EXAME 2021

P1 - “Uma das maiores vantagens de usar o TDD é que estamos a fazer pequenos incrementos ao escrever código e, portanto, os problemas são mais fáceis de corrigir, uma vez que apenas abordam um pedaço de código limitado.” [In: Relatório de projeto de TQS] A vantagem referida pode ser considerada real e fundamentada? Selecione uma opção:
(A) Sim, o código de produção é feito na medida do suficiente para satisfazer os testes e a causa dos eventuais problemas é fácil de localizar.
(B) Não, se cada teste só incidir sobre pedaços limitados do código a correção de problemas é ilusória, porque não verifica a integração dos módulos.
(C) Sim, o TDD previne a inserção de falhas no código.
(D) Sim, em TDD, é frequente reutilizar pedaços limitados do código dos próprios testes para escrever o código de produção.
(E) Não, porque é preciso escrever bastante mais código em produção do que aquele que é necessário para passar os testes.

P2 - A abordagem BDD preconiza a realização de testes numa estratégia de QA, relacionados diretamente com as user stories. Qual das seguintes caraterísticas NÃO se aplica ao BDD? Selecione uma opção:
(A) Os (novos) cenários tornam-se mais fáceis de implementar à medida que mais “passos” ficam implementados, pois há a possibilidade de partilhar/reusar passos comuns.
(B) Facilita a compreensão do comportamento esperado do software, ao fomentar a colaboração entre a equipa técnica e a equipa do “negócio”.
(C) É suportada por uma linguagem natural que permite descrever o teste na área do domínio do problema, entendida pela equipa alargada.
(D) Dispensa a utilização de outras técnicas de especificação de requisitos, uma vez que permite centralizar a especificação do produto na descrição da “feature”, que é sujeita a controlo de versões, com o restante código.
(E) As "features" podem ser chamadas de "especificações executáveis" porque alimentam a realização de testes (em código).

P3 - A utilização de personas (personagens) ajuda a focar a criação de valor nas motivações de utilizadores concretos e cria contextos reutilizáveis para escrever as histórias. Como é que os processos de QA podem beneficiar do recurso às personas? Selecione uma opção:
(A) Embora as personas ajudem a concretizar quem beneficia do produto, é preferível usar os papéis do utilizador diante do sistema (e.g.: um cliente, um aluno) para não enviesar os critérios de aceitação.
(B) As personas fixam um conjunto de caraterísticas do utilizador e ajudam a criar exemplos de utilização coerentes, na escrita dos critérios de aceitação.
(C) Apesar de serem úteis no processo de descoberta de requisitos e fixação das histórias, o conceito de persona não tem continuidade para a definição de testes funcionais.
(D) As personas podem ser usadas parar criar os ramos (“branches”) de um repositório (um “branch” para cada persona), facilitando a distribuição do trabalho e a revisão pelos pares.
(E) As personas são usadas para os projetos que seguem BDD, para incluir nos ficheiros de “features”, com os exemplos de utilização.

P4 - É uma boa prática de QA incluir na lista de condições para concluir um incremento (“definition of done”) a obrigação do programador suplementar o seu código de produção com testes? Selecione uma opção:
(A) Sim, porque os ambientes de CI não podem realizar as respetivas pipelines se não houver testes no projeto.
(B) Não; se a equipa adotar a revisão de código pelos pares (peer code review), os testes são facultativos e apenas necessários em alguns casos mais complexos.
(C) Sim, porque a possibilidade de fazer entrega contínua depende de um elevado grau de confiança na qualidade do código, com níveis de cobertura muito próximos de 100%.
(D) Sim, porque o programador tem conhecimento da lógica interna do módulo e assim forma-se cumulativamente uma “rede de segurança” para prevenir futuras regressões.
(E) Não; existem várias ferramentas de análise de qualidade do código que compensam a eventual falta de testes e há ocasiões em que o programador tem de entregar código e não pode perder tempo com os testes.
P5 - O Gitflow é um fluxo de trabalho baseado em Git que foi primeiramente defendido e popularizado por Vincent Driessen. Quais as regras distintivas desta abordagem? Selecione uma opção:

(A) Usa o ramo “master” para integrar (merge) os novos incrementos; as novas funcionalidades devem ser desenvolvidas em ramos próprios, designados segundo o padrão “feature/my-feature”
(B) É uma variação de um modelo de “feature branch”, mas em prevê que a história global do projeto seja desenvolvida em dois ramos: o “master” (usado para marcar versões principais do projeto) e “develop” (para a integração regular dos incrementos).
(C) Não é recomendado para todos os projetos, uma vez que não contempla, no processo de trabalho da equipa, a inclusão de “merge/pull requests” para a linha do “master” de forma natural.
(D) Cada programador deve criar um branch próprio, fazer alterações no seu branch “privado”, e integrar os incrementos no master partilhado.
(E) Não é recomendado para todos os projetos, uma vez que tem extensões próprias que podem ser incompatíveis com os repositórios Git comuns (e.g.: “git flow release start 0.1”) ou em que a linha principal não se chama master (e.g.: “master” vs “main”).

P6 - Considerando o papel dos testes de integração, identifique a afirmação FALSA: Selecione uma opção:
(A) Os testes de integração podem ser otimizados recorrendo ao uso generalizado de objetos de substituição (mocks) em lugar das dependências.
(B) Os testes de integração, para além da unidade sob teste, usam componentes ou infraestruturas adicionais como a rede, base de dados ou sistema de ficheiros.
(C) Os testes de integração podem ter de esperar que certas etapas do ciclo de CI estejam concluídas, para garantir que houve etapas prévias que prepararam o ambiente necessário para os testes de integração.
(D) Os testes de integração devem ser utilizados para testar as arquiteturas de microsserviços
(E) Quando um teste de integração falha, isso indicia um problema na articulação dos componentes ou na disponibilidade dos serviços.

