Teste 2018/2019

1. O método Unified Process prevê quatro fases principais no desenvolvimento do projeto, cada qual com um objetivo a atingir (milestone) para se poder avançar, por esta ordem:

Phases:

- Inception
  - Establish
  - Prepare a preliminary project schedule and cost estimate
  - Feasibility
  - Buy or develop it
- Elaboration
  - Capture a healthy majority of the system requirements
  - Address known risk factors and to establish and validate the system architecture
- Construction
  - System features are implemented in a series of short, time-boxed iterations
  - Each iteration results in an executable release of the software
  - It is customary to write full-text use cases during the construction phase and each one becomes the start of a new iteration
- Transition
  - In this phase the system is deployed to the target users
  - The Transition phase also includes system conversions and user training

"1/ Decisão de avançar ou parar o projeto; 2/ Arquitetura técnica definida e validada; 3/ Funcionalidades da primeira versão do produto implementadas; 4/ Solução instalada e aceite pelo cliente."
Option: e)

2.
Uma das principais razões para se utilizar métodos ágeis de desenvolvimento, em detrimentos dos métodos sequenciais, é a diminuição do risco do projeto. A entrega frequente de valor diminui a probabilidade de eventuais divergências na perceção dos requisitos. Que prática é decisiva para a mitigação do risco?

"A entrega frequente de valor diminui a probabilidade de eventuais divergências na perceção dos requisitos."
Option: e)

3. Os requisitos devem apresentar as características conhecidas por S.M.A.R.T. Identifique, na lista, um requisito não-funcional adequadamente formulado.

S.M.A.R.T Objectives

- Specific – target a specific area for improvement.
- Measurable – quantify or at least suggest an indicator of progress.
- Assignable – specify who will do it.
- Realistic – state what results can realistically be achieved, given available resources.
- Time-related – specify when the result(s) can be achieved.

Types of requirements:

- Functional - relates to a process or data
- Non-functional - relates to a system quality

“A resposta com a autorização por parte do sistema de micro-pagamentos não pode demorar mais de 2 segundos.”

Option: e)

4. Os casos de utilização e os requisitos funcionais descrevem um sistema segundo uma perspetiva de “caixa opaca”. Qual a relação que se aplica entre estas duas técnicas?

“Os casos de utilização captam os requisitos funcionais de forma contextualizada, nas descrições estruturadas dos cenários de interação.”

Option: c)

5. Os conceitos de Ator e Persona são usados na análise de sistemas para descrever cenários de uso de um sistema.

An actor is defined as an entity that plays a specific role within a system. Conceptually, actors in systems and products are not very different from actors in a theater. In both cases, an actor can play one role in one situation and another role in another. Actors play roles to accomplish some outcome which make them valuable for defining and documenting use cases. Actors are less useful if they are used for eliciting and documenting user stories.
Agile requirements (typically documented as user stories) use the concept of personas to identify interaction with an application or product. A persona represents one of the archetypical users that interacts with the system or product. Persona are far more detailed and structured to connection between the team and the persona. Actors are typically defined as a title. Persona are used in user stories and generally are more robust than actors. A persona is designed to help the team understand who they are developing a system or product. A persona, in the word's everyday usage, is a social role or a character played by an actor.

“A Persona é uma instância/concretização de um Ator.”

Option: b)

6. A arquitetura trata da tomada das grandes decisões técnicas em relação ao sistema a desenvolver, tendo em conta os atributos de qualidade pretendidos. Um exemplo de um assunto/decisão de arquitetura é:

- Especificar os cenários de interoperação com sistemas externos e as tecnologias selecionadas para os implementar.
- Definir estratégias de distribuição de carga para garantir a disponibilidade do sistema em utilização contínua, com 1000 sessões simultâneas.

- Definir os mecanismos técnicos para separar a informação pessoal da informação operacional, para dar resposta aos requisitos previstos no RGPD.

“Todas as opções anteriores são corretas.”

Option: d)

7. Os padrões de desenho fornecem soluções conhecidas para problemas de programação comuns. Alguns exemplos de padrões relacionados com a criação de objetos (creational) são?

