-- CONHECIMENTOS ESPECÍFICOS --

Julgue os próximos itens, a respeito de desenvolvimento de sistemas.

51 Uma forma de realizar desenvolvimento *mobile* utilizando-se JavaScript no *frontend* e COBOL no *backend* é definir o HTML básico na *data division* do COBOL e gerar na *procedure division* o conteúdo dinâmico que poderá interagir com o código JavaScript.

JUSTIFICATIVA - Certo. Em um sistema que usa COBOL no *backend* e JavaScript no *frontend*, a geração de HTML pode ocorrer com o COBOL gerando HTML básico, que é, então, aprimorado e manipulado pelo JavaScript no navegador. Essa é uma forma válida de realizar o desenvolvimento *mobile* híbrido e em camadas, utilizando-se JavaScript no navegador e o COBOL no servidor. Assim, na *data division* do programa COBOL, é realizada a definição do conteúdo HTML básico. A *data division* do COBOL inclui a *working-storage section*, onde se definem as variáveis e as estruturas de dados que são usadas internamente pelo programa. Em seguida, na *procedure division*, é escrito o conteúdo dinâmico deste HTML em uma variável de saída e que poderá interagir com código JavaScript.

52 A classe `R.java`, quando utilizada no desenvolvimento de aplicativos para o sistema Android, é gerada de forma automática durante a transformação do código-fonte Java em um artefato executável e não deve ser manipulada diretamente pelo programador.

JUSTIFICATIVA - Certo. No desenvolvimento de aplicativos para Android, o Java dispõe dos recursos da classe `R.java`. A classe `R` é responsável por fazer a ligação entre o arquivo XML, localizado no diretório `/res`, e o código da aplicação. Essa classe é gerada automaticamente pelo *plugin* da AAPT, quando a *build* é executada (*build* é o código fonte transformado em artefato executável). Não se devem realizar alterações diretamente nessa classe: a classe `R` é gerada automaticamente pelo *plug-in* da AAPT e somente a AAPT pode realizar alterações nessa classe. A classe `R` representa os recursos do projeto Android, que correspondem a todos os conteúdos da pasta `/res`.

53 Na execução do trecho de código a seguir, escrito em JavaScript, o resultado lógico da operação `x == 7` será falso.

```
<script>
let x = "7";
document.getElementById("teste").innerHTML = (x == 7);
</script>
```

JUSTIFICATIVA - Errado. O resultado deve ser verdadeiro lógico neste programa JavaScript, pois a comparação foi feita com o operador de igualdade não estrita (`==`). Nesse caso, o operador `==` realiza uma comparação com coerção de tipo. Quando uma *string* é comparada com um número por meio do operador `==`, o JavaScript tenta converter a *string* em um número. Como a *string* `"7"` convertida em número será igual a `7`, a comparação retorna o valor verdadeiro lógico. É importante notar que, se fosse usado o operador de igualdade estrita `===`, o resultado seria falso lógico, pois este operador não realiza coerção de tipo e compara tanto o valor quanto o tipo.

Julgue os itens que se seguem, referentes a *clean code* e à ferramenta SonarQube.

54 O SonarQube tem uma abordagem embasada no princípio de que o código da base em produção deve ser formatado e revisado para garantir que se usem as interfaces de programação e os recursos de linguagem apropriados.

JUSTIFICATIVA - Errado. O SonarQube tem uma abordagem baseada no princípio de que de que o novo código (código adicionado ou modificado recentemente pelo usuário) deve estar em conformidade com os padrões de qualidade do SonarQube (servidor, nuvem). A solução Sonar implementa o *clean as you code*, avisando sempre que problemas forem detectados no novo código.

55 Na produção de um código, a adoção dos termos primário e secundário em vez de mestre e escravo, respectivamente, é uma maneira de produzir um código em observância ao atributo da responsabilidade do *clean code*.

JUSTIFICATIVA - Certo. O atributo da responsabilidade estabelece que o código deve levar em conta suas obrigações éticas sobre dados, bem como normas sociais.

No que se refere à arquitetura de *software*, julgue os itens subsecutivos.

56 A arquitetura de aplicações para ambiente *web* denominada *web-queue-worker* constitui-se, entre outros componentes, de um *frontend web* para interação com clientes, uma fila de mensagens para comunicação e um trabalhador para processar tarefas complexas ou demoradas.

JUSTIFICATIVA - Certo. Os principais componentes da arquitetura *web-queue-worker* são: um *frontend* da Web que atende às solicitações do cliente e um trabalhador que executa tarefas que consomem muitos recursos, fluxos de trabalho de longa execução ou trabalhos em lote. O *frontend* da Web comunica com a função de trabalho por meio de uma fila de mensagens.

57 Na arquitetura orientada a serviços (SOA), a WSDL (*web services description language*) descreve a maneira pela qual a lógica de negócios dos serviços *web* deve ser acessada.

JUSTIFICATIVA - Certo. WSDL é uma linguagem de descrição que define a interface pública dos serviços *web* e descreve como a lógica de negócio desses serviços deve ser acessada.

58 Na arquitetura orientada a objetos, o polimorfismo é baseado na escolha do método específico a ser realmente chamado no momento da execução da compilação.

JUSTIFICATIVA - Errado. O conceito apresentado é o de acoplamento dinâmico, ou tardio, que é uma técnica que retarda a identificação do tipo do objeto até o tempo de execução. Como tal, o método específico que é realmente chamado não é escolhido pelo sistema orientado a objeto até o sistema estar sendo executado.

