Teste_TP_2020_Correção

1. A resposta correta é a opção **E)** divisão da solução em partes mais pequenas, desenvolvidas incrementalmente. Essa característica não é própria dos processos sequenciais de desenvolvimento de software, como o modelo Waterfall. O modelo Waterfall segue uma abordagem sequencial e linear, em que cada fase é concluída antes de passar para a próxima. Portanto, não há uma divisão da solução em partes menores que são desenvolvidas incrementalmente ao longo do processo. Em vez disso, o desenvolvimento ocorre em fases distintas, como requisitos, design, implementação, testes e manutenção.

2. A resposta correta é a opção **B)** O V-Model prevê a definição de diferentes tipos de testes que, conceitualmente, se pode emparelhar com as etapas do "Waterfall". O V-Model é um modelo de desenvolvimento de software que enfatiza a integração dos procedimentos de garantia de qualidade com o processo de desenvolvimento. Ele propõe a execução de diferentes tipos de testes em cada etapa correspondente do "Waterfall". Esses tipos de testes são conceitualmente alinhados com as etapas do desenvolvimento, garantindo que os testes sejam planejados e executados em paralelo com as atividades de desenvolvimento. Isso ajuda a garantir que os problemas sejam identificados precocemente e corrigidos antes de avançar para a próxima etapa do processo.

3. A resposta correta é a opção **A)** Existem diferentes classes de teste de software, que variam em quantidade e quanto aos seus objetivos. A metáfora da "pirâmide dos testes" transmite a ideia de que existem diferentes classes de testes de software, que variam em quantidade e em relação aos seus objetivos. A pirâmide sugere uma distribuição de testes em diferentes níveis, onde a base da pirâmide representa os testes de unidade, seguidos pelos testes de integração, e no topo estão os testes de sistema ou testes de interface do usuário. Essa distribuição em forma de pirâmide reflete o conceito de que os testes de unidade devem ser mais numerosos e executados com maior frequência, enquanto os testes de sistema e testes de interface do usuário podem ser menos numerosos, mas têm uma cobertura mais ampla. Isso é consistente com a abordagem de teste em camadas, em que os testes em camadas inferiores são usados como base para os testes em camadas superiores.

4. A resposta correta é a opção **E)** O Product Owner representa os interesses dos stakeholders e tem um papel preponderante na definição de prioridades. No framework Scrum, o Product Owner é responsável por representar os interesses dos stakeholders, como clientes, usuários e patrocinadores. Eles têm um papel fundamental na definição das prioridades do backlog do produto, ou seja, na ordenação das funcionalidades e requisitos a serem desenvolvidos. O Product Owner trabalha em estreita colaboração com a equipe Scrum e o Scrum Master para garantir a entrega de valor ao cliente. O Scrum Master é responsável por facilitar o processo Scrum e ajudar a equipe a entender

e seguir as práticas e princípios do Scrum, enquanto o Scrum Leader não é um papel oficial no Scrum.

5. A resposta correta é a opção **C)** Há uma continuidade; as classes conceituais identificadas no modelo do domínio são refinadas no desenho, acrescentando-se os métodos necessários. Durante o processo de desenvolvimento de software, as classes conceituais identificadas no modelo do domínio na fase de análise servem como uma representação abstrata dos conceitos e entidades do problema. Essas classes fornecem uma compreensão clara do domínio em que o sistema será desenvolvido. À medida que o processo avança para o desenho e implementação, as classes conceituais do modelo do domínio são refinadas, ou seja, são adicionados detalhes e funcionalidades específicas necessárias para a implementação do sistema. Isso inclui a adição de métodos, atributos e comportamentos específicos de programação. Portanto, há uma continuidade entre as classes conceituais identificadas no modelo do domínio e as classes utilizadas no código. O desenho e a implementação são uma evolução das classes do modelo do domínio, onde essas classes são refinadas e aprimoradas com os detalhes necessários para a implementação correta do sistema. As opções A), D) e E) não são corretas, pois as classes do modelo do domínio têm importância e utilidade no processo de desenho e implementação do sistema, e há uma relação direta entre as classes do modelo do domínio e as classes do código resultante. A opção **B)** também está correta, pois as classes conceituais do modelo do domínio inspiram e fornecem direção para a definição das classes no código.

6. A opção correta é a **C)** Um resultado do desenho, em que se desenvolve a interação necessária entre objetos e consequente distribuição de responsabilidade, para implementar o caso de utilização. As realizações dos casos de utilização, também conhecidas como use-case realizations, são resultados do processo de desenho do sistema. Elas descrevem como a interação necessária entre os objetos do sistema é desenvolvida e como a responsabilidade é distribuída entre esses objetos para implementar o caso de utilização. Essas realizações envolvem a definição de interfaces, a especificação de comportamentos e a identificação dos objetos envolvidos na realização do caso de utilização. O desenho do sistema visa transformar as descrições dos casos de utilização em um projeto concreto, determinando como as diferentes partes do sistema interagem para alcançar os resultados desejados.

7. A opção correta é **E)** A classe B é uma sub-classe (especialização) de A. A relação de herança em que a classe B é uma subclasse de A não é uma forma/causa de coupling da classe A para a classe B. A herança representa uma relação de especialização ou extensão, onde a classe B herda atributos e comportamentos da classe A, mas não cria uma dependência direta entre elas. Essa relação hierárquica não é considerada um acoplamento entre as classes. Por outro lado, as opções A), B) e C) representam formas/causas de coupling entre as classes A e B. A dependência ocorre quando a classe A possui um atributo do tipo B, invoca um método específico da classe B ou tem um parâmetro do tipo B em um de seus métodos.