In software engineering, creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation.

Examples of patterns:

- Abstract factory pattern
- Builder pattern
- Factory method pattern
- Prototype pattern
- Singleton pattern

“Abstract Factory, Factory Method, Singleton.”

Option: a)

8. Um dos princípios “SOLID” para o desenho por objetos, preconiza a segregação das interfaces. Como é que esse princípio deve ser usado pelo programador?

SOLID is a mnemonic acronym for five design principles intended to make software designs more understandable, flexible and maintainable.

- Single responsability principle - a class should have only one responsability
- Open/closed principle - software entities should be open for extension, but closed for modification.
- Liskov substitution principle - objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program.
- Interface segregation principle - many client-specific interfaces are better than one general-purpose interface.
- Dependency inversion principle - one should "depend upon abstractions, [not] concretions."

“Criar interfaces específicas, evitando que os clientes dependam de interfaces que não utilizam.”

Option: b)

9. No cerne da abordagem TDD (test-driven development), está um ciclo de trabalho com um fluxo característico. Como se desenrola o trabalho do programador?

Test-driven development cycle:

1. Add a test
2. Run all tests and see if the new test fails
3. Write the code
4. Run tests
5. Refactor code

“Adicionar um pequeno teste; executar os testes e verificar que o novo está a falhar; implementar as alterações suficientes para o teste passar; executar os testes e verificar que o novo passa; rever o código para o tornar mais claro.”

Option: a)

10. Uma forma comum de alinhar as histórias do utilizador (user stories) e os métodos de garantia de qualidade do software é?

Uma forma comum de alinhar as histórias do utilizador (user stories) e os métodos de garantia de qualidade do software é:

“As histórias são suplementadas com a descrição de cenários concretos, que estabelecem as condições de aceitação do incremento.”

Opção: e)

11. Qual das seguintes sequências de passos deve ocorrer num processo de Integração Contínua?

- Continuous Integration is the practice of merging all developer working copies to a shared mainline several times a day.
- The main aim of CI is to prevent integration problems.

Practices:

1. Developers commit to a shared repository regularly
2. Changes in SCM are observed and trigger automatically builds
3. Immediate feedback on build failure (broken builds have high-priority)
4. Optional: deploy of artifacts into a reference repository
5. Optional: trigger deployment for integration/acceptance tests
6. Sharing repositories

“Entrega de código (commit), resolução de dependências e compilação no ambiente de integração, execução dos testes automáticos, disponibilização de feedback quanto ao estado da build.”

Option: a)

12. Na terminologia dos projetos de desenvolvimento ágil, o que é a velocidade da equipa (numa iteração)?

“Os pontos acomulados das histórias implementadas, por iteração.”

Option: d)

13. O modelo do domínio é construído na Análise, para mapear os conceitos do universo de discurso.

“Os conceitos identificados no modelo do domínio serão candidatos naturais a constituir classes de programação, numa linguagem por objetos.”

Option: c)

14. Qual o papel característico do Analista numa equipa de desenvolvimento?

“Analisa os processos da organização para identificar oportunidades de melhoria, e delineia o sistema de informação que as implementa.”

Option: c)

15. O Diagrama 1 utiliza o conceito de estereótipo da UML (stereotype) na relação “extend”.

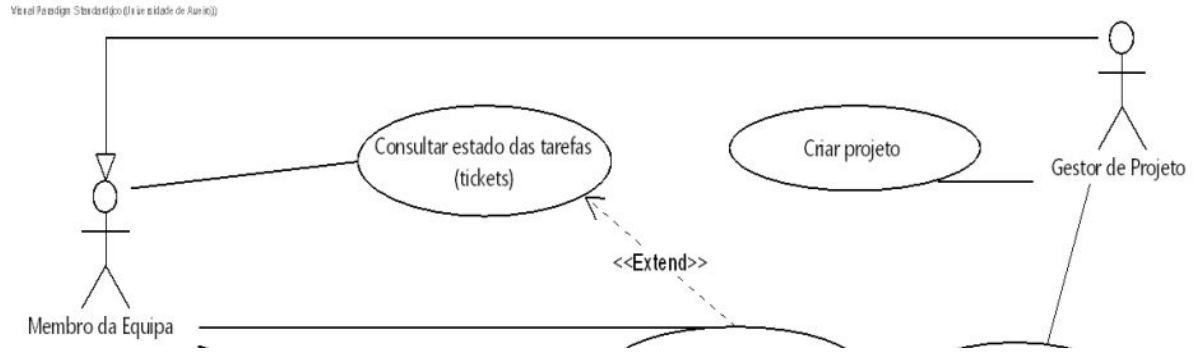
Stereotype is a profile class which defines how an existing metaclass may be extended as part of a profile. It enables the use of a platform or domain specific terminology or notation in place of, or in addition to, the ones used for the extended metaclass.