Julgue os itens a seguir, relativos a tecnologias e padrões para o desenvolvimento *web*, intercâmbio de dados e comunicação entre sistemas.

59 Para otimizar a representação de serviços em um registro UDDI e melhorar a eficiência na troca de dados, pode ser utilizado o JSON em vez do XML tradicional, pois o JSON é capaz de reduzir o tamanho dos dados transmitidos e simplificar o processo de *parsing*.

JUSTIFICATIVA - Certo. O JSON (JavaScript Object Notation) pode ser usado para otimizar a representação de serviços em um registro UDDI (Universal Description, Discovery and Integration), melhorando a eficiência na troca de dados. Ao utilizar JSON em vez do XML tradicional, é possível reduzir o tamanho dos dados transmitidos e simplificar o processo de *parsing*, resultando em comunicações mais rápidas e eficientes entre os sistemas que interagem com o registro UDDI. *Parsing* é o processo de analisar uma sequência de símbolos (como texto ou dados estruturados) de acordo com as regras de uma gramática formal, transformando-a em uma estrutura de dados que pode ser facilmente manipulada por um programa de computador.

60 O princípio *cacheable* do padrão REST estabelece que as respostas às solicitações são gerenciadas pelo servidor, que decide acerca do armazenamento em *cache* dos dados, otimizando o desempenho do cliente.
JUSTIFICATIVA - Errado. O princípio *cacheable* do padrão REST estabelece que as respostas às solicitações devem indicar explícita ou implicitamente se os dados podem ser armazenados em cache, sendo este gerenciado pelo cliente para otimizar o desempenho e reduzir a carga no servidor.

Considerando conceitos e aplicações do DevOps no contexto das metodologias ágeis, julgue os itens a seguir.

61 A cultura da escalabilidade horizontal e vertical das aplicações insere-se na adoção de práticas seguras no DevOps, pois a detecção de vulnerabilidades em ambientes ágeis de entrega contínua, como o Kanban e o XP, é aprimorada.
JUSTIFICATIVA - Errado. O item mistura incorretamente conceitos de escalabilidade horizontal e vertical com práticas de segurança e detecção de vulnerabilidades, o que não é preciso no contexto de DevOps e segurança de *software*. A escalabilidade e a segurança são aspectos importantes, mas distintos, no desenvolvimento e operação de sistemas, e não devem ser confundidos da maneira apresentada no item. Embora a escalabilidade seja um aspecto importante no DevOps, ela não está diretamente relacionada às práticas de segurança e detecção de vulnerabilidades. A escalabilidade se refere à capacidade de um sistema lidar com aumento de carga, enquanto as práticas de segurança focam na proteção contra vulnerabilidades e ameaças.

62 A implementação de uma *pipeline* de CI/CD no DevOps otimiza o processo de *sprint review* em um ambiente Scrum, pois resulta em uma capacidade aprimorada de demonstrar funcionalidades em ambiente próximo ao de produção.
JUSTIFICATIVA - Certo. A implementação de uma *pipeline* de CI/CD no DevOps pode otimizar o processo de *sprint review* em um ambiente Scrum ao automatizar a integração, testes e implantação do código. Isso resulta em *builds* mais frequentes e confiáveis, permitindo que a equipe apresente incrementos de produto totalmente testados e potencialmente entregáveis durante a revisão. Consequentemente, há uma redução no tempo de *feedback*, maior visibilidade do progresso real do *sprint* e uma capacidade aprimorada de demonstrar funcionalidades em um ambiente próximo ao de produção, alinhando-se com o princípio ágil de entrega frequente de *software* funcional.

Julgue os itens seguintes, a respeito de GIT e testes de *software*.

63 A eficiência e a escalabilidade dos *bots* RPA (*robotic process automation*) em diferentes cargas de trabalho são valiosas para os testes de exceção.
JUSTIFICATIVA - Errado. A eficiência e a escalabilidade dos *bots* RPA sob diferentes cargas de trabalho são valiosas para os testes de desempenho.

64 A integração do TDD ao fluxo de trabalho do GIT, por meio da implementação de *hooks* de pré-*commit* e pré-*push*, em associação com práticas de integração contínua, propicia uma abordagem proativa para a manutenção da integridade do código em ambientes de desenvolvimento distribuído com múltiplos *branches*.
JUSTIFICATIVA - Certo. O TDD (*test-driven development*) pode ser integrado eficientemente ao fluxo de trabalho do GIT para garantir a qualidade do código em projetos com múltiplos *branches* e integrações frequentes por meio da implementação de *hooks* de pré-*commit* e pré-*push* que executam automaticamente os testes antes de cada *commit* ou *push*. Isso garante que apenas código testado seja integrado aos *branches*, mantendo a qualidade do código mesmo em projetos com múltiplos *branches* e integrações frequentes.

65 O teste de integração é o mais adequado para verificar se as alterações em um *branch* do GIT não afetaram negativamente as funcionalidades existentes do sistema, no sentido de evitar a introdução de novos *bugs* ou a reintrodução de problemas antigos.
JUSTIFICATIVA - Errado. O tipo de teste mais adequado para essa situação (verificar se as alterações feitas em uma *branch* do GIT, ramificação da *main* de um repositório, não afetaram negativamente as funcionalidades existentes do sistema) é o teste de regressão, que é utilizado para verificar se as mudanças recentes no código não introduziram novos bugs ou reintroduziram problemas antigos em funcionalidades que já estavam funcionando corretamente.