P7 - Nem sempre é necessário, ou conveniente, testar uma unidade de software (ou partes). Quando é que pode ser contraproducente/desaconselhável escrever testes (unitários)? Selecione uma opção:
(A) No código que ainda não foi revisto; é útil aguardar pelo feedback da revisão do código pelos pares para então escrever os testes.
(B) O código que foi automaticamente gerado (e.g.: vários “toString” e "equals“), também deve ser testados para atingir a cobertura próxima de 100%.
(C) Se a equipa não inclui a avaliação de cobertura nos patamares de qualidade requeridos (quality gate), é contraproducente estar a dedicar tempo à escrita de testes unitários.
(D) É sempre necessário e útil escrever testes unitários para a totalidade do código de um projeto.
(E) Objetos que não incluem lógica relevante, como entidades (de persistência) geradas, dispensam a realização de testes.

P8 - “Os testes de sistema irão ser conduzidos pelos stakeholders, com base na informação presente no manual de qualidade do software e no relatório de especificações do produto. Estes testes irão indicar se cada componente é capaz de satisfazer os requisitos definidos nas especificações do produto.” [In: relatório de projeto de TQS] Esta declaração apresenta vários problemas, assinalados a seguir. Qual das afirmações é FALSA? Selecione uma opção:
(A) Os testes são automatizados e, por isso, os stakeholders nunca participam na realização de testes.
(B) Os testes de sistema são, principalmente, orientados para a qualidade interna do produto e não para as necessidades dos utilizadores.
(C) Os testes de sistema servem para validar o sistema globalmente, não cada componente por si.
(D) O Manual de Qualidade não tem informação suficiente para determinar os testes de sistema que podem ser usados na verificação de um produto.
(E) Os stakeholders não realizam testes de sistema, apesar de participar na elaboração de critérios de aceitação

P9 - Em que situação é mais provável que o programador recorra ao uso de objetos de substituição (mocks) num plano de teste? Selecione uma opção:
(A) A equipa adotou uma metodologia BDD para os testes.
(B) A unidade sob teste ainda não está pronta (implementada) e é necessário criar uma implementação provisória do seu comportamento.
(C) A unidade sob teste é um serviço (endpoint) de uma API REST.
(D) A unidade sob teste utiliza uma base de dados em memória.
(E) A unidade sob teste apresenta uma dependência de um serviço remoto, com um resultado variável.

P10 - Considere a seguinte descrição sobre um teste incluído numa aplicação Spring Boot: "este teste procura confirmar que os objetos JSON da resposta estão a ser mapeados corretamente quando é usado um objeto válido no pedido, e se não forem válidos os pedidos, que estamos a enviar o erro HTTP correto, juntamente com uma razão descritiva para a falha." [In: relatório de um projeto de TQS] Qual o tipo de teste (predefinido na Spring Boot) mais eficiente/adequado para o propósito enunciado? Selecione uma opção:
(A) @MockMvc
(B) @SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
(C) @SpringBootTest
(D) @ExtendWith(MockitoExtension.class)
(E) @WebMvcTest

P11 - M. Fowler apresenta dez práticas recomendadas para a preparação de um sistema de Integração Contínua. Nesse contexto, qual das seguintes é ERRADA: Selecione uma opção:
(A) Todos os membros da equipa têm acesso imediato ao feedback do estado das builds.
(B) Todos os membros da equipa devem fazer a entrega frequente de incrementos para a linha principal do desenvolvimento, por exemplo, diariamente.
(C) Devem existir testes feitos em ambientes que mimetizam as condições de produção.
(D) As builds devem ser feitas no ambiente de integração com uma periodicidade diária e os resultados distribuídos por um sistema de feedback rápido (e.g.: Slack, Discord).
(E) Devem existir formas automáticas de fazer as instalações, para configurar e montar diferentes ambientes de teste.

P12 - É muito frequente a utilização de virtualização por containers (contentores), como o Docker, em ambientes de CI/CD. Relativamente ao contributo dos containers nos processos de QA, qual das seguintes afirmações é ERRADA: Selecione uma opção:
(A) A definição dos containers pode ser sujeita a controlo de versões juntamente com o resto do código e recursos do projeto, de forma que a configuração da infraestrutura evolua com história do próprio projeto.
(B) Os containers podem ser geridos num repositório (um registry), de modo que containers criados em certas etapas do processo de CI/CD podem ser reusados em outras subsequentes.
(C) A generalidade dos testes pode beneficiar do uso de containers como ambiente de execução, designadamente com a utilização da biblioteca "Test Containers.
(D) As ferramentas de CI/CD suportam operações com containers, na definição dos respetivos pipelines de CI/CD
(E) Os containers oferecem uma forma natural e eficiente de isolar elementos da arquitetura, que podem ser ativados ou desativados pelos processos de CI/CD

P13 - A expressão "pipeline as code", associada, por exemplo, aos processos de CI do GitLab, significa que: Selecione uma opção:
(A) O pipeline é compilado, como o resto do código fonte, pela ferramenta de montagem/build usada no projeto (e.g.: Maven).
(B) É possível executar condicionalmente algumas etapas do pipeline, fazendo-as depender do sucesso das antecedentes.
(C) Existe uma definição do pipeline escrita numa sintaxe própria e essa configuração está sujeita ao controlo de versões, no repositório onde está o restante código do projeto.
(D) Um pipeline de CI inclui a compilação de código fonte e produz artefactos (binários) que são partilhados na equipa.
(E) Os passos do pipeline são executados quando é entregue novo código fonte no repositório associado (evento de "push").

P14 - A complexidade ciclomática do código influencia a facilidade com que pode ser lido e mantido. Que fatores podem alterar a métrica de complexidade, tal como é definida no SonarQube? Selecione uma opção:
(A) Número de variáveis declaradas numa classe.
(B) Número de métodos públicos de uma classe.
(C) Número de iterações realizadas em ciclos (repetições de um ciclo).
(D) Ocorrência de instruções que dividem o controlo de fluxo, tais como if, for, while, case.
(E) Tamanho de uma classe (número de linhas).

P15 - Um relatório de cobertura, por exemplo, gerado pelo Jacoco, apresenta diversas dimensões segundo as quais a cobertura pode ser calculada (e.g.: métodos, instruções, linhas, ramos alternativos). Qual das seguintes opções é uma descrição adequada do conceito de cobertura de instruções do código? Selecione uma opção:
(A) É uma métrica que determina a relação das instruções que foram executadas num conjunto de testes, em relação ao total de instruções do código analisado.
(B) É uma métrica que traduz a percentagem de código que foi inspecionado pelo ambiente de análise estática (e.g.: SonarQube) em relação à totalidade do projeto.
(C) É uma métrica que traduz o número de métodos (funções) que foram invocados nos testes, em relação à totalidade dos métodos existentes no código.
(D) É uma métrica utilizada para medir a percentagem de linhas de código fonte que são realmente executadas.
(E) É uma métrica usada para medir a percentagem de testes que foram executados com sucesso.

