Exame de Época de Recurso – 2019-07-02 15h00. Duração: 90min (+10 tolerância). A3

NOME: Nr.MEC:

Questões de escolha múltipla: **responda na grelha**, assinalando a opção (mais) verdadeira, exceto se a perguntar fornecer outra orientação. As não-respostas valem zero. **Respostas erradas descontam** ¼ da cotação; as respostas assinaladas de forma ambígua serão consideradas não-respostas. Questões 21 e 22: responder no espaço vazio, no final deste enunciado.

Grelha de resposta → Perguntas 1 a 20: escrever a opção de resposta em maiúscula, A, B, C, D, E, ou deixar vazio.

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	P14	P15	P16	P17	P18	P19	P20

P1.
Qual o propósito da utilização de um modelo de qualidade (como o ISO-25010)?

A. Garantir a integridade dos dados, o não repúdio das ações e a reutilização dos módulos.
B. Avaliar o grau até onde o conjunto de funcionalidades implementadas cobre a totalidade das necessidades e objetivos dos utilizadores.
C. Determinar um conjunto de características de referência que devem ser consideradas quando se avalia as propriedades de um produto de software.
D. Facilitar a integração (compatibilidade) entre sistemas, especialmente quando se trata de um modelo normalizado (e.g.: norma ISO).
E. Avaliar a facilidade com que os utilizadores de um sistema aprendem a usá-lo de forma eficiente e segura.

P2.
A metáfora da "pirâmide dos testes" (de Cohen) transmite a ideia de que:

A. O esforço da equipa com as atividades de teste é cumulativo e aumenta de iteração para iteração.
B. O número de testes diminui com o *burndown*, i.e., à medida que menos itens de trabalho subsistem no *backlog*, há menos testes para executar.
C. Os testes das camadas superiores devem usar os testes das camadas inferiores.
D. Existem diferentes classes de teste de software, que variam em quantidade e quanto aos seus objetivos.
E. Nas metodologias ágeis (associadas à metáfora da "pirâmide") as práticas de teste são o inverso do modelo tradicional ("V-Model").

P3.
Se quiser defender a adoção de uma abordagem BDD numa equipa, qual dos argumentos NÃO É correto?

A. Contribui para colocar o foco do teste nas necessidades do utilizador e não nos detalhes técnicos da implementação.
B. É suportada por uma linguagem natural que permite descrever o teste na área do domínio do problema, entendida pela equipa alargada.
C. 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.
D. Facilita a compreensão do comportamento esperado do software, ao fomentar a colaboração entre a equipa técnica e a equipa do "negócio".
E. Os testes unitários são executáveis e substituem a necessidade de criar especificações de requisitos adicionais.

P4.
Uma estratégia de garantia de qualidade deve incluir testes de caixa-aberta e testes de caixa-fechada.

A. A análise estática de código é um exemplo de uma estratégia de testes de caixa-aberta.
B. Os testes unitários são testes de caixa-aberta que exercitam os caminhos internos de um módulo.
C. O nível inferior da "pirâmide de testes" é genericamente caraterizado por testes de caixa-fechada, enquanto que o nível superior por testes de caixa-aberta.
D. Os testes funcionais são testes de caixa-fechada, eficazes para localizar a origem dos erros no código.
E. O conhecimento da implementação interna é fundamental para escrever testes funcionais eficientes.

P5.
O nível de cobertura reportado no SonarQube para o Projeto XYZ é de 70%. Isso significa que:

A. A dívida técnica do projeto é de 30%.
B. Não se deve avançar para a produção; é altamente desejável obter um nível de cobertura de 100%.
C. A maior parte do código do projeto foi exercitado nos testes disponíveis, mesmo que repetidamente.
D. 70% dos testes executados passaram, e 30% não estão a passar.
E. A análise estática determinou que a probabilidade de não existirem problemas no código é de 70%.

P6.
O que é a "*user story*", tal como é usada nos processos de Garantia de Qualidade em metodologias ágeis?

A. Artefacto usado para monitorizar o progresso, identificado e priorizado por equipas interdisciplinares.

B. Artefacto preparado pelos *testers*, para detalhar os requisitos funcionais do sistema e os critérios de aceitação.
C. Um artefacto do projeto, a produzir pelo *Product Owner* para fazer a aceitação dos incrementos entregues pelos programadores.
D. Um artefacto preparado pelos representantes do negócio/cliente para apresentarem as histórias que pretendem realizar no sistema.
E. Um artefacto, apresentado de forma breve, para documentar novas funcionalidades a desenvolver na presente iteração/sprint.

P7.