Julgue os itens a seguir, a respeito de reúso de *software* e de API (*application programming interface*).

66 As chaves de API verificam se os usuários são de fato quem afirmam ser e se têm os direitos de acesso para uma chamada de API específica.
JUSTIFICATIVA - Errado. As chaves de API não verificam a identidade dos usuários nem conferem direitos específicos de acesso, mas fornecem uma camada básica de autenticação, geralmente vinculada ao aplicativo que realiza a chamada, não ao usuário. Elas são utilizadas principalmente para autorizar o acesso à API em si, mas não para verificar quem o usuário é ou que permissões específicas ele possui. Esse papel pertence aos *tokens* de autenticação, que contêm informações de identidade e permissões do usuário para verificar se ele tem acesso aos recursos específicos da API.

67 No nível de abstração, o reúso ocorre por aproveitamento de conhecimentos, como padrões de projeto, arquiteturas e outras práticas bem-sucedidas, que orientam o desenvolvimento de novos *softwares*.
JUSTIFICATIVA - Certo. O reúso de *software* pode ocorrer em diferentes níveis, como no de abstração, no qual o reúso não é feito diretamente com o *software*, mas por meio do conhecimento de abstrações bem-sucedidas aplicadas ao *design*. Padrões de projeto e arquitetura são exemplos que representam esse conhecimento abstrato, permitindo que soluções eficazes sejam reutilizadas no desenvolvimento de novos sistemas.

Julgue o item seguinte a respeito do uso de cenários como técnica de elicitação de requisitos.

68 Um cenário começa com um esboço da interação, sendo este detalhado na elicitação de requisitos para incluir expectativas iniciais, fluxo normal, possíveis erros, atividades paralelas e o estado final do sistema.
JUSTIFICATIVA - Certo. Os cenários podem ser particularmente úteis para adicionar detalhes a uma descrição geral de requisitos. Referem-se a descrições de exemplos de sessões de interação. Cada cenário geralmente cobre um pequeno número de interações possíveis. Diferentes cenários são desenvolvidos e oferecem diversos tipos de informação em variados níveis de detalhamento sobre o sistema. Um cenário começa com um esboço da interação. Durante o processo de elicitaçâo, são adicionados detalhes ao esboço, para criar uma descrição completa dessa interação. Em sua forma mais geral, um cenário pode incluir uma descrição do que o sistema e os usuários esperam quando o cenário se iniciar, uma descrição do fluxo normal de eventos no cenário, uma descrição do que pode dar errado e como isso é tratado, informações sobre outras atividades que podem acontecer ao mesmo tempo, uma descrição do estado do sistema quando o cenário acaba.

```
return (
  <img
```

```
    className="avatar"
    src="user.imageUrl"
  />
);
```

Considerando o código precedente, desenvolvido em React, julgue o item que se segue.

69 Após ser renderizado, o componente `<img/>` irá exibir uma imagem dinâmica, usando o valor da propriedade `user.imageUrl` como o caminho do atributo `src`.

JUSTIFICATIVA - Errado. No trecho `src="user.imageUrl"`, a expressão passa a *string* literal `"user.imageUrl"` como o valor do atributo `src`, e não o valor da variável `user.imageUrl`. Para que o valor da propriedade `imageUrl` do objeto `user` seja usado como caminho da imagem, seria necessário usar chaves `{ }`: `src={user.imageUrl}`. Somente dessa forma o valor da variável JavaScript será lido e passado corretamente.

```
<!DOCTYPE html>
<html>
   <head>
      <style>
         box {
         width: 150px;
         height: 150px;
         background-color: pink;
         opacity: 1;
         transform: translateY(0);
         transition: all 0.9s ease;
         }
      </style>
   </head>
   <body>
      <h1>Demo of @starting-style</h1>
      <div class="box">HELLO!</div>
   </body>
</html>
```

Em relação ao código HTML e CSS precedente, julgue o próximo item.

70 Ao ser executado, o código exibe o texto `HELLO!` em uma caixa rosa de 150 px, na posição definida e com um efeito de animação do tipo transição suave.

JUSTIFICATIVA - Errado. Esse código exibe uma caixa rosa de 150px por 150 px contendo o texto `HELLO!`. A caixa está totalmente visível (`opacity: 1`) e na posição padrão (`translateY(0)`). Embora a propriedade `transition` esteja definida para aplicar uma transição suave, na execução do código não haverá nenhum efeito de animação, pois não há mudanças de estilo ou classes adicionais para iniciar uma transição.

```
import { createApp } from 'vue';
const app = createApp({
  data() {
    return {
      message: 'Olá, Vue!'
    };
  },
  template: '<h1> message</h1>'
});

app.mount('#app');
```

Com base no trecho de código precedente, referente ao *framework* Vue.js, julgue o item a seguir.

71 A execução do código criará uma aplicação Vue, que renderiza a mensagem `Olá, Vue!` no elemento com `id="app"`, exibindo o conteúdo do *template* `<h1> message </h1>` na página.

JUSTIFICATIVA - Errado. A execução do trecho de código em questão apresentará um erro na renderização de `message` no *template*. A sintaxe está incorreta, pois a variável `message` deveria ser exibida entre `{{ }}` para que a interpolação fosse possível. O *template* correto seria o seguinte:
`template: '<h1>{{ message }}</h1>`

Julgue os itens subsecutivos, relativos aos protocolos HTTPS e SSL/TLS.

