Exame de Época de Recurso – 2016-07-12. Duração: 90min.

Responda de forma objetiva às perguntas colocadas, em folha devidamente identificada.

Q1.

"O software é frágil, como é natural com os sistemas discretos. Uma mudança aparentemente simples pode resultar em consequências não intencionais, simpaticamente conhecidas por *bugs*. Um único bit errado pode causar desastres. O software é complexo; as pessoas que programam computadores cometem erros. Os erros podem passar despercebidos por longos períodos de tempo, mas acabam por se revelar como *bugs*. Devido a estas realidades do desenvolvimento de software, precisamos testar de novo o software de cada vez que há uma alteração." J. W. Grenning, "Scenario Testing with Executable Use Cases," in *Embedded Systems Conference*, 2012.

a) O autor sublinha a importância de testar de novo [todo] o software sempre que há incrementos num projeto, com o objetivo de garantir que o novo código não provoca consequências não pretendidas no que existia.
   Que nome tem esta prática? A sua adoção sistemática, ao longo da vida de um projeto, é viável? Justifique.
b) Um erro pode passar desapercebido por um período mais ou menos extenso, até que se manifeste, podendo causar "desastres".
   Que práticas mais podem contribuir para encurtar o tempo que leva desde a introdução/injeção do problema, até que seja identificado?

Q2.

Responsabilidades essenciais

O engenheiro de Qualidade de Software (SQE) será responsável por:

1. Desenhar planos de testes, cenários, *scripts* e procedimentos;
2. Planear o agendamento de testes de acordo com o âmbito do projeto ou datas de entrega:
3. Executar os testes de acordo com as T-Specs e analisar o resultado.
4. Especificar os requisitos, orientações e processos para determinar a qualidade do produto ou prontidão para lançamento e documentar as especificações dos testes;
5. Desenhar ou desenvolver ferramentas de teste automático e *scripts* para automatizar testes para melhorar o processo de teste do software.
6. Documentar defeitos do software utilizando um sistema de rastreio de *bugs* e reportar defeitos aos *developers* do software.

Extrato (traduzido) de uma oferta de emprego publicada no Facebook do curso de MIECT em 21/5/2016.

a) Tendo presente o ponto 1, descreva uma abordagem metodológica para a identificação dos cenários de teste a implementar.
b) Que abordagem recomendaria relativamente ao ponto 4, tendo em vista a disponibilidade de indicadores para avaliar o estado de prontidão de uma solução para lançamento (para produção)?
c) Que ferramentas poderiam ser consideradas no ponto 5, para automatizar a realização de testes funcionais de uma aplicação Web, em Java EE?

Q3.

O SonarQube é um ambiente de análise estática de código, cuja análise pode ser inserida num processo de integração contínua, por exemplo, através de *plug-ins* do Jenkins.

a) Explique o sentido da afirmação: "a análise estática de código contribui para encontrar defeitos invisíveis".
b) O SonarQube utiliza uma métrica designada por "dívida técnica" (*technical debt*). Explique em que consiste.
c) O que significa a métrica cobertura de código (*code coverage*), tal como é utilizado no SonarQube?

Q4.

A implementação de práticas de revisão colaborativa de código (*code review*) numa equipa de desenvolvimento é exigente e, por vezes, pode revelar-se mais prejudicial que benéfica.

a) Identifique riscos potenciais que podem levar a que os benefícios da revisão de código não sejam conseguidos.
b) Como é que a revisão de código pode ser operacionalizada numa equipa que utiliza o sistema Git para controlo de versões? Certifique-se que dá uma explicação concreta, apresentando ferramentas relevantes.

Q5.

```
31  @RunWith(CukeSpace.class)
32  @Features({"src/test/resources/it/feature/date_conversion.feature"})
33  public class DateConversionFeatureIT {
34
35      @Deployment
36      public static JavaArchive createArchiveAndDeploy() {
37          return ShrinkWrap.create(JavaArchive.class)
38                  .addClasses(LocaleManager.class, TimeService.class)
39                  .addAsManifestResource(EmptyAsset.INSTANCE, "beans.xml");
40      }
41
42      @Inject
        TimeService timeService;
44
45      User user;
46      Date rawDate;
47
48      @Given("^a user named '(.+)'$")
49      public void create_user_with_name(final String name) throws Throwable {
50          user = new User(name);
51      }
52
53      @When("^this user enters the date '(.+)' into the time conversion service$")
54      public void user_enters_the_date(@Format("yyyy-MM-dd HH:mm:ss") final Date date)
55              throws Throwable {
56          rawDate = date;
57      }
58
59      @Then("^the service returns a conversion hint with the message '(.*)'$")
60      public void service_returns_a_converted_date(final String dateConverted) throws Throwable {
61          assertThat(timeService.getLocalizedTime(rawDate, user), equalTo(dateConverted));
62      }
63  }
```

O excerto de código apresenta um teste de integração para Java EE.

a) No contexto do *framework* de testes Arquillian, qual o propósito do método anotado com @Deployment [linhas 35ss] neste teste?

b) Distinga entre os três modos de gestão do ciclo de vida do *container* Java EE suportados no *framework* Arquillian. Qual desses modos está a ser utilizado neste exemplo?

c) Este teste utiliza um recurso "date_conversion.feature" [linha 32], necessário para a execução do teste. Exemplifique um conteúdo viável para este ficheiro, de modo a que este teste possa ter sucesso.

d) O teste apresentado incorpora as recomendações do *Behaviour-Driven Development* ou não? Justifique.

