Teste_TP_2020_Correção

1. A expressão "entrega contínua" (Continuous Delivery) é usada para caracterizar a prática em que:
   B) O software é construído de forma a poder ser lançado para produção a qualquer momento.
   A entrega contínua envolve a construção do software de modo a permitir que seja lançado em produção de forma rápida e fiável a qualquer momento. Isto é alcançado através da automatização dos processos de construção, teste e implementação, permitindo que as equipas de desenvolvimento entreguem software funcional de forma incremental e constante, com riscos mínimos e sem necessidade de grandes lançamentos ou interrupções. A entrega contínua visa fornecer valor aos utilizadores finais o mais rápido possível e permite que as equipas iterem e melhorem continuamente o software com base no feedback recebido.

2. Uma argumentação possível para demonstrar que a utilização de práticas de escrita de testes prévios (TDD) não só não atrasa, como pode acelerar o desenvolvimento no médio prazo é:
   D) O TDD revela os erros mais cedo, e é mais fácil localizar e corrigir os erros no software quanto mais perto do momento em que foram introduzidos.
   Ao adotar o TDD, os testes são escritos antes mesmo da implementação do código de produção. Esses testes são baseados nos critérios de aceitação das histórias (user stories), o que garante que o software cumpra os requisitos definidos desde o início.
   Ao escrever os testes antes do código, os programadores estão a antecipar possíveis erros e problemas de integração. Isso permite que os erros sejam identificados e corrigidos o mais cedo possível, antes que se propaguem para outras partes do sistema. Consequentemente, a necessidade de fazer correções em fases posteriores do desenvolvimento é reduzida, evitando retrabalho e atrasos.
   Além disso, o TDD promove a melhoria da qualidade do código, uma vez que incentiva a escrita de código modular, coeso e de fácil testabilidade. Isso torna o código mais legível, fácil de manter e menos propenso a erros futuros.
   Embora a escrita dos testes prévios possa exigir algum tempo adicional a curto prazo, essa prática resulta num ciclo de desenvolvimento mais rápido e eficiente a médio prazo. A deteção precoce de erros e a melhoria da qualidade do código contribuem para um desenvolvimento mais rápido e seguro, garantindo a entrega de software de maior qualidade em menos tempo.

3. A metáfora da "pirâmide dos testes" transmite a ideia de que:
   D) Existem diferentes classes de teste de software, que variam em quantidade e quanto aos seus objetivos.
   A metáfora da "pirâmide dos testes" é amplamente utilizada para ilustrar a distribuição e proporção dos diferentes tipos de testes em um projeto de desenvolvimento de software. A forma da pirâmide representa a distribuição hierárquica dos testes, com a

base mais larga representando os testes de unidade, seguidos pelos testes de integração e, por fim, os testes de interface do usuário.

Essa metáfora enfatiza que a maioria dos testes deve ser realizada nas camadas inferiores da pirâmide (testes de unidade e integração), enquanto menos testes devem ser realizados nas camadas superiores (testes de interface do usuário). Isso ocorre porque os testes de unidade e integração são mais granulares, abrangendo componentes individuais e suas interações, o que permite identificar e corrigir problemas mais cedo no ciclo de desenvolvimento.

Por outro lado, os testes de interface do usuário são mais voltados para validar o comportamento geral do sistema e a experiência do usuário. Embora esses testes sejam importantes, eles tendem a ser mais lentos e complexos de executar, portanto, devem ser realizados em menor quantidade em comparação com os testes de unidade e integração.

Em resumo, a metáfora da "pirâmide dos testes" destaca a distribuição de diferentes classes de teste em termos de quantidade e objetivos, com foco na realização de testes mais granulares nas camadas inferiores e menos testes mais abrangentes nas camadas superiores.

4. O V-Model descreve uma forma de integrar os procedimentos de garantia de qualidade com o processo de desenvolvimento.

   B) O V-Model prevê a definição de diferentes tipos de testes que, conceptualmente, se podem emparelhar com as etapas do "Waterfall".

