e processamento de dados sensíveis.

(d) Os desafios incluem a integração de sistemas legados com novas tecnologias de nuvem, a migração de dados sensíveis e a garantia da continuidade dos negócios durante o processo de migração. Esses desafios podem ser superados por meio de uma abordagem iterativa de migração, uso de ferramentas de migração de dados confiáveis e implementação de medidas de segurança e monitoramento. As implicações éticas envolvem questões de privacidade dos dados, transparência no tratamento das informações dos usuários e responsabilidade pelo armazenamento e processamento de dados sensíveis.

(e) Os desafios incluem a falta de expertise em nuvem dentro da organização, a migração de sistemas legados críticos para o negócio e a necessidade de garantir a continuidade dos serviços durante a migração. Esses desafios podem ser superados por meio de treinamento e capacitação da equipe, contratação de consultores especializados em nuvem e implementação de planos de contingência abrangentes. As implicações éticas envolvem questões de privacidade dos dados, transparência no tratamento das informações dos usuários e responsabilidade pelo armazenamento e processamento de dados sensíveis.

18. Em um projeto que utiliza o Desenvolvimento Orientado a Testes (TDD), um desenvolvedor está enfrentando um desafio comum: um teste que ele escreveu para uma nova funcionalidade está falhando, apesar de várias tentativas de ajuste no código. Qual das seguintes abordagens é a mais adequada para resolver esse problema, de acordo com as práticas de TDD?

(a) Ignorar o teste temporariamente e prosseguir com o desenvolvimento de outras funcionalidades.

(b) Refatorar o teste para torná-lo menos restritivo, facilitando a aprovação do

(c) Excluir o teste e escrever um novo teste para uma abordagem diferente da funcionalidade.

(d) Revisar e simplificar o teste para verificar se ele está corretamente alinhado com os requisitos da funcionalidade.
(e) Aumentar a complexidade do código para tentar atender aos critérios do teste atual.

19. No contexto do Desenvolvimento Orientado ao Comportamento (BDD), é fundamental que os cenários escritos sejam compreensíveis por todas as partes interessadas (stakeholders) do projeto, incluindo aqueles sem conhecimento técnico em programação. Considere o seguinte cenário escrito em um estilo BDD:

```
Cenário: Cliente fazendo uma reserva de hotel
Dado que o cliente está na página de reservas de hotel
Quando ele seleciona um quarto do tipo 'Suíte Luxo'
E insere as datas de '2024-07-15' a '2024-07-20'
Então o sistema deve mostrar o custo total da reserva como '$3000'
E não deve permitir a reserva se o quarto não estiver disponível
```

Qual das seguintes afirmações é mais apropriada sobre a qualidade e eficácia deste cenário no contexto de BDD?

(a) O cenário é eficaz, pois descreve especificamente o comportamento esperado do sistema em detalhes técnicos.
(b) O cenário é ineficaz, pois não utiliza uma linguagem natural e acessível para todos os stakeholders.
(c) O cenário é eficaz, mas falta definir o comportamento do sistema em caso de quartos indisponíveis.
(d) O cenário é ineficaz, pois deveria incluir mais detalhes técnicos, como o método de cálculo do custo total.
(e) O cenário é eficaz, pois equilibra informações técnicas com descrições compreensíveis para não técnicos.

20. Em um ambiente de Integração Contínua (CI), uma equipe de desenvolvimento enfrenta um desafio comum: após a introdução de novos commits no repositório, vários testes automatizados começam a falhar. Qual das seguintes estratégias é a mais recomendada para identificar e resolver os problemas o mais rápido possível, mantendo a eficiência do processo de CI?

(a) Ignorar temporariamente os testes falhados para não atrasar o desenvolvimento, revisando-os apenas nas fases de teste mais avançadas.
(b) Executar testes de regressão completos após cada commit para identificar todos os problemas possíveis, mesmo que isso retarde o processo de integração.
(c) Utilizar a técnica de ""bisecting"" para identificar rapidamente o commit específico que causou a falha dos testes, e então corrigir o problema.
(d) Limitar a frequência de integração para reduzir a quantidade de falhas de testes e facilitar o gerenciamento de erros.
(e) Reverter imediatamente todos os commits recentes até que os testes voltem a passar, e então reintegrar os commits um a um.

21. Em um projeto de software, enfrentar desafios relacionados à colaboração em uma equipe diversificada e distribuída geograficamente é comum. Qual estratégia é essencial para gerenciar eficazmente uma equipe de desenvolvimento de software nesse contexto, mantendo alta produtividade e boa comunicação?