- ➔ “Configurar módulos do projeto” e “Configurar a equipa” são formas opcionais de complementar o fluxo do caso de utilização “Configurar projeto”

Option: a)

16. Relativamente ao modelo representado no Diagrama 1:

O “Gestor de Projeto” também é um “Membro da Equipa”.



Option: e)

17. Nos elementos modelados no Diagrama 1 há um que está desajustado e carece de revisão.

➔ O caso de utilização “Notificações de progresso” não evidencia nenhum fluxo.

Option: c)

18. Considere a capacidade expressiva do Diagrama 2:

“Um cliente pode ter um anúncio afixado em diferentes Outdoors.”

Option: e)

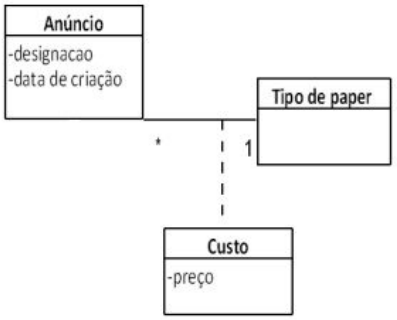
19. Que alterações seria necessário fazer ao Diagrama 2 para captar o requisito: “Um funcionário faz a inspeção de Outdoors da sua zona de atuação.”

“Deve ser introduzida uma classe Zona, associando-a a Funcionário e a Outdoor.”

Option: c)

20. Relativamente ao Diagrama 2:

“O custo de um anúncio depende do tipo de papel.”



Option: d)

21. Relativamente ao Diagrama 3:

"Pode ser um subdiagrama associado a uma classe Perfil ("User account")"

Option: a)

22. O Diagrama 3 descreve o ciclo de vida de um perfil de utilizador num site de vendas on-line.

➔ Todas os perfis podem ser Encerrados ("Closed").

Option: b)

23. Relativamente ao tipo de diagrama ilustrado no Diagrama 3 e à sua relação com outros diagramas da UML:

"Os estados representados podem ser mostrados num diagrama de atividades, como estados de uma entidade de dados (associada a um object flow)."

Option: a)

24. Várias abordagens ao desenvolvimento de software partem do estudo de cenários de utilização como estratégia para determinação de requisitos.

a) Qual é o benefício de utilizar técnicas de determinação de requisitos baseadas em cenários de utilização (por contraponto à decomposição funcional)? São adequadas para todos os tipos de projetos? Justifique.

As especificações baseadas em cenários valorizam o levantamento objetivos dos atores (o que querem alcançar ao usar o sistema), em vez do foco no produto.

Vantagens:

- Os cenários apresentam os requisitos funcionais em contexto;
- Evitam a especificação de funcionalidades órfãs;
- A técnica de "story telling" dos cenários facilita a participação dos stakeholders na especificação e validação.