   No V-Model, os diferentes tipos de testes são definidos em paralelo com as etapas do modelo em cascata (Waterfall). Cada etapa do processo de desenvolvimento tem um conjunto correspondente de testes associados a ela. Esses testes são planeados e especificados antecipadamente, para garantir que cada etapa seja devidamente testada antes de avançar para a próxima.

   Ao adotar o V-Model, o planeamento e a execução dos testes são considerados parte integrante de cada etapa do processo de desenvolvimento. Isso ajuda a identificar e corrigir problemas o mais cedo possível, uma vez que os testes são executados em paralelo com as atividades de desenvolvimento.

   No entanto, é importante destacar que o V-Model é uma abordagem mais tradicional e linear de desenvolvimento de software, em que cada etapa é concluída antes de passar para a próxima. Portanto, o modelo pode não ser tão adequado para processos iterativos e ágeis, em que as etapas são mais fluidas e iterativas.

5. O framework de gestão de equipas Scrum prevê a definição de alguns papéis na equipa:

   D) O Product Owner faz a gestão ativa do backlog.

   E) O Product Owner representa os interesses dos stakeholders e tem um papel preponderante na definição de prioridades.

   No Scrum, o Product Owner é o responsável por representar os interesses dos stakeholders, como clientes e utilizadores, e garantir que o produto esteja alinhado com as suas necessidades e expetativas. Ele é responsável por gerir o backlog do produto, que é uma lista ordenada de itens a serem desenvolvidos, e definir as prioridades desses

itens. O Product Owner trabalha em colaboração com a equipa de desenvolvimento para garantir que as funcionalidades mais valiosas sejam implementadas primeiro.
O Scrum Master, por sua vez, não é responsável pela definição da arquitetura da solução (opção A), mas sim pela facilitação do processo Scrum e pela remoção de quaisquer impedimentos que possam afetar a equipa. Ele atua como um líder de serviço, ajudando a equipa a adotar e seguir as práticas ágeis.
A opção B menciona que o Product Owner supervisiona a realização dos testes do software, o que não é uma responsabilidade direta desse papel. Embora o Product Owner tenha interesse na qualidade do produto, a execução dos testes normalmente é da responsabilidade da equipa de desenvolvimento.
A opção C menciona o Scrum Leader, mas esse papel não é reconhecido no Scrum tradicional. É possível que haja alguma variação ou interpretação específica dentro de determinados contextos, mas não é uma parte intrínseca do Scrum em si.

6. A relação entre as classes usadas no modelo do domínio (na análise) e as classes usadas no código (desenho e implementação) é:
   C) As classes conceptuais do modelo do domínio inspiram a definição das classes do código (classes candidatas, nomes, atributos candidatos, etc.).
   No processo de desenvolvimento de software, a análise do domínio envolve a compreensão e modelagem dos conceitos e entidades relevantes para o problema em questão. Essa análise resulta num modelo do domínio, que capta as classes conceptuais, os seus atributos e os seus relacionamentos.
   Ao passar para o desenho e a implementação do sistema, as classes do modelo do domínio servem de inspiração e guia para a definição das classes no código. Embora possam ocorrer algumas adaptações e refinamentos durante esse processo, a estrutura e os conceitos do modelo do domínio são considerados ao criar as classes no código.
   No entanto, é importante destacar que a opção D menciona que as classes conceptuais do modelo do domínio são refinadas no desenho, acrescentando-se os métodos necessários. Essa é uma prática comum, em que as classes conceptuais são enriquecidas com os métodos específicos necessários para a implementação das funcionalidades do sistema. Portanto, a opção D também pode ser considerada correta, dependendo da interpretação e do contexto específico.

7. O modelo comportamental de um sistema inclui as realizações dos casos de utilização (use-case realizations), que são:
   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 são o resultado do desenho detalhado do sistema, em que são identificadas as classes, os objetos e a interação necessária para implementar o caso de utilização. Isso envolve a definição das responsabilidades de cada objeto, a comunicação entre os objetos e a distribuição de tarefas para alcançar o comportamento desejado do sistema durante a execução do caso de utilização.

8. A situação que não é uma forma/causa de coupling da classe A para a classe B em Java é:

E) A classe B é uma subclasse (especializa) de A.