72 Sítios HTTPS requerem a emissão de um certificado SSL/TLS por uma autoridade certificadora (CA), o qual é compartilhado com o navegador para estabelecer confiança e permitir a troca segura de dados criptografados.

JUSTIFICATIVA - Certo. Para que um sítio seja acessado via HTTPS, é necessária a emissão de certificado SSL/TLS por uma autoridade certificadora (CA) confiável, como a Let's Encrypt, DigiCert ou Comodo. A CA valida a identidade do sítio antes de emitir o certificado para garantir que ele seja seguro e autêntico.

73 O *handshake* TLS é mais lento e complexo que o *handshake* SSL, que, por sua vez, possui menos etapas, estabelecendo uma conexão mais rápida que aquele.

JUSTIFICATIVA - Errado. Os conceitos estão trocados. O *handshake* SSL é mais lento e complexo, enquanto o *handshake* TLS foi otimizado para ser mais rápido e eficiente, com menos etapas envolvidas.

No que se refere a UX (*user experience*), *blockchain*, *design* de *software* e gerenciador de transições distribuídas, julgue os itens subsequentes.

74 No planejamento de interação para aplicações *web*, o envio de *mockups* próximos aos *breakpoints* contribui para a orientação de mudanças no *layout* e para a precisão na UI.

JUSTIFICATIVA - Certo. No planejamento da interação para aplicações *web*, é interessante que se enviem modelos com os limites bem definidos após cada *breakpoint*, para que o desenvolvedor possa entender com precisão os pontos limites do *design*.

75 As redes *blockchain* do tipo consórcio são formadas por organizações pré-selecionadas, as quais são responsáveis por manter a *blockchain* e gerenciar o acesso aos dados.

JUSTIFICATIVA - Certo. A rede de *blockchain* consórcio é gerida por um grupo de organizações previamente selecionadas, que compartilham a responsabilidade de manutenção da *blockchain* e controle do acesso aos dados. Os setores nos quais as organizações têm objetivos comuns e que são beneficiados pela responsabilidade compartilhada normalmente preferem as redes *blockchain* consórcio. Por exemplo, a Global Shipping Business Network Consortium é um consórcio de *blockchains* sem fins lucrativos, que pretende digitalizar a indústria de transportadoras e aumentar a colaboração entre os operadores da indústria naval.

76 Nas transações distribuídas, o processo *two-phase commit* verifica a prontidão dos gerenciadores de recursos antes de confirmar ou reverter uma transação em caso de falha.

JUSTIFICATIVA - Certo. Em transações distribuídas, um gerenciador de transações coordena e gerencia a transação entre dois ou mais gerenciadores de recursos. O processo *two-phase commit* ocorre quando há mais de um gerenciador de recursos participando da transação. No processo *two-phase commit*, o gerenciador de transações envia uma chamada de preparação para verificar se todos os gerenciadores de recursos estão preparados para confirmar. Quando recebe o reconhecimento de todos os

gerenciadores de recursos, a chamada de confirmação é emitida.

77 Em *design* de *software* na arquitetura hexagonal, os adaptadores estão conectados diretamente ao núcleo do domínio, pois não há necessidade de portas ou interfaces intermediárias para comunicação.
JUSTIFICATIVA - Errado. Na arquitetura hexagonal, os adaptadores não estão conectados diretamente ao núcleo do domínio. Eles dependem das portas para garantir a separação entre o domínio e as camadas externas. Sem as portas, a arquitetura hexagonal perde seu propósito principal de isolamento e independência da lógica de negócios.

A respeito de inteligência artificial, de tipos de análise de dados e de *Big Data*, julgue os itens que se seguem.

78 A veracidade em *Big Data* reflete o desafio de se assegurar a qualidade e precisão de dados frequentemente confusos, ruidosos e propensos a erros que comprometem sua confiabilidade.
JUSTIFICATIVA - Certo. Em *Big Data*, a veracidade é um pilar crítico, pois ajuda a garantir que os dados usados sejam verdadeiros, relevantes e úteis. Esse conceito não se refere apenas à coleta, mas também aos processos de limpeza, validação e análise dos dados para assegurar que as decisões baseadas neles sejam sólidas.

79 Modelos discriminativos classificam dados conhecidos em categorias, enquanto modelos generativos preveem características completas a partir de um rótulo, explorando probabilidades conjuntas.
JUSTIFICATIVA - Certo. Os modelos discriminativos, ou focados na classificação de pontos de dados, tentam determinar a relação entre fatores conhecidos e desconhecidos, enquanto os modelos generativos tentam prever recursos a partir de determinado rótulo por meio de cálculos avançados de probabilidade.

80 A análise descritiva utiliza estatísticas e projeções para recomendar ações estratégicas, auxiliando a tomada de decisões e o alcance de melhores resultados.
JUSTIFICATIVA - Errado. A análise descritiva não tem como objetivo recomendar ações estratégicas ou realizar projeções. Ela se concentra em resumir e interpretar dados históricos para entender o que aconteceu no passado.

Em relação a configurações básicas de MS Windows Server e Linux, julgue os itens seguintes.

81 Processos do Linux são identificados por seus respectivos identificadores únicos denominados PID, sendo o canal de comunicação entre dois processos conhecido como *pipe*.
JUSTIFICATIVA - Certo. PID é o identificador único de um processo no Linux. É possível criar um canal entre dois processos, denominado *pipe*, no qual um processo pode escrever um fluxo de *bytes*. para o outro ler.