As técnicas baseadas em cenários de utilização são adequadas a sistemas com interação com utilizadores finais. São menos adequadas (e insuficientes) para sistemas com componentes não-interativos relevantes, como sistemas embebidos, computação intensiva, etc…

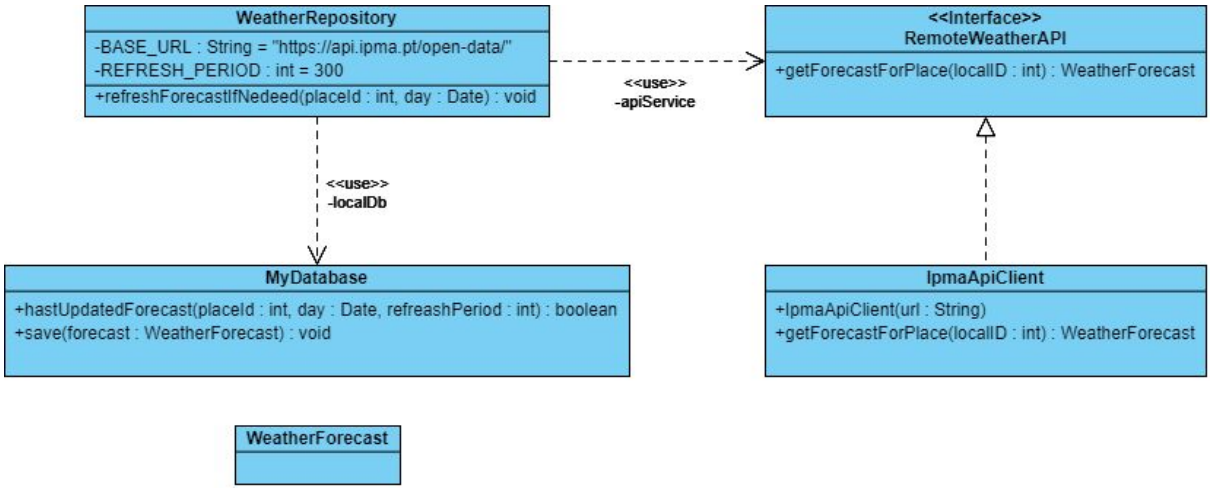
b) Compare a utilização dos casos de utilização (use cases) e das histórias do utilizador (user stories) enquanto técnicas de especificação, destacando os aspetos comuns e diferenciadores entre elas.

Os UC são uma técnica adequada à descrição de alto nível, para construir a big-picture na Análise; incorporam a documentação dos cenários de forma estruturada, para caracterizar a interação esperada.
As US são mais granulares, focadas em pequenos incrementos com valor para o negócio, adequadas à gestão diária do backlog; a adição de detalhe não é feita de início, mas perto de quando for preciso, com a participação regular de “domain experts".
UC e US usam diferentes níveis de abstração, mas não se contradizem; podem ser usadas em complementaridade; US podem ser deduzidas dos cenários dos CU.

