**Análise e Modelação de Sistemas**

**A) O que é que está incluído no SDLC?**

**O trabalho do Analista na equipa de desenvolvimento**

**● Explicar o que é o ciclo de vida de desenvolvimento de sistemas (SDLC)**

**SDLC (System Development Life Cycle**) é processo que visa compreender de que forma um Sistema de Informação (SI) pode suportar requisitos de negócio, através do design, construção e entrega do sistema aos seus utilizadores. O conceito de SDLC sustenta muitos tipos de metodologias de desenvolvimento de software.

«««Sistema de Informação –> Conjunto de recursos interrelacionados para satisfazer as necessidades de informação de uma organização e dos seus processos de negócio.»»»

**● Descrever as principais atividades/assuntos dentro de cada uma das quatro fases do SDLC**

O SDLC tem quatro fases fundamentais. Estas fases podem ser abordadas de forma diferente conforme as necessidades do negócio.

Diferentes projetos podem enfatizar diferentes partes do SDLC, mas todos os projetos têm elementos destas quatro fases.

Cada fase é composta por uma série de passos, baseados em técnicas que produzem relatórios e ficheiros que facilitam a compreensão do projeto.

- **Planeamento:** Definir a transformação digital pretendida

A fase de planeamento é o processo fundamental de compreensão do porquê de um SI dever ser construído e determinar como a equipa do projeto irá construí-lo.

-> Arranque do projeto: determinar o valor de negócio do Sistema para a organização? Os pedidos do sistema e a análise de viabilidade são apresentados a um comité de aprovação

-> Gestão do projeto: Criação de um plano de trabalho, preparar a equipa para controlar e direcionar o projeto através do SDLC

- **Análise:**

A fase de análise responde às perguntas de quem irá utilizar o sistema, o que o sistema deve fazer, e onde e como será utilizado. Durante esta fase, a equipa investiga quaisquer sistemas atuais e identifica oportunidades de melhoria e desenvolve um conceito para o novo sistema.

-> Estudo do domínio e análise dos sistemas existentes

-> Levantamento de requisitos: Trabalho de ligação com os stakeholders para sistematizar as necessidades

-> Conceito para a solução: Proposta de um sistema que resolve as necessidades identificadas

-**Design (Desenho):**

A fase de desenho decide como o sistema será construído, em termos de harware, software, infraestruturas de redes, a interface, formulários e relatórios do utilizador, programas específicos, base de dados e ficheiros que serão necessários.

-> Estratégia de desenvolvimento

-> Conceção da arquitetura do sistema

-> Conceção do modelo de dados

-> Desenho das entidades de software e seleção de frameworks

-**Implementação:**

Na fase de implementação, o sistema é de facto construído, com a escrita do código, integração de sistemas, desenvolvimento das bases de dados, verificação do software (testes)… inclui também a transição para o ambiente de produção

-> Implementação de sistemas: construção e garantia de qualidade

-> Instalação e transição: colocar em “produção”

-> Plano de suporte: revisões pós-instalação e gestão de modificações

**Existem autores que incluem uma 5ª fase, a fase de manutenção**

**● Definir o termo “processo de software” (*software process*)**

O SDLC é concretizado usando um processo de software sistemático.

Um processo de software é um guião para as atividades e tarefas que são necessárias para construir software de qualidade

**● Distinguir atividades de análise do domínio (de aplicação) de atividades de especificação do software.**

A análise do domínio tem como propósito especificar ou documentar o domínio de aplicação/negócio, e não implica ou assume uma implementação em software

Elementos do modelo representam entidades do negócio

A análise e o desenho de sistemas de software é a análise da estrutura e comportamento de sistemas baseados em software, especialmente adequada para o desenvolvimento por objetos.

Os elementos do modelo representam entidades do mundo do software

**● Descrever o papel e as responsabilidades do Analista no SDLC**

O analista do sistema é quem analisa a situação do negócio, identifica oportunidades de evolução e desenha o sistema de informação que implementa essas oportunidades.