8. A opção correta é **A)** Abstract Factory, Factory Method, Singleton. Os padrões de criação de objetos (creational patterns) são focados na criação de objetos de maneira flexível e eficiente. Alguns exemplos de padrões de criação de objetos incluem: Abstract Factory: Fornece uma interface para criar famílias de objetos relacionados sem especificar suas classes concretas; Factory Method: Define uma interface para criar um objeto, mas permite que as subclasses decidam qual classe concreta instanciar; Singleton: Garante que uma classe tenha apenas uma instância, fornecendo um ponto de acesso global para essa instância.

9. A opção correta é **D)** Indicar a arquitetura necessária para satisfazer as ações dos atores. As narrativas dos casos de utilização podem ajudar a identificar os atores e suas interações com o sistema, o que pode influenciar a arquitetura necessária para suportar essas ações dos atores. Ao descrever as interações entre os atores e o sistema, as narrativas dos casos de utilização fornecem informações relevantes para a definição da arquitetura do sistema.

10. A propriedade caraterística dos casos de utilização (e não das histórias) é:
    D) Descreve como é que o ator se imagina a interagir com o sistema para realizar os seus objetivos.
    Os casos de utilização descrevem as interações entre os atores e o sistema, mostrando como o ator se imagina interagindo com o sistema para alcançar seus objetivos. Eles fornecem uma narrativa que descreve o fluxo de interação e as responsabilidades esperadas do sistema.
    As histórias, por outro lado, são declarações concisas das necessidades de uma persona. Elas são usadas como uma forma de capturar requisitos funcionais em uma linguagem simples e direta. As histórias geralmente são escritas em uma estrutura "Como um [tipo de usuário], eu quero [realizar algo], para [atingir um objetivo]".

11. A ideia apresentada está presente no conceito de:
    D) Requisitos evolutivos.
    A abordagem de requisitos evolutivos sugere que não é necessário desenvolver toda a Especificação de Requisitos de Software (SRS) para o produto completo antes de iniciar o desenvolvimento. Em vez disso, os requisitos para cada incremento ou fase do desenvolvimento podem ser detalhados à medida que são construídos. Isso permite uma abordagem mais flexível e iterativa, em que os requisitos são refinados e ajustados ao longo do tempo à medida que o produto é desenvolvido e o entendimento das necessidades dos usuários evolui.

12. Um exemplo de uma regra do negócio ("Business rule") no contexto de um sistema para pagamento de refeições em uma cafetaria seria:
    B) Para os utilizadores que se apresentam sem cartão, a refeição tem um custo suplementar de 1EUR.
    Essa regra define uma condição específica relacionada ao pagamento das refeições. Ela estabelece que, para os utilizadores que não possuem um cartão válido, há um custo adicional de 1EUR para a refeição. Essa regra do negócio determina como o sistema deve

lidar com essa situação e estabelece uma política de pagamento para os casos em que os utilizadores não possuem o cartão necessário.

13. O conjunto que corresponde a uma coleção de técnicas de levantamento de requisitos estudadas é:
    A) Entrevistas, workshops, questionários, análise documental, observação no local, "focus group".
    Essas são técnicas comumente utilizadas para coletar informações e requisitos dos stakeholders durante o processo de levantamento de requisitos. As entrevistas permitem obter informações diretas dos indivíduos envolvidos, os workshops facilitam a colaboração e a troca de ideias entre os participantes, os questionários são utilizados para coletar dados em larga escala, a análise documental envolve a revisão de documentos existentes, a observação no local permite entender o contexto de uso do sistema e o "focus group" possibilita a discussão em grupo para obter diferentes perspectivas.

14. A opção correta é:
    D) O TDD revela os erros mais cedo, e é mais fácil de localizar e corrigir os erros no software, quanto mais perto do momento em que foram introduzidos.
    Ao adotar práticas de escrita de testes à-priori (TDD), os testes são escritos antes mesmo da implementação do código de produção. Isso significa que os erros são identificados logo no início do processo de desenvolvimento, o que permite uma correção mais rápida e eficiente. Conforme os testes são executados continuamente durante o desenvolvimento, os erros são descobertos em estágios iniciais, facilitando sua localização e correção.
    Essa abordagem evita que os erros se acumulem e se tornem mais complexos de resolver no futuro. Além disso, ao escrever os testes primeiro, os desenvolvedores são obrigados a refletir sobre os requisitos e o comportamento esperado do sistema, o que leva a uma melhor compreensão dos requisitos e a uma implementação mais precisa.
    No médio prazo, o TDD pode acelerar o desenvolvimento, pois a detecção precoce de erros reduz a necessidade de retrabalho extensivo e diminui o tempo gasto em depuração. Além disso, a prática de ter testes automatizados garante que as funcionalidades existentes continuem funcionando conforme o código é modificado ou expandido, o que aumenta a confiança e a estabilidade do sistema como um todo.
    Portanto, a utilização de práticas de escrita de testes à-priori (TDD) pode acelerar o desenvolvimento no médio prazo, devido à detecção precoce de erros e à melhoria da qualidade do software.