(a) Concentrar todas as decisões importantes nas mãos do gerente de projeto para evitar confusões.
(b) Implementar um modelo de trabalho presencial obrigatório para todos os membros da equipe.
(c) Utilizar uma única ferramenta de comunicação para todas as interações da equipe, independentemente das preferências individuais.

Boa sorte!

(d) Estabelecer regras rígidas de trabalho com horários fixos de operação, independentemente dos fusos horários dos membros da equipe.
(e) Promover práticas de comunicação assíncrona e inclusiva, respeitando as diferenças de fuso horário e preferências culturais.

22. Como um gerente de projeto pode efetivamente lidar com o fenômeno conhecido como 'inchaço de requisitos' (scope creep) durante a fase de desenvolvimento de um projeto de software, garantindo que o projeto permaneça fiel aos requisitos originais enquanto acomoda mudanças necessárias?

(a) Aceitando todas as mudanças de requisitos para garantir a satisfação do cliente, mesmo que isso leve a atrasos e aumento de custos.
(b) Estabelecendo um processo rigoroso de revisão de mudanças de requisitos, avaliando o impacto de cada mudança proposta no escopo, orçamento e cronograma do projeto.
(c) Limitando as mudanças de requisitos a uma fase específica do projeto e rejeitando todas as mudanças posteriores.
(d) Delegando a decisão sobre mudanças de requisitos exclusivamente para a equipe de desenvolvimento, sem consulta a outras partes interessadas.
(e) Ignorando as mudanças de requisitos e mantendo-se estritamente ao plano original para evitar complicações.

23. Considere um projeto de software onde múltiplas equipes trabalham em diferentes módulos do mesmo sistema. Qual estratégia deve ser implementada para garantir a alta qualidade do software como um todo, considerando as diferentes partes desenvolvidas por diferentes equipes?

(a) Confiar exclusivamente nos testes realizados por cada equipe individualmente em seus próprios módulos.
(b) Implementar uma política de 'zero bugs' em que nenhum novo código é escrito até que todos os bugs existentes sejam corrigidos.
(c) Definir e aplicar padrões de codificação rigorosos, realizar revisões de código cruzadas e integrar testes automatizados abrangentes e contínuos em todo o projeto.
(d) Delegar a responsabilidade da qualidade apenas para uma equipe de controle de qualidade, separada das equipes de desenvolvimento.
(e) Limitar o escopo do projeto para reduzir a complexidade e a possibilidade de erros.

24. Como a métrica "Mean Time Between Failures" (MTBF) é utilizada no contexto da garantia de qualidade de software, e quais são as implicações de um MTBF baixo em um projeto de software?

(a) MTBF é usado principalmente para medir a eficiência do processo de desenvolvimento de software e um valor baixo indica alta eficiência.
(b) Essa métrica mede o tempo médio de resposta do software e um MTBF baixo implica em um software mais rápido e eficiente.
(c) MTBF é uma métrica que avalia o tempo médio entre falhas de um software, indicando sua confiabilidade. Um MTBF baixo sugere uma maior frequência de falhas e potenciais problemas de estabilidade.
(d) O MTBF mede a compatibilidade do software com diferentes sistemas operacionais e um valor baixo indica pouca compatibilidade.
(e) Esta métrica avalia a facilidade de uso do software; assim, um MTBF baixo indica que o software é fácil de usar e entender.

25. Em um ambiente de desenvolvimento de software que utiliza microsserviços, quais são os desafios associados ao gerenciamento de dados e à consistência transacional entre os serviços, e como esses desafios podem ser efetivamente abordados?

(a) Os dados em microsserviços são sempre mantidos em um único banco de dados centralizado para garantir consistência, sem desafios significativos.

Boa sorte!

(b) Uma abordagem comum é a implementação de um serviço de orquestração que gerencia todas as transações de dados entre os microsserviços para manter a consistência.

(c) Em microsserviços, cada serviço gerencia seu próprio banco de dados, o que pode levar a desafios de consistência transacional. Esses desafios são geralmente abordados através de padrões como Saga ou compensações de transações.

(d) Os microsserviços devem evitar operações que requerem consistência transacional, limitando-se a tarefas que não envolvem o gerenciamento de dados.

(e) A consistência transacional em microsserviços é garantida automaticamente pela natureza independente de cada serviço, sem necessidade de estratégias adicionais.