O seu principal objetivo é criar valor para a organização

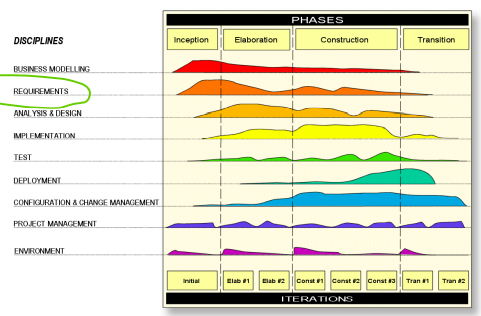
**● Distinguir as competências de “análise de sistemas” das de “programação de sistemas”, em engenharia de software. Relacionar com os conceitos de “soft skills” e “hard skills”.**

**Processo de software e o Unified Process/OpenUP**

●**Descrever a estrutura do UP/OpenUP (fases e objetivos; iterações)**

O UP oferece uma abordagem ao SDLC concebida como uma matriz, cruzando diferentes disciplinas técnicas com iterações no projeto.

A análise de requisitos é realizada principalmente no início do projeto (requisitos básicos), mas também durante as iterações (requisitos evolutivos)



●**Descrever os objetivos e principais atividades de cada fase do UP/OpenUP**

* **Inception (conceção)**

Há uma concordância na visão do projeto e seus objetivos? Deve o projeto avançar?

- Apenas uma pequena iteração

- Entregável: documento de Visão e caso de negócio

-Desenvolvimento de requisitos de mais alto nível (O que é preciso de um modo mais geral)

-Redução de risco: identificação dos requisitos chave

-Perceber que os requisitos vão inevitavelmente mudar

-Produção de protótipos conceptuais sempre que necessário

Milestone: A este ponto, é examinado o custo e os benefícios do projeto e decidido se é para continuar ou cancelar.

* **Elaboration (elaboração)**

Há uma concordância na arquitetura a usar para desenvolver o SI? O valor produzido até então e o risco que resta é aceitável?

- Várias iterações

-Atenuar o risco através da produção de valor (código testado) e fornecendo uma base estável para o grande esforço de desenvolvimento da próxima fase.

-Produção e validação de uma arquitetura executável

-Implementação de alguns componentes chave

-Identificar dependências com sistemas externos e proceder à integração

-Algum código implementado (~10%)

-Arquitetura conduzida pelos Use Cases

Milestone: Validação da arquitetura

* **Construction (construção)**

O sistema está perto o suficiente da entrega? A equipa já está na fase de passar a uma finalização que assegura a entrega bem-sucedida do sistema?

-Várias iterações.

-Construir, desenhar, implementar e testar em todos os cenários possíveis

-Incremento a incremento

-Guiado pela arquitetura.

-Demonstrações frequentes do avanço do projeto e implementação parcial

-Construção diária através de um processo de construção automatizado.

Milestone: Neste ponto, o SI está pronto para ser entregue à equipa de transição. Todas as funcionalidades foram desenvolvidas e todos os testes alfa foram concluídos. Além do software, um manual de utilizador for desenvolvido, e há uma descrição do atual lançamento

* **Transition (transição)**

O SI está pronto para a entrega?

-Estabilização e entrega

-Bug-fixes releases

-Documentação produzida e organizada

Milestone: Aprovação do cliente após rever e aceitar os entregáveis do projeto

●**O OpenUP pode ser considerado “método ágil”?**

**<<** Desenvolvimento Iterativo e Evolutivo **>>**

Baseia-se numa atitude de abraçar a mudança e a adaptação como fatores inevitáveis e, na verdade, essenciais.

Apesar de reconhecermos todos os valores, valorizamos:

- Indivíduos e interações mais que processos e ferramentas

- Software funcional mais do que documentação abrangente

- Colaboração com o cliente mais que negociação contratual

- Responder à mudança mais do que seguir um plano

O OpenUP é considerado um método ágil que promove as melhores práticas de desenvolvimento de software:

* Eficiente, rápido e adaptável, responde à mudança.
* Comunicação efetiva entre stakeholders.
* Trazer o cliente para a equipa.
* É guiado pela discrição dos requisitos do cliente.
* Reconhecer que os planos duram pouco tempo.
* Software desenvolvido iterativamente com ênfase nas atividades de construção

●**Porque é que o UP se assume como “orientado por casos de utilização, focado na arquitetura, iterativo e incremental”?**

Focado na arquitetura → O OpenUP é foca-se na arquitetura no sentido de minimizar o

risco e organizar o desenvolvimento.