P16 - O guia de estilo para os colaborados do Android Open Source Project indica que a situação ilustrada no exemplo junto não é aceitável. Em termos gerais, qual é a alternativa preferencial para resolver esta situação?

```
void setServerPort(String value){
	try{
		serverPort =Integer.parseInt(value);
	} catch( NumberFormatException e){
}
```

Selecione uma opção:
(A) Registar a exceção no log da aplicação.
(B) Receber a exceção e lançar uma nova ocorrência da exceção genérica (Exception).
(C) Receber a exceção e lançar uma nova ocorrência de RuntimeException.
(D) Ignorar a exceção, sem a tratar, desde que se inclua um comentário no código a documentar a razão para o fazer.
(E) Lançar uma nova exceção, mas apropriada ao nível de abstração da operação (com a semântica adequada)

P17 - O suporte para testes existente no Spring Boot integra bem com o framework Mockito. O que seria uma situação natural para usar o Mockito no teste de componentes da Spring Boot? Selecione uma opção:
(A) O Mockito pode ser usado num teste @DataJpaTest de um componente @Repository para definir os resultados esperados das queries que seriam colocadas à base de dados.
(B) O Mockito pode ser usado num teste @DataJpaTest de um componente @Service para substituir as chamadas à base de dados por respostas sintetizadas, fazendo mock do componente repositório.
(C) O Mockito é um framework para usar no contexto de testes unitários e, genericamente, não é útil em testes da SpringBoot que precisam de injetar componentes adicionais (“beans”).
(D) O Mockito pode ser usado num teste @WebMvcTest restrito a um componente do tipo @RestController, para substituir a lógica do serviço, injetando um “bean” sintetizado (via @Spy).
(E) O Mockito pode ser usado num teste unitário de um módulo, para predefinir os resultados do RestTemplate, quando o módulo depende da invocação de métodos de uma API externa.

P18 - O excerto ilustrado consta de um ficheiro, de um projeto de alunos de TQS. O que é que este excerto mostra?

```
Scenario Outline: Logged user wants to search products category
        Given at least one <category> product exists
        And <user> is in the home page
        When <user> clicks the search button
        Then he sees the search results page
        And results are for <category>
        And number of results is more than zero

        Examples:
                |    user        |    category   |
                |    Francisco   |    BOOKS      |
```

Selecione uma opção:

(A) A definição parcial de uma "feature" compatível com o framework Cucumber, com três pós-condições avaliadas em "steps" próprios. **(Pré - given, açao - when, pós – then, … and)**

(B) O resultado de um teste com Cucumber, em que o utilizador "Francisco" pesquisou as entradas da categoria "BOOKS"

(C) Uma "feature" que leva à ativação, entre outros, do método de teste com a assinatura "@Then("number of results is more than zero") public void number_of_results_is_more_than_zero(**int** expectedValue) {}" **(NÃO TEM INT DEFINIDO)**

(D) O log produzido pelo Hibernate quando, na aplicação, foi solicitado a um componente do tipo "repositório" a consulta de dados da entidade utilizador, com uma projeção nos campos "user" e "category".

(E) A definição de um teste funcional do Selenium, com recurso à sintaxe do Gerkin.

P19 - Podemos beneficiar da utilização do framework Mockito para escrever testes da camada de serviços (i.e., componentes do tipo @Service), tendo presente a arquitetura comum de uma aplicação Spring Boot. Qual das situações corresponde a uma (boa) prática típica? Selecione uma opção:

(A) Utilizar @InjectMock no serviço sob teste e @Mock dos componentes (de tipo) repositório requeridos.

(B) Fazer @Mock do serviço que se pretende testar e utilizar o TestEntityManager para realizar as operações sobre a base de dados.

(C) Marcar a classe de teste com @Service, o que satisfaz automaticamente as dependências assinaladas com @InjectMock (i.e., obtém "mocks" dos repositórios necessários).

(D) Anotar o serviço que se pretende testar com @WebMvcTest **(testa controllers)** e as dependências necessárias com @MockMvc (especialmente os repositórios).

(E) Utilizar TestRestTemplate **(testes de integração)** para aceder à interface (programática) do serviço que se pretende testar e fazer @Mock dos componentes de tipo repositório requeridos

P20 - O excerto de código na Figura 1 mostra um cenário de teste usando práticas comuns da Spring Boot. Qual é a situação que se está a testar com este código (isto é, o que se pode concluir se o teste passar)?

```
@WebMvcTest(GreetingController.class)
public class WebMockTest {

        @Autowired
        private MockMvc mockMvc;

        @MockBean
        private GreetingService service;

        @Test
        public void greetingShouldReturnMessageFromService() throws Exception {
                when(service.greet()).thenReturn("Hello, Mock");

this.mockMvc.perform(get("/greeting")).andDo(print()).andExpect(status().isOk())

.andExpect(content().string(containsString("Hello, Mock")));
        }
}
```

Figura 1- Teste Spring Boot.

Selecione uma opção:
(A) O GreetingService contém um método de "greet()" que devolve a concatenação de "Hello" e o argumento passado na invovação do endpoint "/greeting".
(B) O GreetingService#greet() retorna uma resposta JSON que inclui sempre "Hello, Mock".
(C) O GreetingService usa uma instância mock do GreetingController que responde a pedidos HTTP no endpoint "/greeting".
(D) Existe um endpoint "/greeting", mapeado por algum método da classe GreetingController, que, por sua vez, usa o método GreetingService#greet()
(E) O GreetingController define um mapeamento para o caminho "/greeting" que deve ser invocado com um argumento do tipo String, contendo "Hello, Mock".