Na opção E, a classe B é uma subclasse de A, o que implica em uma relação de herança. Nesse caso, a classe B herda os membros e comportamentos da classe A, mas não há acoplamento direto da classe A para a classe B. O acoplamento ocorre quando uma classe depende diretamente de outra, como nas opções A, B e C.

9. –

10. As narrativas dos casos de utilização suportam diferentes atividades do processo de desenvolvimento, mas não servem para:

    D) Indicar a arquitetura necessária para satisfazer as ações dos atores.

    As narrativas dos casos de utilização têm várias finalidades no processo de desenvolvimento de software. Elas mostram como um ator usa o sistema para realizar seus objetivos, descrevendo os fluxos de interação e as responsabilidades esperadas do sistema (opção A). Além disso, as narrativas fornecem contexto para entender como uma determinada capacidade do sistema será usada pelo ator no ambiente de produção (opção B). Elas também podem captar requisitos funcionais necessários para suportar as atividades do programador (opção C) e suportar o trabalho de escrita de testes de aceitação, definindo o comportamento esperado do sistema (opção E).

    No entanto, as narrativas dos casos de utilização não são usadas para indicar a arquitetura necessária para satisfazer as ações dos atores (opção D). A arquitetura é uma preocupação separada e geralmente é definida por meio de atividades de design e análise arquitetural.

11. A propriedade característica dos casos de utilização (use cases) e não das histórias (user stories) 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 como os atores (usuários) interagem com o sistema para alcançar seus objetivos. Eles são usados para capturar os requisitos funcionais do sistema, fornecendo uma visão detalhada das interações entre os atores e o sistema. Os casos de utilização descrevem os fluxos de interação e as responsabilidades esperadas do sistema durante a execução de um cenário específico.

    Por outro lado, as histórias (user stories) são uma técnica de especificação ágil que descreve uma funcionalidade do sistema do ponto de vista do usuário. Elas geralmente são declaradas de forma concisa e orientada para as necessidades de uma persona específica (opção B). As histórias são frequentemente usadas para priorizar o trabalho e gerenciar o backlog de um projeto (opção A) e também podem incluir critérios de aceitação (opção E), que explicitam as condições necessárias para a aceitação da implementação daquela história.

    A opção C é aplicável tanto aos casos de utilização quanto às histórias, pois ambos podem se beneficiar do acesso fácil aos especialistas do domínio de aplicação para obter detalhes adicionais ou esclarecer requisitos.

12. A ideia de que "Não é necessário desenvolver a SRS para todo o produto antes de iniciar o desenvolvimento; os requisitos para cada incremento devem ser detalhados quando se vai construí-lo" está presente no conceito de:
    D) Requisitos evolutivos.
    Requisitos evolutivos são aqueles que podem ser refinados e atualizados ao longo do ciclo de desenvolvimento do software. Em vez de tentar definir todos os requisitos antecipadamente, a abordagem evolutiva permite que os requisitos sejam desenvolvidos de forma iterativa e incremental, à medida que o sistema é construído e a compreensão dos requisitos aumenta.
    Essa abordagem é consistente com a ideia de que os requisitos para cada incremento do produto devem ser detalhados quando se vai construí-lo, em vez de tentar definir todos os requisitos de uma vez na especificação completa (SRS - Software Requirements Specification).

13. Um exemplo de 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 processo de pagamento de refeições na cafetaria. Ela estabelece que, caso um utilizador não apresente um cartão para efetuar o pagamento, haverá um custo suplementar de 1EUR para a refeição.
    As regras de negócio são diretrizes ou restrições que definem como o negócio deve operar. Elas são importantes para garantir a consistência e o correto funcionamento do sistema em relação aos processos e requisitos do negócio.
    As opções A, C, D e E descrevem características técnicas ou requisitos de sistema, enquanto a opção B é uma regra específica relacionada ao negócio da cafetaria.