Iterativo e incremental → O OpenUP promove práticas que permitem à equipa ter contínuo feedback dos stakeholders, assim como demonstrar-lhes o valor de cada incremento, com a finalidade de lhes fornecer o máximo valor.

Orientado por casos de utilização → A equipa utiliza casos de utilização para orientar todo o trabalho de desenvolvimento, desde a Inception até à Construção.

●**Identificar características distintivas dos processos sequenciais, como a abordagem waterfall.**

Sequencial e linear

→ Avança-se para a fase seguinte só quando a atual estiver concluída.

→ A partir do momento que uma fase é concluída, não se volta atrás.

→ Não há flexibilidade

→ Não há espaço para alterações ou erros, é tudo feito conforme planeado inicialmente.

**●Identificar as práticas distintivas dos métodos ágeis (o que há de novo no modelo de processo, comparando com a abordagem “tradicional”?**).

Os métodos ágeis (OpenUP, SCRUM) surgiram como uma "solução" para as desvantagens da metodologia waterfall.

→ Em vez de um processo de design sequencial, a metodologia Agile segue uma

abordagem incremental e iterativa.

→ O desenvolvimento iterativo foca a entrega de valor orientada por ciclos curtos.

→ Cada iteração produz algum resultado executável, ao contrário da metodologia em

cascata, onde só no fim do processo começam a ser produzidos resultados.

→ Os desenvolvedores começam com um projeto simples a trabalhar em pequenos

módulos. O trabalho nesses módulos é feito em sprints semanais ou mensais e, no final de cada sprint, as prioridades do projeto são avaliadas e são executados testes.

→ Ciclos curtos e entrega de valor frequente, integração em contínuo, desenvolvimento

orientado por testes (TDD).

→ O objetivo dos métodos ágeis é dar resposta rápida à alteração de condições.

→ Entregas frequentes, integração em contínuo → redução de risco

●**Distinguir projetos (de desenvolvimento de software) sequenciais de projetos evolutivos.**

**● Discuta o argumento que “A abordagem em cascata tende a mascarar os riscos reais**

**de um projeto até que seja tarde demais para fazer algo significativo sobre eles.”**

A mudança é inevitável ao longo do desenvolvimento de software. À medida que se avança

num projeto seguindo o método waterfall, a contínua fixação às condições iniciais tende a

mascarar os riscos, até que se chega a uma fase onde a equipa reconhece que não foram

corrigidos erros em fases anteriores, não sendo possível voltar atrás para corrigi-los.

**● Dado um “princípio” (do Agile Manifest), explique-o por palavras próprias, concentrando-se na sua novidade (com relação às abordagens “clássicas”) e impacto/benefício.**

**1. Indivíduos e interações mais do que processos e ferramentas**

São as pessoas que respondem às necessidades de negócios e conduzem o processo de

desenvolvimento. Se o processo ou as ferramentas impulsionam o desenvolvimento, a

equipa responde menos às mudanças e tem menor probabilidade de atender às

necessidades dos clientes.

**Modelação visual e a UML**

●**Justifique o uso de modelos na engenharia de sistemas**

Na engenharia de sistemas é usado modelos de modo a gerir a complexidade.

Segundo G. Booch existem 4 razões para usar modelos:

- Ajudar a visualizar um sistema

-Especificar/documentar a estrutura e o comportamento do sistema

-Serve como referência para orientar a construção

-Documentar as decisões que foram feitas

**●Descreva a diferença entre modelos funcionais, modelos estáticos e modelos de comportamento.**

Modelos funcionais: Descreve o que o sistema deve fazer, do ponto de vista do observador externo. (caixa fechada)

Modelos estáticos: Descreve quais são as partes que constituem o relacionamento estrutural entre entidades. Representam as partes estáticas do sistema, tais como classes, objetos, interfaces e as relações entre todos.