P21 - Os testes unitários devem ser atómicos, sucintos e independentes. Neste contexto, identifique a declaração FALSA. Selecione uma opção:
(A) O resultado de um teste unitário não deve depender dos resultados de outro teste prévio.
(B) Para evitar a proliferação de métodos numa classe de testes, o programador deve avaliar a oportunidade de verificar várias condições em cada método.
(C) Um teste unitário centra-se no teste de uma única função, o que torna clara a origem do problema, caso o teste falhe.
(D) Manter os testes unitários pequenos ajuda a que os ciclos de CI sejam eficientes.
(E) Os testes unitários devem ser independentes da ordem pela qual são executados, e até independentes do facto de os testes anteriores terem passado ou não.

P22 - Um teste típico escrito com Selenium WebDriver é um teste que: Selecione uma opção:
(A) Exercita parte de um sistema integrado, realizando o teste sobre a interface do utilizador, na web.
(B) Invoca ações numa aplicação web, controlando o browser Firefox ou Chrome (i.e., o WebDriver abstrato é estendido ou por FirefoxDriver ou por ChromeDriver).
(C) É usado para realizar os testes de aceitação no servidor de integração contínua, em lugar dos utilizadores concretos.
(D) Permite fazer mock das dependência dos serviços (componentes @Service) para testar a camada web de modo isolado.
(E) Simula a carga criada por um conjunto configurável de sessões de utilizadores, a aceder à camada da interface web.

P23 - Identifique, nos casos listados, uma situação em que recomendaria o uso do framework “Test Containers” na implementação de testes.
(A) O teste unitário deve ser executado num ambiente específico, diferente do ambiente em que o programador está a trabalhar.
(B) O teste de integração deve ligar-se a uma base de dados, configurada num ambiente de pré-produção.
(C) O teste de automação na web vai correr num servidor, sem ambiente gráfico.
(D) Há um subconjunto de testes de integração que devem testar a camada de acesso a dados sobre vários motores de bases de dados, de forma tão realista quanto possível e.g.: dialetos de SQL,…).
(E) A aplicação em desenvolvimento utiliza containers Docker para a instalação do ambiente de produção, designadamente com “Docker compose”.

P24 - Como é que o CI/CD pode ajudar nas alturas em que há “picos” de trabalho/pressão pelos resultados, por exemplo, nas alturas próximas das entregas dos milestones?
(A) Nesses casos, o pipeline de CI/CD pode revelar-se um “bottleneck”, e pode ser conveniente aligeirar os patamares de qualidade requeridos (quality gates).
(B) O CI/CD com a aplicação de testes e patamares de qualidade (quality gates) de forma automática e obrigatória, é determinante para prevenir a degradação da qualidade dos produtos.
(C) O CI deve ser simplificado, limitando-se os critérios da análise estática, mas não dos testes, de modo a não atrasar a entrega de incrementos.
(D) Quando a equipa está familiarizada com as práticas de CI/CD não ocorrem picos de entrega; o conceito de “contínuo” significa que não flutuações nas entregas.
(E) Nas alturas de pico de trabalho, é quando é mais necessário poder automatizar os testes de desempenho, no pipeline de CI/CD.

P25 - Considere que está a rever o código de outro programador que deve implementar um módulo de uma cache genérica (Figura 3), que observa um certo tempo de vida das entradas (TTL). Aconselharia algum dos seguintes “refactorings”?
(A) O método get(key) não deve provocar uma NullPointerException, porque o módulo que o invocou não tem como saber se a chave procurada existe ou não na estrutura.
(B) Os métodos apresentados não devem gerar NullPointerException; em alternativa, devem registar esses eventos num logger da aplicação.
(C) O métodoputOrTouch deve ser desagregado, de modo a que os respetivos testes unitários possam ser independentes entre si.
(D) O tempo fixo de expiração definido na interface (ENTRY_TTL) é um problema, uma vez que isso tira flexibilidade, designadamente para a escrita de testes que verificam se uma entrada já expirou.
(E) O código está bem; não oferece oportunidades relevantes de refactoring

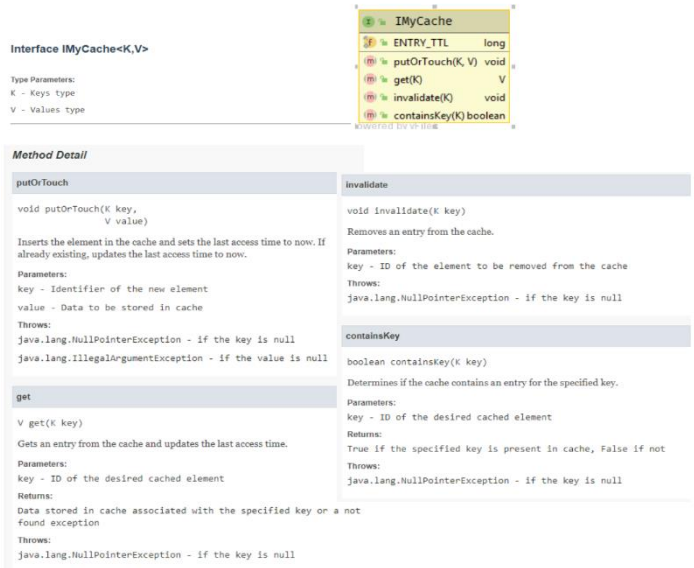

Figura 2: Interface para um módulo de Cache.

EXAME 2022

1- A norma ISO/IEC25010 define um modelo de qualidade para sistemas de software. Qual a sua importância para as empresas? Selecione uma opcao de resposta:

A. Facilitar a integração (compatibilidade) entre sistemas que seguem este modelo de qualidade.

B. Define estratégias para garantir a integridade dos dados, o não repúdio das ações e a reutilização dos módulos.

C. Serve para avaliar o grau até onde o conjunto de funcionalidades implementadas cobre a totalidade das necessidades e objetivos dos utilizadores.

D. Determinar um conjunto de características de referência que devem ser consideradas para maximizar a facilidade de manter um produto (maintainability), sem introduzir defeitos.

E. Pode ser usado com base para um sistema de avaliação de qualidade, uma vez que o modelo estabelece as características a considerar na avaliacao de um produto de software.

3 - O excerto junto mostra o conteudo do ficheiro "application-integration.properties" de um projeto SpringBoot. Qual a sua utilidade no contexto dos testes?

```
spring.datasource.url=jdbc:mysql://localhost:3306/demo
spring. jpa.hibernate.ddl-auto=create-drop
spring.datasource.username=demo
spring.datasource.password=demo
```

Selecione uma opçao de resposta:

A. Não é usado nos testes; teria de se designar "application.properties" para ser utilizado pela ação da anotação @AutoConfigureTestDatabase.