82 Caso o recurso de *logon* único do protocolo Kerberos seja utilizado na autenticação baseada em domínio, a parte em uma extremidade da conexão de rede verificará se a parte na outra extremidade é a entidade que diz ser.
JUSTIFICATIVA - Errado. O recurso apresentado no item é o de autenticação mútua; nesse recurso, uma parte na extremidade de uma conexão de rede pode verificar se a parte na outra extremidade é a entidade que ela diz ser.

83 No MS Windows Server, a zona DNS é uma parte específica de um `namespace` DNS, em que o armazenamento, por padrão, está localizado na pasta `%windir%\System32\Dns` no servidor.
JUSTIFICATIVA - Certo. No Windows Server, uma zona DNS é a parte específica de um `namespace` DNS é hospedada em um servidor DNS. Essa zona contém registros de recursos e o servidor DNS responde a consultas de registros nesse `namespace`. Por padrão, o arquivo de zona primária é denominado zone_name.dns e está localizado na pasta `%windir%\System32\Dns` no servidor.

84 No MS Windows Server, o DHCP oferece uma configuração do endereço IP confiável, que minimiza erros de configuração manual de IP, e uma configuração centralizada e automatizada.
JUSTIFICATIVA - Certo. O DHCP é um protocolo de cliente/servidor que fornece automaticamente um *host* IP com seu endereço IP e outras informações de configuração relacionadas, como a máscara de sub-rede e o *gateway* padrão. OS RFCs 2131 e 2132 definem o DHCP como um padrão IETF (Internet Engineering Task Force) com base no BOOTP (Bootstrap Protocol), um protocolo com o qual o DHCP compartilha muitos detalhes de implementação. O DHCP permite que os *hosts* obtenham as informações de configuração TCP/IP necessárias de um servidor DHCP.

85 Uma floresta do *Active Directory* é uma coleção de uma ou mais unidades organizacionais (UOs), nas quais os domínios na mesma floresta são vinculados manualmente a relações de confiança bidirecionais e transitivas.
JUSTIFICATIVA - Errado. Uma floresta é uma coleção de um ou mais domínios do *Active Directory* que compartilham uma estrutura lógica comum, um esquema de diretório (definições de classe e atributo), uma configuração de diretório (informações de sítio e replicação) e um catálogo global (recursos de pesquisa em toda a floresta). Os domínios na mesma floresta são automaticamente vinculados a relações de confiança bidirecionais e transitivas.

Julgue os próximos itens, relativos a gerenciamento de memória, conceitos de processo e *threads* e LDAP.

86 O LDAP é utilizado para o acesso, através de uma rede, a serviços de diretório, cuja função é compartilhar informações dos usuários de maneira centralizada e organizada de forma hierárquica.
JUSTIFICATIVA - Certo. O LDAP (Lightweight Directory Access Protocol) é um protocolo utilizado para acesso a serviços de diretório através de uma rede, conforme descrito na RFC 1777. A principal função dos serviços de diretório consiste no compartilhamento de informações dos usuários de maneira centralizada, organizada de forma hierárquica e separada em departamentos, equipes, funcionários, etc. Com o LDAP é possível utilizar um único usuário e uma única senha para diversos sistemas, sem a necessidade de ser criado um cadastro para cada aplicação.

87 No gerenciamento de memória *swap*, cada programa tem seu próprio espaço de endereçamento, o qual é dividido em blocos denominados páginas, com uma série de endereços adjacentes.
JUSTIFICATIVA - Errado. O conceito apresentado no item é o de memória virtual.

88 *Threads* são usadas para agrupar recursos; e processos são entidades escalonadas para execução na CPU.
JUSTIFICATIVA - Errado. O item está errado pois os conceitos estão trocados.

89 Quando um computador usa a memória virtual, os endereços virtuais não vão diretamente para o barramento da memória, mas para uma unidade de gerenciamento de memória, que mapeia os endereços virtuais em endereços de memória

física.
JUSTIFICATIVA - Certo. A maioria dos sistemas de memória virtual usa uma técnica chamada de paginação. Quando a memória virtual é usada, os endereços virtuais não vão diretamente para o barramento da memória. Em vez disso, eles vão para uma MMU (*memory management unit* — unidade de gerenciamento de memória) que mapeia os endereços virtuais em endereços de memória física.

90 Cada processo tem um contador de programa, que controla a instrução a ser executada em seguida, e registradores, que armazenam suas variáveis de trabalho atuais.
JUSTIFICATIVA - Errado. O conceito apresentado é o de *thread*.

No que se refere a virtualização de servidores, contêineres e computação em nuvem, julgue os próximos itens.

91 No modelo de serviço de nuvem IaaS, são oferecidos ao usuário recursos computacionais como capacidade de processamento, armazenamento e redes, sobre os quais o usuário pode instalar e executar qualquer tipo de *software*, como sistemas operacionais e aplicações.
JUSTIFICATIVA - Certo. No modelo de serviço de nuvem IaaS, o serviço oferecido ao usuário é um conjunto de recursos computacionais básicos, tais como capacidade de processamento, armazenamento e redes, sobre os quais pode ser instalado e executado qualquer tipo de *software*, incluindo sistemas operacionais e aplicações. Neste caso, embora a infraestrutura de nuvem seja invisível para o usuário, ele pode controlar completamente os sistemas operacionais, espaço de armazenamento e aplicações alocados por ele.