Modelos de comportamento: Descrevem os aspetos dinâmicos de um SI, como é que as partes colaboram/interagem ao longo do tempo.

Durante a análise os modelos comportamentais descrevem qual a lógica interna dos processos sem especificar como os processos vão ser implementados.

Durante o Desenho/Implementação um dos principais objetivos é mostrar como os objetos de um domínio trabalham em conjunto para formar uma colaboração que realiza cada um dos cenários dos casos de utilização

**●Enumerar as vantagens dos modelos visuais.**

Os benefícios dos modelos visuais são:

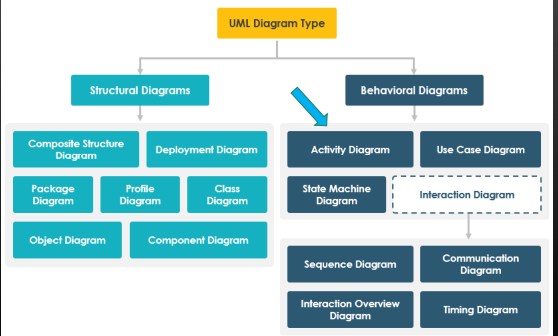
-Promover a comunicação mais clara e sucinta

-Manter o desenho (planeamento) e a implementação (construção) coerentes

- Mostrar ou esconder diferentes níveis de detalhe, conforme apropriado

-Pode suportar, em parte, processos de construção automática (gerar a solução a partir do modelo)

**●Explicar a organização da UML (classificação dos diagramas)**



A UML é uma linguagem própria para modelação que possui uma interface gráfica para a sua interpretação, através de um conjunto muito estrito de figuras representativas de várias atividades, ações, sequências, desenvolvimentos, lançamentos, … entre outros.

**●Caraterizar o “ponto de vista” (perspetiva) de modelação de cada diagrama da UML usado nas aulas Práticas.**

Diagramas de atividades: Servem para documentar a lógica de uma operação ou método simples, componente do fluxo de um processo de negócio.

Estes são aplicados para:

- Modelar fluxos de trabalho/processos de negócios

-Descrever um algoritmo complexo

-Descrever a sequência de interações entre atores e o sistema sob especificação, num caso de utilização

Diagramas de uso de casos: Forma principal de identificar os requisitos do sistema para um novo programa de software em desenvolvimento. Casos de uso especificam o comportamento esperado (o que), e não o método exato de fazer isso acontecer (como).

**●Relacionar os diagramas UML com o momento em que são aplicados, ao longo do projeto de desenvolvimento.**

Os diagramas de atividades são usados para explicar os procedimentos do domínio (contexto de aplicação de um projeto).

As classes são usadas para representar os conceitos da área do problema (modelo do domínio)

**●Identificar os elementos comuns (dos diagramas) da UML e exemplificar a sua utilização.**

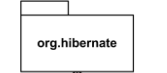
Os elementos comuns dos diagramas são:

- Anotações: Um comentário pode ser usado para anotar qualquer elemento.

Uma imagem com texto

Descrição gerada automaticamente

-Pacotes: Um mecanismo para dividir um modelo em partes. Serve como mecanismo genérico para fazer agrupamentos.



-Estereotipo: Uma especialização da semântica de um elemento da modelação. Marcada com «…» ou com a alteração da decoração

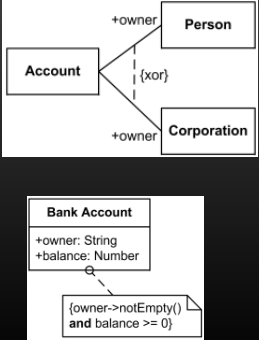


-Valores etiquetados: Entender elementos do modelo com uma linguagem “computável”

Uma imagem com texto

Descrição gerada automaticamente

-Restrições: Adicionar regras ao modelo ou condicionar a sua interpretação. Condição ou restrição relacionada com um ou mais elementos. Linguagem própria para declarar restrições



**●Interpretar e criar Diagramas de Atividades, Diagramas de Casos de Utilização, Diagramas de Classes, Diagramas de Sequência, Diagramas de Estado, Diagramas de Implementação1, Diagramas de Pacotes 1 e Diagramas de Componentes 1.**

Diagrama de Classes → é um tipo de diagrama de estrutura estática que descreve a estrutura de um sistema, mostrando as classes do sistema, seus atributos, operações (ou métodos) e as relações entre objetos.