B. Serve para criar um contexto de dados persistente ao longo das várias execuções dos testes, de maneira a que cada ciclo de CI possa usar os dados (na base de dados) do ciclo anterior.

C. Serve para configurar uma base de dados de teste temporária, em memória, que é destruída em cada execução dos testes.

D. Não é usado nos testes; serve para configurar a base de dados de runtime (produção).

E. Serve para definir uma base de dados temporária, por exemplo, configurada na anotação @TestPropertySource

4 - Uma das regras do SonarQube sinaliza como uma vulnerabilidade do código, em projetos de Spring Boot, a utilização de direta de entidades como argumentos de métodos de uma API: «Persistent entities should not be used as arguments of "@RequestMapping" methods»

O programador pode resolver esta vulnerabilidade de varias formas, mas qual destas ideias NÃO é adequada? Selecione uma opção de resposta:

A. Usar identificadores das entidades, em vez da entidade em si, na definição dos argumentos, quando isso seja possível.

B. Os DTO ("Data Transfer Objects") auxiliares são versões mais leves que as Entidades (que mantêm os mapeamentos do JPA), e tornam mais rápido e seguro a sua utilização.

C. Especificar, na configuração do @RequestMapping, que o método usa o formato JSON para codificar os dados.

D. Usar um POJO auxiliar, sem contexto de persistência, como argumento do método da API.

E. Utilizar objetos específicos para fazer o mapeamento dos atributos das entidades para os parâmetros dos métodos da API, através de "Data Transfer Objects".

5 - Os testes unitários são escritos pelo programador, como parte da estratégia de QA.
É FALSO que estes testes o ajudem:
A. Na revisão do seu código pelos pares (em code reviews).
B. A escrever menos testes de aceitação.
C. A aumentar a cobertura do código.
D. A documentar a utilização pretendida de um componente;
E. A entender o contrato do módulo (requisitos do que vai construir);

6 - Há alguma vantagem na utilização do Maven para configurar um projeto ("Maven project") quando se esta a usar um ambiente de CI? Selecione uma opçao de resposta:
A. Não, o ambiente de CI precisa sempre de analisar o projeto e proceder à respetiva construção (build)
B. Sim, o Maven garante que o código é compilado e testado e passa os binários ao serviço de Cl livres de erros.
C. Sim, o Maven contribui para uma descrição do projeto neutra em relação aos IDE facilitando a construção do projeto no ambiente de CI.
D. Não, a linguagem para a escrita do pipeline de Cl é diferente e não tira partido da configuração descrita no POM.xml
E. Não, a maior parte dos ambientes de CI nao esta preparado para integrar com projetos Maven.

8 - A que e que se refere a ideia de continuidade subjacente as designacoes de integracao continua e entrega contínua (CI/CD)? Selecione uma opçao de resposta:
A. É uma ideia exigente, que transmite o princípio de que a equipa deve entregar novos incrementos ao cliente de forma regular e frequente, ou seja, contínua, ao longo do projeto.
B. Através de triggers (como webhooks), os sistemas estão continuamente a monitorizar o repositório de código para reagir a alterações e proceder a novas instalações, em cada entrega de código.
C. Os programadores são incentivados a fazer a entrega e integração de código com grande frequência e, daí, a ideia de continuidade.
D. A cultura da equipa, os processos de trabalho e as ferramentas estão preparados para, a qualquer altura, integrar os contributos dos programadores e entregar novas funcionalidades do produto.
E. Uma equipa que trabalha com rotinas de CI/CD é capaz de propagar as novas funcionalidades para produção, sem interromper as operações; do ponto de vista dos utilizadores, as operações decorrem de forma contínua, sem disrupção.

9 - Várias ferramentas, como o Jenkins, Circle CI, GitLab CI, GitHub Actions, entre outras, utilizam a noção de pipeline. O que é um pipeline de Integração Contínua (continuous integration)? Selecione uma opçao de resposta:
A. É uma visualização das várias etapas (stages) configuradas no projeto, que devem ser realizadas sempre que ha um novo incremento, de forma sequencial ou paralela.
B. É uma implementação automatizada do processo de compilação, montagem, teste e instalaçao dos ambientes de ensaio e de produçao de uma aplicação
C. É um ficheiro (na notação .yml) usado para configurar as etapas de compilação e teste de um projeto, quando ocorrem incrementos de código num repositório.
D. É um ficheiro de configuração que deve acompanhar o repositório Git, de modo a que as novas entregas sejam detetadas de imediato.
E. É uma especificação dos passos que devem ser feitos sempre que é detetado um novo incremento, e que decisões devem ser tomadas quando certas condições ocorrem (e.g .: partes do processo passam ou falham).

12 - Considere o excerto de codigo que testa um componente Spring Boot (EmployeeRestController.class), relativo ao método POST acessível em "/api/employees".
Este teste esta a falhar, com a indicaçao de erro na injecao da dependência na variavel "service". Do que se pode inferir, qual é o problema do código que provoca o erro do teste?

```
@WebMvcTest(EmployeeRestController.class)
public class EmployeeControllerTest {

	@Autowired
	private MockMvc mvc;

	@MockBean
	private EmployeeService service = new EmployeeServiceImp1();

	@Test
	public void whenPostEmployee_thenCreateEmployee() throws Exception {
		Employee alex = new Employee("alex");
		given(service.save(Mockito.any())).willReturn(alex);

	mvc.perform(post("/api/employees")
		.contentType(MediaType.APPLICATION_JSON)
		.content(JsonUtil.toJson(alex)))
		.andExpect(status().isCreated())
		.andExpect(jsonPath("$.name", is("alex")));
	verify(service, VerificationModeFactory.times(1))
		.save(Mockito.any());
	reset(service);
}
```

Selecione uma opção de resposta:
A. Na declaração da variável "service" deveria estar a anotação "@InjectMocks" e não "@MockBean"
B. As instâncias marcadas com @MockBean deve ser criadas (injetadas) pelo ambiente e não com o operador "new".
C. O parâmetro Mockito.any() não é adequado como argumento do método service.save().
D. A anotação @WebMvcTest não é adequada para este teste, uma vez que o teste acede a um endpoint REST (EmployeeRestController.class)
E. A operação Mockito.verify() deve ser executada no início do teste (e não no final), uma vez que verifica ser as instâncias são nulas e, nesse caso, procede à iinjeção das dependências.