Qual das seguintes práticas NÃO ESTÁ de acordo com as dez recomendadas por M. Fowler, relativamente à preparação de um sistema de Integração Contínua:

A. As *builds* que falham devem ser corrigidas de imediato.
B. O projeto deve manter um repositório para *branches* de desenvolvimento e um repositório para integração (*master*).
C. Devem existir teste feitos em ambientes que mimetizam as condições de produção.
D. Todos os membros da equipa têm acesso imediato ao *feedback* do estado das *builds*.
E. Deve existir formas automáticas de fazer as instalações, para montar os ambientes de teste e, quando for o caso, de produção.

P8.

A expressão "*pipeline as code*", associada, por exemplo, às versões mais recentes do Jenkins, significa que:

A. Um *pipeline* de CI inclui a compilação de código fonte e produz artefactos são partilhados na equipa.
B. Os passos do *pipeline* são executados quando é entregue novo código fonte no repositório associado.
C. A definição do *pipeline* é escrita numa linguagem própria e o esse ficheiro sujeito ao controlo de versões, no repositório de código.
D. É possível executar condicionalmente algumas etapas do *pipeline*, fazendo-as depender do sucesso das antecedentes.
E. O *pipeline* é compilado, como o resto do código fonte, na realização das tarefas da ferramenta de montagem (e.g.: Maven).

P9.

Qual a hierarquia de elementos necessária na escrita de um *pipeline* declarativo do Jenkins, com inclusão de testes de integração?

A. Pipeline, docker, stages, stage, post
B. Pipeline, agent, stages, stage, steps
C. Pipeline, agent, stage, mock, post
D. Node, agent, stages, step, springboottest
E. Node, stage, test, deploy

P10.

Considere a história: "*SENDO* um Viajante, *QUERO* reservar um quarto para ficar no período de tempo indicado, *DE MODO A* não me preocupar mais em procurar sítio." Que situação de teste NÃO É indicada/relevante para a validação da funcionalidade?

A. Testes unitários, para verificar o comportamento da operação de reserva do módulo gestor de reservas.
B. Testes funcionais, para confirmar que o viajante consegue, na página web, selecionar as datas e inserir a reserva.
C. Testes de aceitação, para confirmar que as mensagens presentes ao Viajante, na página, são claras e informativas.
D. Testes de aceitação, para confirmar que o Viajante obtém *feedback* claro e informativo, no caso de a marcação não poder ser satisfeita.
E. Testes de regressão, para verificar que as novas reservas não colidem com reservas já existentes.

P11.

Considere a informação da Figura 2. Assumindo que a solução assume as práticas comuns de desenvolvimento e teste com Spring Boot, assinale a afirmação FALSA:

A. O componente sob teste é o RegisterRestController.
B. O componente RegisterUseCase não é usado no teste.
C. O teste "*whenValidInput…*" insere uma entidade e verifica que o serviço *registerUser* foi invocado exatamente uma vez.
D. Apesar de o teste usar o protocolo HTTP como consumidor de serviços, não inicia verdadeiramente um servidor web com a aplicação.
E. O teste verifica o (código de) estado e o conteúdo da resposta do pedido HTTP para avaliar o comportamento esperado do componente.

P12.

Os testes funcionais implementados com recurso a Selenium/WebDriver podem recorrer ao padrão "*page object model*" (POM). Em que consiste este padrão?

A. A ferramenta Maven utiliza a modelo do POM para diferenciar entre testes unitários e de integração, executado os testes "Selenium" neste último grupo.
B. O padrão POM utiliza exemplos escritos em linguagem do domínio (*features*) para alimentar a execução dos "testes Selenium".
C. Cada página web tem um controlador (da interação) associado, sendo possível mapear os campos na página com os atributos do controlador.
D. A lógica de interação sob uma página, através de WebDriver, deve ficar estruturada em métodos de uma classe, que representa essa página nos testes.
E. A programação da página web deve usar uma linguagem por objetos e um modelo de interação MVC, facilitadora dos testes.

P13.

Quais as regras que deve observar uma equipa que adotou o "GitHub flow", na utilização do repositório partilhado?

A. Trabalhar sobre o *branch* "master", com *commits* frequentes; manter um *branch* adicional para registar as *releases* do produto.
B. Criar um novo *branch* para cada programador; fazer alterações no seu *branch* "privado"; integrar os incrementos no *master* partilhado.
C. Criar um novo *branch* local para cada *feature*; adicionar as alterações; integrar no *master* partilhado para revisão pelos pares.
D. Criar um novo *branch* por *feature*; desenvolver a *feature* no respetivo *branch*; integrar no *master*; manter um *branch* adicional para as *releases* em separado.
E. Criar um *branch* para cada nova *feature*; fazer alterações neste *branch*, com *commits* regulares; integrar o incremento no *master*, mediante um pedido de integração ("*pull request*").