Diagrama de Componentes → Os diagramas de componentes UML são usados na modelagem dos aspectos físicos de sistemas orientados a objetos que são usados para visualizar, especificar e documentar sistemas baseados em componentes e também para construir sistemas executáveis por meio de engenharia direta e reversa. Os diagramas de componentes são essencialmente diagramas de classes que focam os componentes de um sistema que geralmente são usados para modelar a visualização da implementação estática de um sistema.

Diagrama de implementação→ Um diagrama de implementação da UML é um diagrama que mostra a configuração dos nós de processamento de tempo de execução e os componentes que residem neles. Os diagramas de implementação são um tipo de diagrama de estrutura usado na modelagem dos aspectos físicos de um sistema orientado a objetos. Eles costumam ser usados para modelar a visualização de implantação estática de um sistema (topologia do hardware).

Diagrama de pacotes → O diagrama de pacotes, um tipo de diagrama estrutural, mostra o arranjo e a organização dos elementos do modelo em projetos de média e grande escala. O diagrama de pacotes pode mostrar a estrutura e as dependências entre subsistemas ou módulos, mostrando diferentes visões de um sistema, por exemplo, como um aplicativo de várias camadas (também conhecido como multicamadas) - modelo de aplicativo com várias camadas.

Diagrama de atividades → O diagrama de atividades é outro diagrama comportamental importante na UML para descrever aspectos dinâmicos do sistema. O diagrama de

atividades é essencialmente uma versão avançada do fluxograma que modela o fluxo de

uma atividade para outra.

Diagrama de máquina de estados → Diagramas de Máquina de Estado UML mostram os diferentes

estados de uma entidade. Os diagramas de máquina de estado também podem mostrar

como uma entidade responde a vários eventos mudando de um estado para outro. O

diagrama de máquina de estado é um diagrama UML usado para modelar a natureza

dinâmica de um sistema.

Diagrama de uso de casos → Um diagrama de caso de uso da UML é a forma principal de identificar requisitos de sistema / software para um novo programa de software em

desenvolvimento. Casos de uso especificam o comportamento esperado (o que), e não o método exato de fazer isso acontecer (como). Os casos de uso, uma vez especificados, podem ser designados como representação textual e visual (como UML). Um

conceito-chave da modelagem de casos de uso é que ela nos ajuda a projetar um sistema a partir da perspectiva do utilizador final. É uma técnica eficaz para comunicar o

comportamento do sistema nos termos do usuário, especificando todo o comportamento do sistema visível externamente.

Diagrama de sequência: Ilustrar os objetos que participam numa colaboração e as mensagens que passam entre eles ao longo do tempo.

**B) Compreender as necessidades do negócio (atividades e resultados da Análise)**

**Práticas de engenharia de requisitos**

**●Distinguir entre requisitos funcionais e não funcionais**

Um requisito é uma declaração do que o sistema deve fazer ou uma característica que deve ter. Evoluirá mais tarde para uma descrição técnica de como o sistema será implementado.

Requisitos funcionais: Refere-se a comportamento -> um processo computacional ou tratamento de dados.

Captam o comportamento pretendido do sistema. Serviços, funções ou tarefas que o sistema deve realizar. Podem ser encontrados em descrições de cenários de interação. Pode ser detalhado com diagramas de comportamento, atividades, sequência etc.

Requisitos não funcionais: Diz respeito a uma qualidade ou propriedade do sistema.

Restrições globais/qualidades operacionais num sistema de software. Normalmente, não se limitam a uma função/módulo, mas sim a uma característica transversal.