92 *Workload* consiste no dimensionamento da infraestrutura virtualizada, consideradas a velocidade de entrada/saída (I/O), a velocidade da CPU, a capacidade de paralelismo e a eficiência do sistema operacional.
JUSTIFICATIVA - Errado. O item apresenta o conceito de *thoughtput*.

93 Em uma máquina virtual, um processo, gerenciado por um motor de isolamento, faz a intermediação com o sistema operacional, limitando os recursos que podem ser usados e as *syscalls* que podem ser chamadas e passadas para o sistema operacional.
JUSTIFICATIVA - Errado. O conceito apresentado é o de contêiner.

94 As limitações e dificuldades para a utilização da virtualização incluem a falta de profissionais especializados e o uso de aplicativos de carga excessiva.
JUSTIFICATIVA - Certo. As principais limitações e dificuldades apontadas para se utilizar a virtualização são: aplicativos de carga excessiva e a falta de profissional especializado.

A respeito de arquitetura de aplicações, julgue os itens a seguir.

95 O processamento cooperativo em uma arquitetura cliente-servidor ocorre quando dois ou mais processadores processam uma simples transação.
JUSTIFICATIVA - Certo. Nesse tipo de processamento, a cooperação requer que haja dois ou mais processadores distintos para completar uma simples transação.

96 Na arquitetura de *cloud computing*, as camadas de aplicações e de dados do modelo PaaS (*Platform as a Service*) são gerenciadas pelo provedor de serviços.
JUSTIFICATIVA - Errado. No tipo de serviço plataforma como serviço (PaaS), o usuário pode instalar e gerenciar suas próprias aplicações, desenvolvidas por ele ou adquiridas de terceiros, utilizando as ferramentas e bibliotecas oferecidas pelo provedor. Ou seja, as aplicações que rodam em uma PaaS são desenvolvidas especificamente para ela.

97 Os requisitos funcionais definem as funcionalidades de certa aplicação e os não funcionais definem a maneira pela qual a aplicação executa tais funcionalidades.
JUSTIFICATIVA - Certo. Requisitos funcionais estão associados às funcionalidades que ditam o que o sistema deve fazer; já os não funcionais estão associados às restrições de funcionalidades que ditam a forma como o sistema deve fazer determinada ação.

No que concerne ao padrão MVC (*Model-View-Controller*) e a sistemas de N camadas, julgue os próximos itens.

98 No padrão MVC, o elemento do modelo encapsula as funcionalidades, os objetos de conteúdo e os estados da aplicação.
JUSTIFICATIVA - Certo. A arquitetura *model-view-controller* (MVC) desassocia a interface do usuário da funcionalidade e do conteúdo de informações de uma aplicação. O modelo contém todo o conteúdo e a lógica de processamento específicos à aplicação, inclusive todos os objetos de conteúdo, acesso a fontes de dados/informações externas e toda a funcionalidade de processamento específica para a aplicação.

99 Em aplicações construídas em camadas, as camadas mais internas são conectadas ao sistema operacional, enquanto as camadas mais externas atendem às interfaces dos usuários.
JUSTIFICATIVA - Certo. Na camada mais externa, os componentes atendem às operações da interface do usuário, contudo, na camada mais interna, os componentes fazem a interface com o sistema operacional. As camadas intermediárias fornecem serviços utilitários e funções de *software* de aplicação.

Acerca de API, julgue os itens subsequentes.

100 Quando utilizados em operações em bancos de dados, os métodos HTTP de `GET`, `POST`, `PUT` e `DELETE` podem ser associados aos comandos CRUD de `SELECT`, `CREATE`, `UPDADE` e `DELETE`, respectivamente.
JUSTIFICATIVA - Certo. Os métodos HTTP são os responsáveis por provocar alterações nos recursos identificados pelas URLs. Essas modificações são padronizadas, de maneira que o método `GET` é utilizado para recuperar os dados identificados pela URL, o `POST`, para criar um novo recurso, o `PUT`, para atualizar um recurso e o `DELETE`, para apagar um recurso. Esses quatro métodos podem ser diretamente relacionados a operações de banco de dados, da seguinte forma: o método `GET` do HTTP se relaciona com o `SELECT`, o `POST`, com o `CREATE`, o `PUT`, com o `UPDATE` e o `DELETE`, com o `DELETE`.

101 Na execução de uma API, os códigos de erro originados no servidor seguem o padrão `4xx`, e os códigos de sucesso seguem o padrão `1xx`.
JUSTIFICATIVA - Errado. Toda requisição que é enviada para o servidor retorna um código de *status*, sendo tais códigos categorizados em cinco famílias. O código `1xx` refere-se aos códigos informacionais, `2xx`, aos códigos de sucesso, `3xx`, aos códigos de redirecionamento, `4xx`, aos erros causados pelo cliente e `5xx`, aos erros originados no servidor.

102 Em conformidade com o formato JSON, os dados de um cliente podem ser representados da forma apresentada a seguir.

```
{
"cliente:" {
  "nome:" "João da Silva",
  "cpf:" "123.456.789-10"
```

```
}
}
```

JUSTIFICATIVA - Errado. No código, na segunda linha, após o termo "cliente:" deveria haver o operador : (dois pontos). O código deveria ser o seguinte.

```
{
"cliente": {
  "nome": "João da Silva",
  "cpf": "123.456.789-09"
}
 }
```

No que se refere à interoperabilidade e componentização, julgue os itens que se seguem.