P14.

Considere o trecho de código apresentado na Figura 2. Considere que o teste é executado sem erros, mas que falha (i.e., o teste corre, mas não passa). Que instruções podem sinalizar ao *framework* de teste que as condições exercitadas não "passam"?

A. As instruções das linhas 47 e 48.
B. As instruções das linhas 45, 47 e 48;
C. As instruções das linhas 25, 28, 31, 47 e 48.
D. As instruções das linhas 38, 45, 47 e 48.
E. As instruções das linhas 31, 45, 47 e 48.

P15.

Qual o ciclo caraterístico de uma abordagem BDD?

A. Adicionar o novo incremento; escrever os testes funcionais que o verificam; executar todos os testes; melhorar o código, se necessário.
B. 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*).
C. Descrever o comportamento esperado com exemplos; escrever um teste com os passos correspondentes aos do exemplo; verificar que falha; fazer as alterações para o teste passar; reformular o código (*refactoring*).
D. Escrever uma história (*user story*); adicionar um novo teste; implementar o código da funcionalidade; observar o *feedback* do sistema de CI.
E. Escrever os testes no início da iteração; verificar que estão a falhar; implementar o código necessário para fazer passar os testes; fazer *refactoring* do código na medida necessária.

P16.

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. Conveniência para retornar valores limite dos parâmetros para exercitar diferentes percursos no módulo sob teste.
B. Manter os testes unitários rápidos, isolados de potenciais latências (de serviços necessários).
C. Introduzir previsibilidade na resposta de um serviço remoto;
D. Facilitar a preparação das condições necessárias para o teste, através da definição das espectativas.
E. Contar o número de vezes que o teste é executado (com cláusulas de verificação).

P17.

Uma utilização natural do *framework* Mockito no teste de um componente Spring Boot (SB) anotado com *@Service*, seria:

A. Sintetizar um objeto do tipo repositório, substituindo o acesso concreto à base de dados, por respostas predefinidas.
B. Substituir o contexto do servidor web por uma versão mais simples no teste, acelerando a execução.
C. Viabilizar a injeção das dependências necessárias para realizar o teste, automaticamente satisfeitas pelo SB.
D. Avaliar a resposta dos métodos da API REST sob teste, por exemplo, quanto ao código de retorno bem-sucedido.
E. Viabilizar testes de integração, em conjunto com a anotação @SpringBootTest, executados num contexto da aplicação especial para os testes.

P18.

Os testes unitários são uma peça importante numa estratégia de garantia de qualidade, no entanto, também se podem observar falácias (*pitfalls*) na sua utilização, tais como:

A. A escrita de testes unitários à cabeça ajuda a entender o contrato do módulo que se vai construir;
B. Os testes unitários ajudam a revelar erros de regressão no novo código;
C. Quanto maior o número de testes unitários, mais confiança podemos ter na *build.*
D. Contribuem para documentar a utilização esperada de um componente, facilitando a manutenção;
E. Aumentam a confiança da equipa para executar operações de reformulação do código (*refactoring*) com frequência.

P19.

Qual a interpretação mais adequada do conceito "dívida técnica" (*technical debt*), usado em ambientes de qualidade?

A. É a diferença do nível de cobertura atual para os 100% de cobertura.
B. É a diferença da velocidade atual da equipa (medida como a média de pontos aceites nas últimas 3 iterações) para a velocidade estabelecida como objetivo.
C. É o número de *user stories* não aceites na iteração corrente, por não passarem os testes definidos.

D. É uma estimativa do tempo de trabalho necessário para fazer passar os testes que estão a falhar.
E. É uma estimativa do tempo necessário para corrigir os problemas encontrados durante a análise estática.

P20.

Relativamente à utilização da exceção NullPointerException nos contratos dos métodos da Figura 3:

A. É adequada, visto que os utilizadores do módulo (i.e., programadores) devem assegurar a verificação dos parâmetros antes de chamar os métodos.
B. É desadequada, porque não dá ao programador que usa o módulo a possibilidade de receber e tratar a condição de exceção.
C. É adequada, visto que as exceções "checked" são sempre incluídas na documentação do módulo, alertando o programador para essa eventualidade.
D. É desadequada, uma vez que não acrescenta valor ao módulo, já que na impossibilidade de retornar valores encontrados, os métodos podem retornar "null".
E. O módulo seria mais fácil de usar se os métodos lançassem exceções especializadas, com a semântica detalhada da condição verificada (e.g.: KeyNotFoundException)

P21. [Opcional]

(Esta questão é facultativa: não é necessária para obter a cotação máxima do exame, mas pode somar até 1 valor adicional no grupo de escolha múltipla.)