13 - Relativamente ao conceito de "user story", tal como é entendido nos processos de Garantia de Qualidade em metodologias ágeis, qual é a afirmação verdadeira? Selecione uma opçao de resposta:
A. A user story descreve uma funcionalidade, segunda a perspetiva de quem dela beneficia, e é usada como unidade para prioritizar o trabalho e a entrega de valor.
B. As user stories são criadas no início de dada iteração, apresentado de forma breve as novas funcionalidades a desenvolver na presente iteração/sprint.
C. As user stories são preparadas pelos representantes do negócio/cliente para apresentarem as histórias que pretendem realizar no sistema.
D. As user stories sãopreparadas pelos testers, para detalhar os requisitos funcionais do sistema e os critérios de aceitação.
E. As user stories são escritas pelo dono do produto (product owner) e entregues à equipa no início de cada iteração/sprint.

15 - Os projetos baseados em práticas ágeis colocam maior ênfase na automação dos testes do que os projetos "tradicionais" (desenvolvimento sequencial), porque:
A. As alterações aos requisitos acontecem diariamente e os testes precisam de ser regerados de forma automática, para garantir a velocidade da equipa e a entrega diária de valor.
B. Os projetos ágeis dependem sobretudo de testes unitários, em detrimento de outros. Como os testes unitarios devem ser em grande numero, nao podem ser executados manualmente.
C. As iterações/sprints são de duração fixa. A equipa deve garantir que todos os testes podem ser completamente executados no final de cada iteração/Sprint.
D. Os testes são executados regularmente, tanto para novos incrementos como para o código ja existente, inviabilizando o recurso a testes manuais no ciclo de CI.
E. Os testes geram as métricas de qualidade do código que determinam o grau de prontidão/confiança na qualidade do código e a passagem nos controlos de qualidade ("quality gates") definidos.

16 - Qual a interpretação mais adequada do conceito "dívida técnica" (technical debt), tal com é usada no ambiente de qualidade do SonarQube?
A. É uma estimativa do tempo de trabalho necessário para fazer passar os testes que estão a falhar.
B. É a diferença do nível de cobertura atual para cobertura plena (100%).
C. É uma estimativa do tempo necessário para corrigir os problemas encontrados durante a análise estática do código.
D. É o diferencial do nível atual de qualidade do projeto, determinado pela ferramenta de análise estática, para o nível de qualidade plena (sem quaisquer problemas).
E. É o número de user stories nao aceites na iteracao corrente, por nao passarem os testes definidos.

19 - É muito frequente a utilização de virtualização por containers (contentores), como o Docker, em ambientes de CI/CD. Qual das seguintes afirmações é ERRADA em relação ao contributo dos containers nos processos de QA: Selecione uma opcao de resposta:
A. As ferramentas de CI/CD integram suportam operações com containers, na definição dos respetivos pipelines de CI/CD.
B. Os containers oferecem uma forma natural e eficiente de isolar elementos da arquitetura, que podem ser ativados ou desativados pelos processos de CI/CD.
C. Os containers são muito úteis para instanciar localmente, mas limitados, quando é necessario a ativacao de varios containers em varios servidores (i.e., nao é possível trabalhar com clusters de containers distribuídos).
D. Os containers podem ser geridos num repositório (um registry), de modo a que containers criados em certas etapas do processo de CI/CD podem ser reusados em outras.
E. A definição dos containers pode ser sujeita a controlo de versões juntamente com o resto do código e recursos do projeto, de forma a que a configuracão da infraestrutura evolua com história do próprio projeto.

20 - Os testes funcionais implementados com recurso a Selenium/WebDriver podem recorrer ao padrão "page object model" (POM). Qual a vantagem de utilizar este padrão?
A. A programação da página web deve usar uma linguagem por objetos, com injeção de dependências, que e facilitador para os testes.
B. O padrão POM utiliza exemplos escritos em linguagem do domínio (features) para alimentar a execução dos testes de aceitacao, tornando os testes mais expressivos.
C. Cada página (web) é abstraída por uma classe; os testes de aceitação podem ser vistos como testes unitarios sobre métodos de uma classe, permitindo reutilizar ferramentas e ambientes de teste.
D. É vantajoso quando a implementação da camada web é baseada em Spring Boot MVC, mas não tem uma aplicação geral (e.g .: React JS).
E. Cada página (web) é abstraída por uma classe própria; as interações possíveis com essa página são expostas como métodos da classe (e.g .: searchByFirstName), tornado os testes mais expressivos e legíveis.

23 - Qual das seguintes práticas é mais recomendável na implementação de rotinas de revisão de código pelos pares:
A. A revisão do código deve ser feita preferencialmente pelos membros juniores na equipa, de modo a familiarizarem-se com os padrões de trabalho da equipa e com a próprio código/módulos já existentes.
B. Concentrar a revisão do código num período específico da semana, tomando o tempo que for necessário, para aumentar a produtividade da revisão.
C. O código a analisar deve já ter sido objeto de análises automáticas de qualidade (e.g .: análise estática) para descartar problemas prévios à própria revisão do código.
D. A pessoa que vai rever deve comecar por incluir no código fonte os seus pressupostos e observações (como comentários), para reduzir ambiguidades na comunicação.
E. Quando a equipa pratica a revisão de código pelos pares, a aprovação do código através de pull /merge requests torna-se dispensável, eliminado esse possível ponto de bloqueio.