103 Nos barramentos de serviços corporativos (ESB), o recebimento e a conversão de mensagens para o formato esperado são realizados pelos *endpoints*.
JUSTIFICATIVA - Errado. Nos barramentos de serviços corporativos (ESB), o recebimento e conversão de mensagens para o formado esperado são realizados pelos adaptadores.

104 A componentização visa utilizar apenas um componente gráfico padronizado nas diversas interfaces de uma aplicação.
JUSTIFICATIVA - Errado. Uma das técnicas fundamentais quando se tem grandes aplicações é a componentização. Uma componentização de *software* distingue-se pelo fato de que a base de código é dividida em pequenas porções que fornecem comportamento por meio de interações limitadas e bem definidas com outros componentes. O uso da abordagem de componentização geralmente é descrito em termos de incentivar reuso ou ter boas propriedades arquiteturais, como baixo acoplamento.

105 Componentes de um *software* podem ter acesso a outros componentes desse mesmo *software* ou de outro *software*.
JUSTIFICATIVA - Certo. Os componentes podem ter dependências em relação a outros componentes. Isso significa que um componente pode invocar a interface de outro componente para realizar uma chamada.

No que diz respeito a *design* de *software*, julgue os próximos itens.

106 Uma aplicação é excessivamente complexa quando há nela muitas ocorrências do mesmo trecho de código.
JUSTIFICATIVA - Certo. A característica apresentada é um sintoma de complexidade excessiva de uma aplicação.

107 Separação por afinidades é uma técnica utilizada para separar aplicações em unidades menores, facilitando-se a resolução de problemas de *design*.
JUSTIFICATIVA - Certo. A separação por interesses é um conceito de *design* de projeto que sugere que qualquer problema complexo pode ser tratado mais facilmente se for subdividido em trechos a serem resolvidos e(ou) otimizados independentemente. O interesse se manifesta como uma característica ou um comportamento especificado como parte do modelo de requisitos do *software*. Por meio da separação por interesses em blocos menores e, portanto, mais administráveis, um problema toma menos tempo para ser resolvido.

108 Um dos princípios do DDD (*domain-driven design*) é que o *software* possa ser construído mesmo sem o entendimento do domínio do cliente.
JUSTIFICATIVA - Errado. No *domain-driven design*, é impossível resolver o problema no domínio do cliente sem entendê-lo profundamente.

A respeito dos padrões e das tecnologias de comunicação e integração de sistemas, julgue os itens que se seguem.

109 Uma API RESTful permite a comunicação entre cliente e servidor via HTTP, com arquitetura *stateless*, com suporte a *cache*, interface padronizada e sistema em camadas, garantindo escalabilidade e eficiência.
JUSTIFICATIVA - Certo. Uma API RESTful facilita a comunicação entre um cliente (como um navegador ou aplicativo) e um servidor por meio de solicitações HTTP. Na comunicação via HTTP, as APIs RESTful utilizam métodos HTTP, como GET, POST, PUT e DELETE, para interagir com recursos no servidor.

110 Em JSON, um objeto é um conjunto não ordenado de pares nome/valor, delimitado por chaves, com os nomes e valores separados por vírgulas e os pares separados por dois-pontos.
JUSTIFICATIVA - Errado. Um objeto em JSON é um conjunto não ordenado de pares nome/valor. Isso significa que os itens dentro de um objeto não têm uma ordem fixa e podem ser acessados por seus nomes, independentemente da posição. Os objetos são iniciados com uma chave esquerda { e terminados com uma chave direita }. Tudo que está entre essas chaves faz parte do objeto. Os itens de um objeto JSON são organizados como pares nome/valor.

Julgue os seguintes itens, relativos a práticas e ferramentas de DevOps e integração contínua de código.

111 A expansão gradual em Canary Releases consiste na liberação imediata de uma nova versão para a totalidade da base de usuários, sendo a nova versão do *software* monitorada em tempo real.
JUSTIFICATIVA - Errado. Em Canary Releases, a nova versão não é liberada imediatamente para toda a base de usuários, mas sim de forma gradual e controlada, começando por um pequeno grupo e expandindo progressivamente. Embora a versão seja monitorada em tempo real, a liberação para todos os usuários ao mesmo tempo não é condizente com a estratégia.

112 Em Kubernetes, a política padrão de *pull* é `IfNotPresent`, mas pode ser configurada como `Always` para forçar o *pull* da imagem, ajustando a `imagePullPolicy` ou usando a *tag* `latest`.
JUSTIFICATIVA - Certo. A política padrão de *pull* é `IfNotPresent` a qual faz com que o kubelet ignore o processo de *pull* da imagem, caso a mesma já exista. Caso seja necessário forçar o processo de *pull*, é necessário que se utilize uma das opções a seguir:

- definir a `imagePullPolicy` do contêiner para `Always`.
- omitir `imagePullPolicy` e use: `latest` como a *tag* para a imagem a ser usada.
- omitir `imagePullPolicy` e a *tag* da imagem a ser usada.
- habilitar o `AlwaysPullImages` controlador de admissão.

Quando `imagePullPolicy` é definido sem um valor específico, ele também é definido como `Always`.

A respeito de gerência de configuração de *software*, julgue os itens subsecutivos.