Considere a informação da Figura 1. Explique:

a) A natureza do "*code smell*" reportado;
b) O que fazer para reformular o código ("*refactoring*") de modo a resolver este problema.

P22. [desenvolvimento]

Considere que está encarregue de orientar um programador júnior que deve implementar uma *cache* (Figura 3). É um módulo que a equipa vai usar em diferentes aplicações Spring Boot que está a desenvolver, usando a metodologia TDD.

a) Enumere alguns dos casos de teste que recomendaria ao programador implementar para verificar o comportamento do módulo IMyCache. Nessa perspetiva de tutor, explique como podem ser implementados (tecnologias) e potenciais dificuldades.
b) Exemplifique, em código, dois dos testes mais relevantes e que sejam específicos do comportamento deste componente (uma *cache*). O código não precisa de estar compilável, mas deve ser possível observar a correta utilização das construções próprias dos testes, e a lógica dos mesmos.

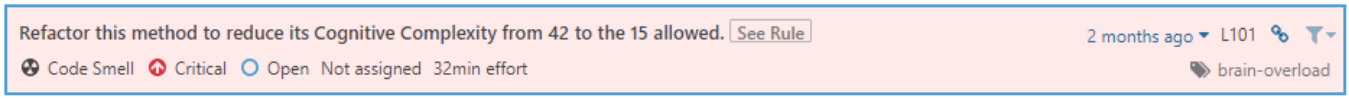
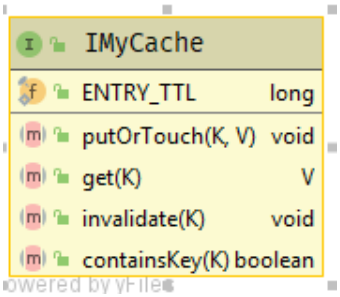


Figura 1: Detalhe da dashboard do SonarQube.

```
@WebMvcTest(controllers = RegisterRestController.class)
class RegisterRestControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper objectMapper;

    @MockBean
    private RegisterUseCase registerUseCase;

    @Test
    void whenValidInput_thenMapsToBusinessModel() throws Exception {

        UserResource user = new UserResource( name: "Zaphod", email: "zaphod@galaxy.net");

        mockMvc.perform(post( urlTemplate: "/forums/{forumId}/register", ...uriVars: 42L)
                .contentType("application/json")
                .param( name: "sendWelcomeMail", ...values: "true")
                .content(objectMapper.writeValueAsString(user)))
                .andExpect(status().isOk());

        ArgumentCaptor<User> userCaptor = ArgumentCaptor.forClass(User.class);
        verify(registerUseCase, times( wantedNumberOfInvocations: 1)).registerUser(userCaptor.capture(),
                eq( value: true));
        assertThat(userCaptor.getValue().getName()).isEqualTo("Zaphod");
        assertThat(userCaptor.getValue().getEmail()).isEqualTo("zaphod@galaxy.net");

    }
```

Figura 2: Código relativo a um teste de um projeto que usa o framework Spring Boot.



Interface IMyCache<K,V>

Type Parameters:

K - Keys type

V - Values type

Method Detail

putOrTouch

```
void putOrTouch(K key,
                V value)
```

Inserts the element in the cache and sets the last access time to now. If already existing, updates the last access time to now.

Parameters:

key - Identifier of the new element

value - Data to be stored in cache

Throws:

java.lang.NullPointerException - if the key is null

java.lang.IllegalArgumentException - if the value is null

get

```
V get(K key)
```

Gets an entry from the cache and updates the last access time.

Parameters:

key - ID of the desired cached element

Returns:

Data stored in cache associated with the specified key or a not found exception

Throws:

java.lang.NullPointerException - if the key is null

invalidate

```
void invalidate(K key)
```

Removes an entry from the cache.

Parameters:

key - ID of the element to be removed from the cache

Throws:

java.lang.NullPointerException - if the key is null

containsKey

```
boolean containsKey(K key)
```

Determines if the cache contains an entry for the specified key.

Parameters:

key - ID of the desired cached element

Returns:

True if the specified key is present in cache, False if not

Throws:

java.lang.NullPointerException - if the key is null

Figura 3: Interface para um módulo de Cache.

TQS - Exame (E. Beunso.)

P1 (c)
P2 (d)
P3 (e)
P4 (b)
P5 (c)
P6 (a)
P7 (b)
P8 (e)
P9 (b)
P10 (e)
P11 (e)
P12 (d)
P13 (e)
P14 (d)
P15 (e)
P16 (e)
P17 (a)
P18 (c)
P19 (e)
P20 (e)