**●Apresentar técnicas de recolha de requisitos e recomendá-las para diferentes tipos de projeto.**

As técnicas de Recolha de Requisitos é um processo utilizado para descobrir todos os requisitos e obter apoio e confiança entre os utilizadores.

Pode ser usada uma combinação de técnicas

As técnicas a utilizar são:

- Entrevistas

-Workshops de Desenho Colaborativo

-Questionários, Análise de documentos

-Observação

-Focus Group

Como técnicas alternativas temos:

-Análise da área (domínio): Estudar as características do domínio; Aprender com outros;Aumentar a propensão para escalar mais clientes

-Mapas de conceitos: Representam relações importantes entre conceitos; Concentrar as pessoas num pequeno número de ideias-chave

-User Stories: Associado a métodos de desenvolvimento ágeis; Pouca tecnologia utilizada, facilmente atualizável

A importância relativa dos diferentes atributos de qualidade depende do projeto:

Loja on-line → muitas sessões em simultâneo, transações seguras, …

App Dialer (telefonar) → uso intuitivo, comunicação clara do estado da chamada

Homebanking → segurança, disponibilidade, …

**●Distinguir entre abordagens centradas em cenários (utilização) e abordagens centradas no produto para a determinação de requisitos**

Cenários (utilização): Uma perspetiva mais abrangente daquilo que é pretendido por quem vai utilizar o sistema e aquilo que pretende dele.

Produto: Uma perspetiva mais próxima dos stakeholders para perceber o que os próprios querem do produto final, criando também uma relação de confiança entre os developers e o cliente.

**●Identificar, numa lista, requisitos funcionais e atributos de qualidade.**

**●Justifique que “a determinação de requisitos é mais que a recolha de requisitos”.**

O coração do desenvolvimento de requisitos é a elicitação, o processo de identificação das necessidades e restrições das várias partes interessadas para um sistema de software. Elicitação não é o mesmo que “Reunir requisitos”. Também não é uma simples questão de transcrever exatamente o que os utilizadores dizem. Elicitação é um processo colaborativo e analítico que inclui atividades para colecionar, descobrir, extrair e definir requisitos. A elicitação é usada para descobrir negócios, utilizadores, requisitos funcionais e não funcionais, juntamente com outros tipos de informação. A elicitação de requisitos é talvez o aspeto mais desafiador, crítico, propenso a erros e intensivo em comunicação do desenvolvimento de software.

**●Identifique requisitos bem e mal formulados (aplicando os critérios S.M.A.R.T.)**

Requisitos de qualidade simplistas, como "O sistema deve ser fácil de usar" ou "O sistema deve estar disponível 24x7" não são úteis. O primeiro é muito subjetivo e vago; o último raramente é realista ou necessário. Nem é mensurável. Tais requisitos fornecem pouca orientação aos desenvolvedores. Assim, o passo final é criar requisitos específicos e verificáveis a partir das informações que foram obtidas em relação a cada atributo de qualidade. Ao escrever requisitos de qualidade, tenha em mente o útil mnemônico SMART: Idealmente falando, cada objetivo corporativo, departamento e seção deve ser:

****

**●Identifique requisitos bem e mal formulados (aplicando os critérios do ISO-IEEE29148)**

Uma formulação de uma característica que:

-tem de ser satisfeita ou detida por um sistema para resolver um problema ou para atingir um objetivo de alguma ‘parte interessada’.

-pode ser verificada

-define o comportamento ou a capacidade do sistema quando usado, mas não uma capacidade do utilizador ou operador.

-se expressa em linguagem natural, a declaração deve incluir um sujeito, um verbo e um complemento.

**●Discutir as “verdades incontornáveis” apresentadas por Wiegers, sobre os requisitos de sistemas software [original, cópia disponível no material das TP].**

**●Identificar/exemplificar regras de negócio (distinguindo-as do conceito de requisitos).**

As regras de negócio é uma política, orientação, norma ou regulamento que define ou restringe algum dos aspetos do negócio.

Regra vs Requisitos

Não é um requisito de software em si (porque têm uma existência para além dos limites de qualquer aplicação de software específica), mas é a origem de vários tipos de requisitos de software.