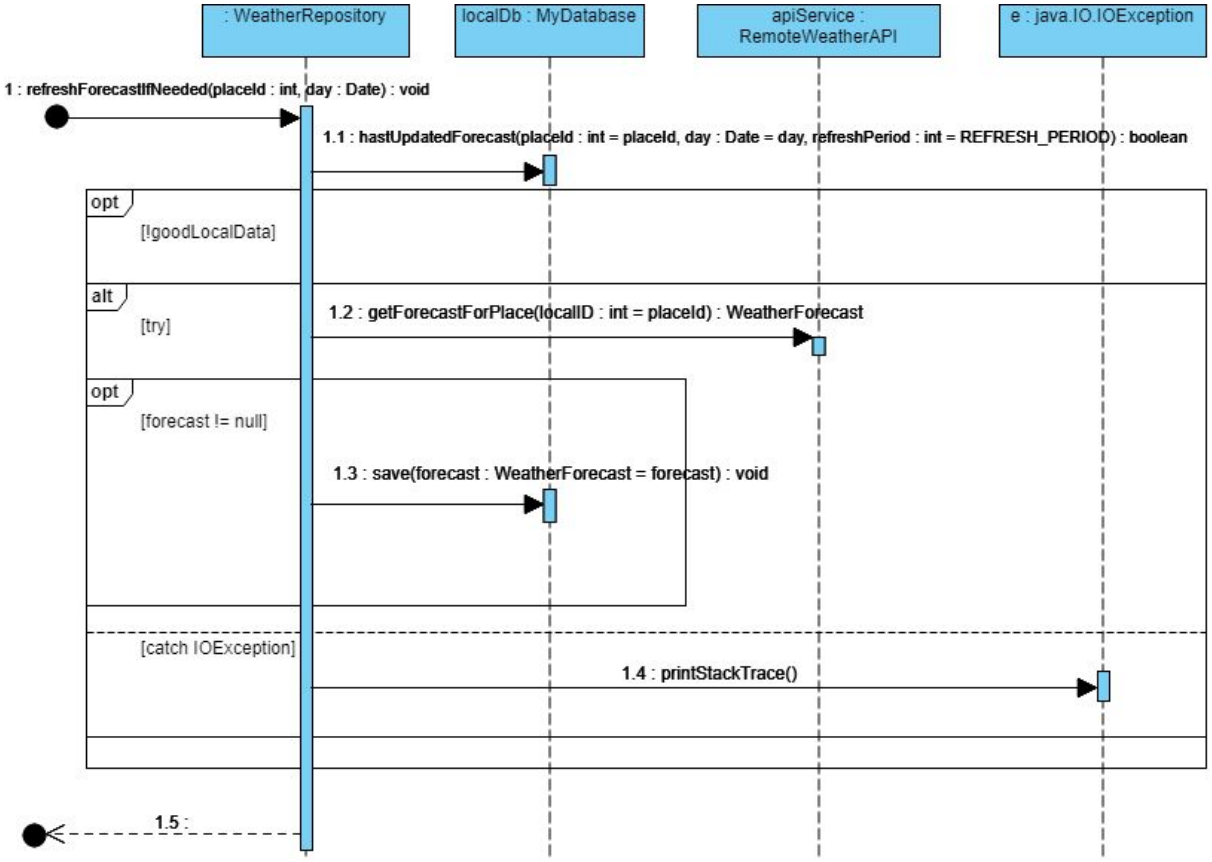
25.

```java
/**
 * Manages the requests to get the weather forecast. If the local data
 * is still recent, no remote requests are made. Otherwise, the IPMA's
 * API is invoked.
 */
public class WeatherRepository {
    private static final String BASE_URL = "https://api.ipma.pt/open-data/";
    private static final int REFRESH_PERIOD = 300;

    private static RemoteWeatherAPI apiService = new IpmaApiClient( BASE_URL);
    private MyDatabase localDb;

    public void refreshForecastIfNeeded( int placeId,  Date day) {
        boolean goodLocalData = localDb.hastUpdatedForecast(placeId, day, REFRESH_PERIOD);
        if (! goodLocalData) {
            WeatherForecast forecast = null;
            try {
                forecast = apiService.getForecastForPlace( placeId);
                if (forecast != null) {
                    localDb.save(forecast);
                }
            } catch (IOException e) { e.printStackTrace(); }
        }
    }
}
```

a)


WeatherRepository
-BASE_URL : String = "https://api.ipma.pt/open-data/"
-REFRESH_PERIOD : int = 300
+refreshForecastIfNedeed(placeId : int, day : Date) : void
<<use>>
-apiService
<<Interface>>
RemoteWeatherAPI
+getForecastForPlace(localID : int) : WeatherForecast
<<use>>
-localDb
MyDatabase
+hastUpdatedForecast(placeId : int, day : Date, refreashPeriod : int) : boolean
+save(forecast : WeatherForecast) : void
IpmaApiClient
+IpmaApiClient(url : String)
+getForecastForPlace(localID : int) : WeatherForecast
WeatherForecast

b)


: WeatherRepository
localDb : MyDatabase
apiService : RemoteWeatherAPI
e : java.IO.IOException
1 : refreshForecastIfNeeded(placeId : int, day : Date) : void
1.1 : hastUpdatedForecast(placeId : int = placeId, day : Date = day, refreshPeriod : int = REFRESH_PERIOD) : boolean
opt
[!goodLocalData]
alt
[try]
1.2 : getForecastForPlace(localID : int = placeId) : WeatherForecast
opt
[forecast != null]
1.3 : save(forecast : WeatherForecast = forecast) : void
[catch IOException]
1.4 : printStackTrace()
1.5 :