25 - O framework Mockito permite sintetizar objetos de substituicao (mocks) que sao usados em lugar das verdadeiras implementações. Duas das principais anotações sao @Mock e @Spy.
Assinale a afirmação verdadeira
A. @Mock serve para marcar o objeto que deve ser sintetizado; @Spy serve para marcar a classe sob teste que precisa de usar o objeto sintetizado (e injeta o mock).
B. @Spy e usado para observar um objeto real; podemos verificar se certos métodos foram ou não utilizados nesse objeto, mas não é possível declarar, com expectativas (expectations), o valor predefinido que um determinado método deve retornar.
C. Se o objeto spyList foi anotado com @Spy, entao spyList.add("one") nao adicionada realmente o objeto à lista e o tamanho dela não é alterado.
D. @Spy é usado para observar um objeto real; permite chamar e executar os métodos normais do objeto, e rastrear todas as interações ocorridas, tal como faríamos com um objeto de substituição (mock).
E. Se o objeto mockList foi anotado com @Mock, entao mockList.add("one") nao adicionada realmente o objeto à lista; o tamanho da lista só é incrementado quando for invocado Mockito.verify(mockList).add("one");

26 - O "V-model" e uma forma conhecida de relacionar diferentes tipos de testes com diferentes âmbitos do sistema. Qual das afirmações é caraterística de uma estratégia de testes que segue o "V-Model"?
A. Os testes das camadas superiores devem usar os testes das camadas inferiores.
B. O esforço da equipa com as atividades de teste é cumulativo e aumenta de iteração para iteração.
C. Os testes são definidos antes da implementação do código, associados ao trabalho da etapa respetiva do ciclo de desenvolvimento.
D. Em cada iteração, os testes são executados depois da implementação, de forma ascendente, i.e., dos unitários para os de aceitação.
E. Existem diferentes tipos de teste, que devem ser executados numa ordem bem definida, segundo um esquema "top-down": âmbito dos requisitos, âmbitos dos serviços, âmbito das unidades.

30 - Que problemas poderiam ser apontados a um portfolio de testes baseado numa "pirâmide invertida" (poucos testes unitários, muitos testes de aceitação)?
A. Não assegura os testes de integração associados, por exemplo, à verificação de serviços complexos.
B. Não é escalável porque os testes de aceitação requerem a participação dos stakeholders.
C. Depende de ambientes gráficos para a execução dos processos de Cl (usualmente assegurados por servidores remotos)
D. É limitativo quanto à localizacao da causa dos erros, no caso de eles existirem.
E. É impeditivo da adoção de práticas de refactoring de código, pois não oferece a necessária "rede de segurança" para prevenir regressões.

EXAME 2019

P1 - O modelo de qualidade do ISO-25010 identifica vários parâmetros relativos à qualidade do produto (de software), entre eles a "Maintainability", que está relacionado com:
a) Integridade dos dados, não repúdio das ações, reutilização dos módulos.
b) Facilidade de operação por parte dos utilizadores, tolerância a falhas pontuais do hardware, facilidade de alterar um componente sem introduzir defeitos ou degradar a qualidade já existente.
c) Capacidade para estabelecer e executar testes de um sistema, uso de módulos para gerir a interdependência de componentes, capacidade de analisar o impacto de uma alteração no sistema global.
d) Facilidade de testar, facilidade de operação, perceção da utilidade do sistema pelos utilizadores.
e) Eficiência na utilização interna dos recursos, tempos de resposta das operações de acordo com os requisitos estabelecidos, existência de mecanismos de prevenção dos erros de utilização

P2 - A metáfora da "pirâmide dos testes" quer transmitir a ideia de que:
a) Existem testes mais importantes que outros (do topo para a base da pirâmide).
b) O esforço da equipa com as atividade de teste é comulativo e aumenta de iteração para iteração.
c) O número de testes diminiu com o burndown, i.e., à medida que menos itens de trabalho subsistem no backlog, há menos testes para executar.
d) Os testes podem ser agrupados tendo em conta a sua granularidade e objetivos.
e) Os testes das camadas superiores devem usar os testes das camadas inferiores

P3 - Que problemas poderiam ser apontados a um portfolio de testes baseado numa "pirâmide invertida" (menos testes unitários, mais testes de aceitação)?
a) Demora muito tempo a executar, é difícil de manter, perde especificidade na localização dos problemas.
b) Não é escalável porque os testes de aceitação requerem operação manual.
c) Obriga a fazer a gestão explícita de user stories, numa ferramenta adicional, não integradas no repositório de código.
d) Não é adequado para projetos orientados à disponibilização de API/serviços.
e) É impeditivo da adoção de práticas de refactoring de código, pois não oferece a necessária "rede de segurança" para prevenir regressões.

P4 - A análise estática de código:
a) é uma forma de detectar defeitos com muito baixo custo.
b) permite a validação antecipada/precoce dos requisitos dos utilizadores.
c) a análise estática, com ferramentas completas, tornam o teste dinâmico desnecessário.
d) a análise estática torna possível encontrar defeitos de runtime logo início do ciclo de desenvolvimento.
e) na análise de vulnerabilidades de segurança, a análise estática tem menos valor que os testes dinâmicos, já que estes são mais eficazes a localizar os defeitos do código.

P7 - Qual a definição mais adequada de "user story", tal como é usada nos testes em métodos ágeis?
a) Um artefato a preparar pelos os testers, para detalhar apenas os requisitos funcionais do sistema.
b) Um artefacto do projeto, a produzir pelos programadores e que o Product Owner deve aprovar antes que os testes possa começar.
c) Um artefato preparado pelos representantes do negócio/cliente para orientar os programadores e testers quanto às condições de aceitação do incremento.
d) Um artefato escrito colaborativamente pelos programadors, testers e especialistas do negócio para fixar os requisitos do produto.
e) Um artefacto, apresentado de forma breve, para documentar novas funcionalidades e bugs que precisam de ser corrigidos.

P8 - Segundo M. Fowler, qual das seguintes práticas NÃO é recomendada num sistema de Integração Contínua:
a) As builds que falham devem ser corrigidas de imediato.
b) A realização de uma build deve ser rápida e, por isso, excluir testes de aceitação.
c) Os testes devem ser feitos em ambientes específicos, que "clonam" as condições de produção.
d) Todos os programadores devem fazer entrega de código para o repositório partilhado, com regularidade (e.g.: pelo menos, diariamente).
e) Todos os membros da equipa têm acesso imediato ao feedback do estado das builds

P9 - O que é o pipeline de Entrega Contínua (continuous delivery)?
a) É um ficheiro de configuração que deve acompanhar o repositório Git, de modo a que as novas entregas sejam detetadas de imediato.
b) É uma implementação automatizada do processo de compilação, montagem, teste e instalação de uma aplicação.
c) É um processo automático para instalar uma aplicação num ambiente de Cloud (e.g.: usando containers).
d) É uma visualização possível da execução dos projetos (jobs) configurados no Jenkins, considerando diferentes etapas na construção (stages), que dependentes do sucesso das antecedentes.
e) É a configuração das regras de qualidade (do código) e do nível de cobertura necessários para que a build passe com sucesso.