Exemplos:

• "Um novo cliente deve pagar 30% da taxa de consultoria estimada e das despesas de viagem, em adiantamento".”

• “Os clientes devem ser maiores de idade (>=18anos).”

• “Há uma joia de admissão, a pagar na inscrição, no valor de 50% da mensalidade.”

Em suma, estas características não dependem do software A ou B, mas terão de ser tidas em consideração na implementação do software.

**●Qual a abordagem proposta no OpenUp para a documentação de requisitos de um produto de software (outcomes relacionados)?**

Para a documentação de requisitos é usado o *SRS report (software requirements specification*),

**●Comentar a afirmação “o processo de determinação de requisitos (requirements elicitation) é primeiramente um desafio de interação humana”.**

**Modelação funcional com casos de utilização**

**●Descrever o processo usado para identificar casos de utilização.**

1. Identificar a fronteira do sistema

2. Identificar os atores que, de alguma forma, interagem com o sistema

3. Para cada autor, identificar os objetivos/motivações para usar o sistema

4. Definir os CaU que satisfazem os objetivos dos autores

**●Ler e criar diagramas de casos de utilização.**

**●Rever modelos de casos de utilização existentes para detetar problemas semânticos e sintáticos.**

**●Descrever os elementos essenciais de uma especificação de caso de uso.**

→ Um identificador único e um nome sucinto que indica o objetivo do utilizador;

→ Uma breve descrição textual que descreve o propósito do caso de uso;

→ Uma condição de disparo que inicia a execução do caso de uso;

→ Zero ou mais pré-condições que devem ser satisfeitas antes que o caso de uso possa começar;

→ Uma ou mais pós-condições que descrevem o estado do sistema após o caso de uso ser concluído com êxito;

→ Uma lista numerada de etapas que mostra a sequência de interações entre o ator e o sistema - um diálogo - que leva das condições prévias às pós-condições.

**●Explicar o uso complementar de diagramas de casos de utilização, diagramas de atividades e narrativas de casos de utilização.**

Desenvolver casos de utilização a partir dos requisitos: como um sistema de negócios interage com o seu ambiente. Inclui um diagrama e uma narrativa para retratar as atividades discretas que os users fazem

Desenvolver diagramas de atividades a partir de casos de utilização: este modela o negócio de processos ou como um negócio opera. Usado para ilustrar o movimento de dados entre atividades

**●Explicar o sentido da expressão “desenvolvimento orientado por casos de utilização”.**

O desenvolvimento orientado por casos de utilização (Use Case Driven Development) é a prática que descreve como capturar requisitos com uma combinação de casos de uso e requisitos de todo o sistema e, em seguida, orientar o desenvolvimento e os testes a partir desses casos de uso.

**●Explicar os seis “Princípios para a adoção de casos de utilização” propostos por Ivar Jacobson(com relação ao “Use Cases 2.0”)**

Existem seis princípios básicos no coração de qualquer sucesso de aplicação de casos de uso:

1. Manter a simplicidade contando histórias (stories).

2. Entender a grande figura (big picture).

3. Concentre-se no valor.

4. Construir o sistema em fatias (slices).

5. Entregar o sistema em incrementos.

6. Adaptar-se para atender às necessidades da equipa.

**●Explicar a relação entre requisitos e os casos de utilização**

Os CaU contam histórias que mostram os requisitos funcionais em contexto, num formato que legível e compreensível para o utilizador final.

**●Identificar as disciplinas e atividades relacionadas aos requisitos no OpenUP**

**●Relacionar o caso de utilização (entidade de modelação) com os cenários (formas de percorrer o caso de uso).**

**●objetos.**

**●Identifique o uso adequado de classes de associação.**

Associação é a relação semântica que se estabelece entre duas ou mais classes que descreve as ligações existentes entre as respetivas instâncias

A multiplicidade de uma associação é o número de instâncias de uma classe que se relacionam com uma instância de outra.

Uma agregação é uma forma especial de associação que modela uma relação de todo-parte, entre o agregador e suas partes constituintes

Navegabilidade indica a possibilidade de navegar de uma classe de partida para uma classe de chegada, usando a associação