14. 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 técnicas são comumente utilizadas no processo de levantamento de requisitos para coletar informações e entender as necessidades dos stakeholders. As entrevistas são realizadas com indivíduos-chave para obter insights diretos, os workshops promovem a colaboração e o envolvimento de múltiplos participantes, os questionários permitem coletar informações de forma estruturada, a análise documental envolve a revisão de documentos existentes, a observação no local permite obter conhecimento sobre o ambiente de trabalho real e o "focus group" reúne um grupo de pessoas para discussão e geração de ideias.
    As opções B, C e D incluem outras técnicas e práticas relacionadas a diferentes aspectos do processo de desenvolvimento de software, mas não representam um conjunto de técnicas de levantamento de requisitos específicas.

15. Com base nas opções fornecidas, a resposta correta é:

A) Quando o dentista cria um diagnóstico, pode consultar exames de imagiologia que existam.

Essa opção descreve corretamente a relação entre o dentista e a consulta de exames de imagiologia no Diagrama 1. O dentista tem a opção de consultar exames de imagiologia que existam ao criar um diagnóstico. Isso indica que a consulta dos exames não é obrigatória, mas é uma possibilidade disponível para o dentista durante o processo de diagnóstico.

As opções B e C são muito restritivas em suas afirmações, exigindo que o dentista sempre consulte os exames de imagiologia ao criar um diagnóstico ou exigindo que o dentista insira um diagnóstico sempre que um novo exame é preparado.

A opção D não é relevante para a questão, pois se refere a uma seção específica que deve ser retirada do diagrama.

A opção E também não é relevante para a questão, pois não se discute a associação do dentista com o caso "Consultar exames" no enunciado.

16. Diagramas do gênero do Diagrama 1, que são cenários de utilização associados a um sistema de informação, são usados ao longo do SDLC (Ciclo de Vida de Desenvolvimento de Sistemas) de várias maneiras. As opções corretas que descrevem o uso desses diagramas são:

    A) Podem ser usados para detalhar/suplementar os conceitos identificados no modelo do domínio.

    B) Podem ser detalhados/suplementados com diagramas de sequência.

    A opção A é correta porque os diagramas de cenários de utilização podem ser usados para aprofundar e complementar os conceitos identificados no modelo do domínio. Eles fornecem uma visão mais detalhada e específica das interações e fluxos de eventos entre atores e o sistema.

    A opção B também é correta, pois os diagramas de sequência podem ser usados para detalhar e suplementar os cenários de utilização. Os diagramas de sequência mostram a interação entre objetos em um determinado cenário, fornecendo uma representação visual da ordem das mensagens trocadas entre os objetos.

17. Em relação ao Diagrama 2, a afirmação correta é:

    B) O serviço “ProductSearch” é implementado pelo subsistema “WebStore”.

18. A opção correta é a seguinte:

    C) É uma vista de arquitetura que mostra a forma como a solução está dividida em componentes, e as interfaces entre eles.

    O Diagrama 2 é um exemplo de um diagrama de componentes, que faz parte da visão de arquitetura de um sistema. Ele representa a estrutura interna do sistema, identificando os componentes e as interfaces entre eles. Essa visão ajuda a compreender a organização e a interação dos componentes na solução.

19. A opção correta é:

    D) A execução das invocações dentro do fragmento "alt" é condicional.