P10 - Qual a hierarquia de elementos necessária na escrita de um pipeline declarativo do Jenkins, com inclusão de testes unitários?
a) Pipeline, agent, stages, stage, steps
b) Node, agent, stages, stage, steps, step
c) Pipeline, docker, stages, stage, post
d) Pipeline, agent, stage, junit, post
e) Node, stage, test, deploy

P11 - Considere a implementação da story de login, com sucesso, quando o utilizador já se encontra registado. O interface é baseado numa página web e acede aos serviços de retaguarda, invocando uma API. Que testes podem ser úteis, neste contexto?
a) Testes unitários, para validar se a interação com a página web envolvida.
b) Testes de regressão, para considerar os dados de autenticação das tentativas de acesso anteriores e encontrar vulnerabilidades.
c) Testes de sistema, para confirmar o comportamento do serviço de autenticação, em situações de utilização intensiva.
d) Testes de aceitação, para confirmar a conformidade das passwords com as regras de complexidade mínima definidas.
e) Teste unitário, para verificar o contrato dos métodos de cifra e validação de passwords.

P13. Os testes funcionais implementados com recurso a Selenium/WebDriver podem recorrer ao padrão "page object model" (POM). Qual das seguintes afirmação não é um benefício aplicável à utilização desse padrão?
a) O padrão POM permite escrever testes funcionais muito mais legíveis (do que na utilização "normal" dos scripts de teste com Selenium).
b) O padrão POM permite reduzir a duplicação de código e separar a navegação (entre páginas) da verificação.
c) Os testes ficam mais focados e curtos, já que podemos reutilizar classes (que representam páginas) em diferentes testes.
d) As alterações no UI são facilmente implementadas, pois a manutenção necessária dos testes está bem compartimentada no modelo (cada página tem a sua classe associada).
e) o padrão POM é diretamente suportado pelos métodos do WebDriver, que automatiza a instanciação dos objetos ("page objects"), à medida que são necessários no teste.

P16 - Qual o ciclo caraterístico de uma abordagem TDD? **ciclo do TDD: Red → Green → Refactor (unit 1st)**
a) Adicionar o novo incremento; escrever os testes unitários que o verificam; executar todos os testes; melhorar o código, se necessário.
b) Adicionar os testes relativos à user story em mãos; executar os testes e confirmar que falham; implementar o código necessário para fazer passar os testes; rever e aceitar o incremento.
c) Adicionar um teste; executar todos os testes e ver o novo a falhar; fazer as alterações necessárias para o teste passar; correr todos os testes e confirmar que passam; rever o código (refactoring).
d) Limpar o código anterior; adicionar um novo teste; implementar o código necessário e submeter no repositório partilhado; observar o feedback do sistema de CI.
e) Escrever os testes unitários no início da interação; implementar o código necessário para fazer passar os testes; escrever os testes de integração; fazer refactoring do código na medida necessária.

P17 - A existência de código duplicado indícia, geralmente, uma má prática. Que refactoring seria INADEQUADO para resolver este code smell?
a) os blocos de código duplicados ocorrem dentro da mesma classe: introduzir uma constante e atribuir-lhe o bloco de código comum.
b) o mesmo código é encontrado em duas subclasses do mesmo nível: extrair para um método, e colocá-lo num nível de hierarquia acima.
c) o mesmo código é encontrado em duas subclasses do mesmo nível: se o código duplicado estiver dentro de um construtor, colocar a parte comum num costrutor num nível acima.
d) se o código duplicado for encontrado em duas classes diferentes, extrair o bloco para uma superclasse nova e as classes mantém as funcionalidades anteriores.
e) se o código duplicado for encontrado em duas classes diferentes, mas não for natural criar uma única superclasse, criar uma classe adicional e mover para lá o bloco comum

P18 - A utilização de ambientes de mocking ajuda na utilização de objetos sintetizados, em substituição de objetos/serviços reais. Qual das afirmações NÃO é uma vantagem atribuível a estes ambientes?
a) Capacidade de retornar valores extremos, para exercitar condições limite.
b) Manter os testes unitários rápidos, isolados de potenciais latências (de serviços necessários).
c) Introduzir previsibilidade no resposta de um serviço remoto;
d) Fornecer uma implementação simplificada de um módulo atribuído a outro programador;
e) Facilitar a preparação das condições necessárias para o teste, através da definições das espetativas.

- **Pirâmide de testes** = organizar testes por **granularidade/custo**:
muitos unitários (rápidos e baratos) → alguns de integração → poucos E2E/aceitação/UI (lentos e frágeis).
- **Pirâmide "normal" (boa)**:
muitos unit tests, **alguns integration**, **poucos E2E**.
Vantagens: feedback rápido no CI, falhas fáceis de localizar, menos "flaky", dá confiança para refactoring.
- **Pirâmide invertida (má)**:
poucos unit tests e **muitos E2E/aceitação** (ex.: muito Selenium).
Problemas: testes **lentos**, **difíceis de manter**, **instáveis** (ambiente/UI/timing) e **difíceis de diagnosticar** (não se percebe onde está a causa).

Resumo

- **Unitário:** isolado, rápido, com mocks, causa fácil - validar lógica interna
- **Exemplos (Spring):** JUnit + Mockito (@ExtendWith(MockitoExtension.class), @Mock, @InjectMocks).

- **Integração:** componentes + infra (DB/rede), testa articulação. - **interação entre componentes**
- **Exemplos (Spring):**
    - @DataJpaTest (camada persistência + BD de teste)
    - @SpringBootTest (contexto completo)
    - Testcontainers para DB real em container.

- **Sistema:** sistema completo integrado (ambiente parecido a produção) - validar o comportamento global do sistema, **não** é "testar cada componente isoladamente".

- **Aceitação:** critérios da user story/negócio;
- **Forma comum:** BDD (Gherkin/Cucumber) e/ou E2E.
- **E2E/UI (Selenium):** via browser, caro/frágil, poucos.

- **Regressão:** objetivo (evitar que volte a falhar), não um tipo único.