Q6.

A classe TqsPatientsQueue gere uma fila de espera para atendimento de doentes numa unidade de cuidados agudos (e.g.: urgência de um Hospital), seguindo o sistema de triagem de Manchester. Neste sistema, os doentes recebem um código de prioridade no momento em que são triados (a TqsPatientsQueue é uma fila com prioridade).

A estrutura TqsPatientsQueue permite gerir a fila de espera de acordo com a prioridade de triagem, em que os mais urgentes passam à frente dos menos urgentes (do vermelho para o azul).

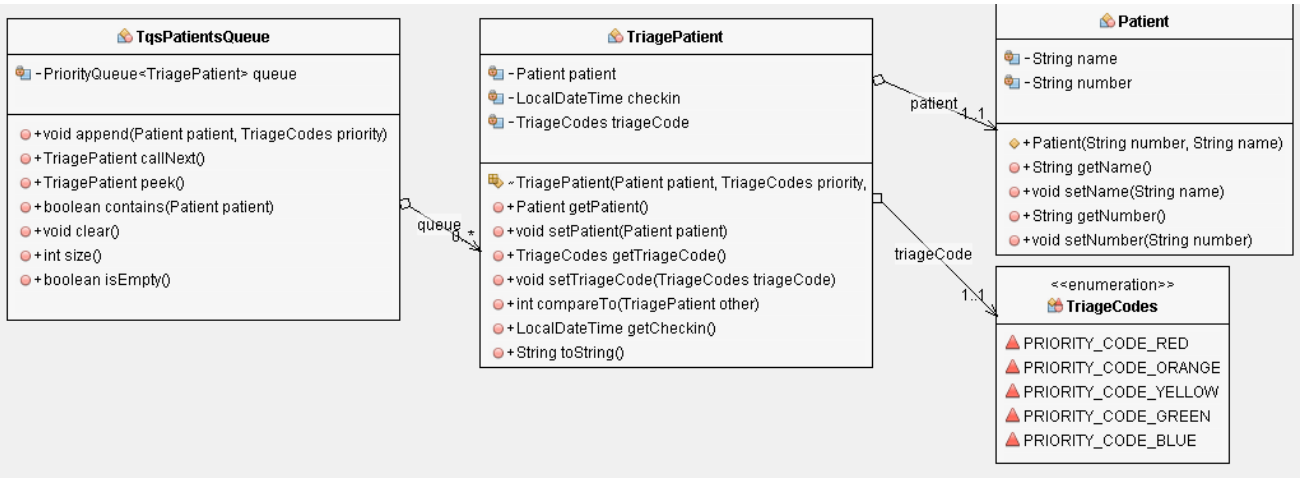
a) No contexto de uma abordagem TDD, indique um conjunto de testes unitários que o programador deveria considerar para confirmar a implementação do contrato de TqsPatientsQueue.

Nota: as alíneas b), c) e d) devem ser respondidas num trecho de código comum.

b) Crie um programa de teste recorrendo ao *framework* JUnit e utilize a anotação @Before para preparar dois objetos TqsPatientsQueue, um vazio, outro com valores exemplificativos (que devem ser usados nas alíneas a seguir).

c) Escreva um teste para verificar o comportamento do TqsPatientsQueue quando se pretende inserir um utente duplicado na lista de espera.

d) Selecione dois testes adicionais referidos em a) mais pertinentes para esta classe e escreva a sua implementação.



Detalhe de alguns métodos da classe TqsPatientsQueue:

```
public void append(Patient patient,
                   TriageCodes priority)
```

Add a patient to the waiting queue, assigning a triage priority.

Parameters:

patient - the patient

priority - the triage code

Throws:

InvalidParameterException - if the patient is already present in the waiting queue

```
public TriagePatient callNext()
```

Removes the patient in the top of the waiting queue, observing the triage rules

Returns:

the patient, with priority and the check-in time.

Throws:

IllegalStateException - if the waiting queue is empty

```
public TriagePatient peek()
```

Sees who is next, without removing from the waiting queue

Returns:

the next patient in queue, with priority and the check-in time.

Throws:

IllegalStateException - if the waiting queue is empty

```
public boolean contains(Patient patient)
```

verifies if the patient is included in the queue

Parameters:

patient - the patient to look for

Returns:

true if present, false otherwise