113 A integração contínua no GitHub Actions é configurada manualmente em cada repositório, e os *workflows* são executados apenas quando o código é aprovado por uma revisão de código.
JUSTIFICATIVA - Errado. No GitHub Actions, a configuração dos *workflows* é feita automaticamente através de arquivos YAML, que são adicionados ao repositório no diretório .github/workflows. Os workflows são executados automaticamente em resposta a eventos, como *push*, *pull request*, entre outros, sem depender de uma aprovação manual por revisão de código. O GitHub Actions não requer aprovação para iniciar um *workflow*, embora seja possível

configurar algumas regras de aprovação dentro de um *workflow*.

114 Em GIT, o comando `cherry-pick` aplica o *commit* específico de uma *branch* a *branch* atual, criando um novo *commit* na *branch* de destino.
JUSTIFICATIVA - Certo. O comando GIT *cherry-pick* permite aplicar *commits* específicos de uma *branch* em outra *branch*. Esse comando é útil quando se deseja trazer alterações específicas de um *branch* (por exemplo, uma correção de *bug* ou uma funcionalidade) para outra *branch*, sem a necessidade de mesclar a *branch* completa.

115 Considerando que a *branch feature* já esteja ativa e que todas as ações devam ser realizadas nela, o comando a seguir organiza a *branch* local, sincroniza-a com a *branch* remota, aplica um *commit* específico, limpa o histórico recente e restaura as alterações locais salvas temporariamente.

```
git stash - m "Salvar mudanças temporárias" && git pull origin feature && git cherry-pick --no-commit <commit-hash> && git rebase -i HEAD~5 && git stash pop
```

JUSTIFICATIVA - Certo. Esse comando realiza uma série de operações do GIT:
• `git stash - m "Salvar mudanças temporárias"`: Armazena alterações locais não confirmadas em um *stash* temporário com a mensagem "`Salvar mudanças temporárias`".
• `git pull origin feature`: atualiza o *branch* ativo com as últimas mudanças do repositório remoto.
• `git cherry-pick --no-commit <commit-hash>`: aplica as alterações de um *commit* específico ao *branch* atual, mas sem criar um *commit*.
• `git rebase -i HEAD~5`: Inicia um *rebase* interativo nos últimos 5 *commits*, permitindo editar, reordenar ou remover *commits*.
• `git stash pop`: restaura as alterações armazenadas no *stash* e remove o *stash*.

A respeito do padrão arquitetural MVVM (*Model*, *View* e *View-Model*) e dos padrões de projetos GoF, julgue os itens que se seguem.

116 O padrão Iterator oferece uma forma sequencial de acessar os elementos de uma coleção de objetos, expondo a estrutura interna dessa coleção.
JUSTIFICATIVA - Errado. O Padrão Iterator oferece uma maneira de acessar os elementos de uma coleção de objetos sequencialmente sem expor a estrutura interna dessa coleção. O objetivo principal do padrão Iterator é fornecer um meio de percorrer a coleção sem que o cliente precise conhecer ou interagir diretamente com sua implementação interna. Dessa forma, a estrutura interna da coleção permanece encapsulada, promovendo o desacoplamento.

117 A *view-model* é a ponte entre a *view* e o *model*, sendo responsável por expor dados e comandos para a interface do usuário, mantendo a lógica de apresentação independentemente da interface e da lógica de negócios.
JUSTIFICATIVA - Certo. No padrão MVVM (*model-view-viewmodel*), a *view-model* é de fato o componente central. Ela atua como uma ponte entre a *view* (interface do usuário) e o *model* (dados e lógica de negócios).

118 O padrão Facade simplifica a interação com sistemas internos ao consolidar várias funcionalidades em uma única interface, proporcionando um acesso simplificado aos subsistemas sem alterar sua estrutura ou autonomia.
JUSTIFICATIVA - Errado. O padrão Facade é um padrão de *design* estrutural usado para fornecer uma interface simplificada para um sistema complexo de classes, bibliotecas ou *frameworks*. Ele não suprime a autonomia granular dos subsistemas envolvidos, mas atua como um ponto de acesso único que simplifica a interação com o sistema.

Julgue os próximos itens, a respeito da arquitetura hexagonal e da autenticação única (*single sign-on*).

119 O IdP (*identity provider*) realiza autenticação transmitindo credenciais em texto para os provedores de serviço, utilizando *basic authentication*, e mantendo sessões armazenadas em *caches* centralizados.
JUSTIFICATIVA - ERRADA. Um *Identity Provider* (IdP) não delega credenciais brutas aos provedores de serviço. Em vez disso, o IdP autentica o usuário e emite *tokens* de segurança ou asserções que os provedores de serviço utilizam para verificar a identidade do usuário. Protocolos como OAuth, SAML e OpenID Connect são comumente usados para esse fim, garantindo uma autenticação segura e sem a necessidade de transmitir credenciais brutas. Além disso, as sessões não são armazenadas em *caches* compartilhados diretamente acessíveis por todos os aplicativos conectados.

120 Na arquitetura hexagonal, a camada de aplicação em um adaptador atua como a interface de orquestração, responsável por interpretar as solicitações recebidas pelas portas de entrada.
JUSTIFICATIVA - Certo. Na arquitetura hexagonal (também conhecida como arquitetura de portas e adaptadores), a camada de aplicação desempenha o papel de interface de orquestração. Ela é responsável por interpretar as solicitações recebidas pelas portas de entrada (*adapters*) e direcioná-las para os componentes apropriados dentro do sistema.

Espaço livre