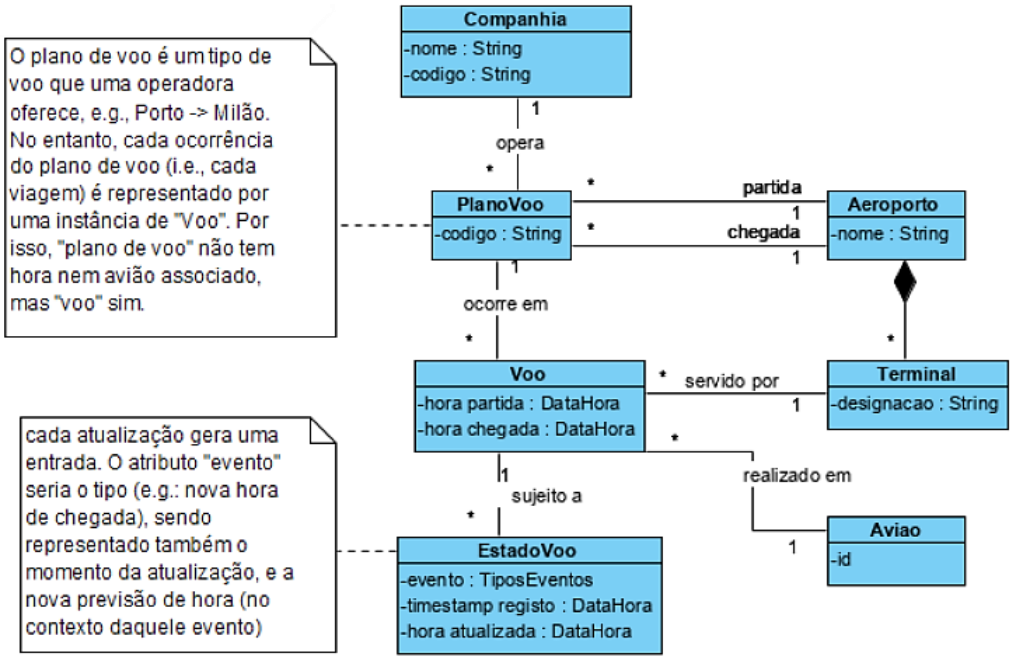
No Diagrama 3, é possível observar o uso do fragmento "alt", que representa uma estrutura condicional. A condição especificada determina qual trecho de código será executado. Portanto, a execução das invocações dentro desse fragmento é condicional, dependendo da condição especificada no diagrama.

20. Os diagramas do tipo do Diagrama 3 (diagrama de sequência) são utilizados num projeto principalmente para:
    A) O analista prepara esses diagramas para mostrar a realização dos casos de utilização. Os diagramas de sequência ajudam a ilustrar a interação entre os objetos e a ordem das mensagens trocadas durante a realização de um caso de uso específico. Eles permitem ao analista visualizar como os objetos colaboram entre si para alcançar os objetivos do caso de uso.
    Embora os outros itens mencionados também tenham alguma relação com os diagramas de sequência, a resposta mais direta e específica é a opção A, pois os diagramas de sequência são particularmente úteis na análise dos casos de uso e na representação da interação entre os objetos durante a execução desses casos de uso.

21. Existem duas abordagens principais na definição de requisitos: centrada no produto e centrada no utilizador. A abordagem centrada no produto enfoca as características do produto, enquanto a abordagem centrada no utilizador prioriza as necessidades e experiências dos utilizadores.
    Exemplos:
    - abordagem centrada no produto: uma aplicação de edição de fotografias com filtros e ferramentas avançadas;
    - abordagem centrada no utilizador: uma aplicação de fitness com monitorização de atividades e planos de treino personalizados.

22. O OpenUP é um processo ágil que identifica quatro etapas principais com objetivos e marcos correspondentes. Essas etapas são adaptadas às necessidades do projeto e fornecem uma abordagem iterativa e incremental para o desenvolvimento de software. O OpenUP enfatiza a colaboração entre a equipe, a entrega de valor em curtos períodos e a capacidade de adaptação às mudanças. Ele compartilha princípios semelhantes aos métodos ágeis, como o foco no cliente, a flexibilidade e a entrega contínua de software funcional. Portanto, o OpenUP é considerado um processo ágil, pois promove os valores e práticas ágeis no desenvolvimento de software.

23.


O plano de voo é um tipo de voo que uma operadora oferece, e.g., Porto -> Milão. No entanto, cada ocorrência do plano de voo (i.e., cada viagem) é representado por uma instância de "Voo". Por isso, "plano de voo" não tem hora nem avião associado, mas "voo" sim.
Companhia
-nome : String
-codigo : String
1
opera
*
PlanoVoo
-codigo : String
*
partida
1
*
chegada
1
Aeroporto
-nome : String
1
ocorre em
*
*
Voo
-hora partida : DataHora
-hora chegada : DataHora
*
servido por
1
Terminal
-designacao : String
*
realizado em
1
Aviao
-id
cada atualização gera uma entrada. O atributo "evento" seria o tipo (e.g.: nova hora de chegada), sendo representado também o momento da atualização, e a nova previsão de hora (no contexto daquele evento)
1
sujeito a
*
EstadoVoo
-evento : TiposEventos
-timestamp registo : DataHora
-hora atualizada : DataHora