

Tópicos de estudo para o exame

Utilizar este documento de apoio como uma revisão dos assuntos a serem estudados. Resulta essencialmente de compilar os objetivos de aprendizagem que já constam nas apresentações das aulas TP.

Revisto em: 2024-06-14

Conteúdos:

Visão geral dos conteúdos da disciplina	2
A) O que é que está incluído no SDLC?	2
O SDLC e o trabalho do Analista	2
Processo de software e o Unified Process/OpenUP	3
Modelação visual e a UML	3
B) Compreender as necessidades do negócio (atividades e resultados da Análise)	3
Práticas de engenharia de requisitos	3
Modelação funcional com casos de utilização	4
A modelação do contexto do problema: modelo do domínio/negócio	4
C) Modelos no desenvolvimento	4
Orientação aos objetos no SDLC.....	4
Modelação estrutural	4
Modelos de comportamento.....	5
Vistas de arquitetura	5
Desenho do software (perspetiva do programador)	5
D) Práticas selecionadas na construção do software	5
Garantia de qualidade	5
Integração contínua/Entrega Contínua.....	6
E) Práticas dos métodos ágeis.....	6
Principais características dos métodos ágeis de desenvolvimento	6
Histórias (=user stories) e métodos ágeis	6
O framework SCRUM	7

Visão geral dos conteúdos da disciplina



A mensagem central de AS: perante o papel cada vez mais decisivo dos sistemas de software no processo de transformação digital das economias e da sociedade, coloca-se uma crescente exigência no processo de desenvolvimento (o *software process*). Um dos pontos críticos é a correta determinação dos requisitos: não pode haver um produto de sucesso perante requisitos mal definidos. Há várias maneiras de abordar a definição de requisitos, com claras vantagens para as abordagens centradas na utilização, através de análise de cenários (de interação). Com uma visão clara das motivações dos utilizadores e *stakeholders*, a construção do software deve ser evolutiva, ao longo de vários ciclos (incrementos), em que se constrói e entrega pacotes de funcionalidade relevantes para o promotor/cliente. No desenvolvimento incremental, podemos procurar um equilíbrio entre as técnicas de modelação mais abrangentes (e.g.: use cases) e as técnicas de especificação mais leves (e.g.: *user stories*).

Os modelos (e.g.: construídos na UML) são uma ferramenta para facilitar a comunicação e a colaboração na equipa, usados de forma transversal no *software process*.

A) O que é que está incluído no SDLC?

O SDLC e o trabalho do Analista

- Explicar o que é o ciclo de vida de desenvolvimento de sistemas (SDLC)
- Descrever as principais atividades/assuntos dentro de cada uma das fases do SDLC (há autores que incluem 4 fases no SDLC, outros que incluem 5 fases com a Manutenção).
- Definir o termo “processo de software” (*software process*)
- Distinguir atividades de análise do domínio (de aplicação) de atividades de especificação do (produto de) software.
- Descrever o papel e as responsabilidades do Analista no SDLC
- 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” com o papel do analista.

Processo de software e o Unified Process/OpenUP

- Descrever a estrutura do UP/OpenUP (fases e objetivos; iterações)
- Descrever os objetivos e principais atividades de cada fase do UP/OpenUP
- O OpenUP pode ser considerado “método ágil”?
- Porque é que o UP se assume como “orientado por casos de utilização, focado na arquitetura, iterativo e incremental”?
- Identificar características distintivas dos processos sequenciais, como a abordagem *waterfall*.
- Identificar as práticas distintivas dos métodos ágeis (o que há de novo no modelo de processo, comparando com a abordagem “tradicional”?).
- Distinguir projetos (de desenvolvimento de software) sequenciais de projetos evolutivos.

Modelação visual e a UML

- Justifique o uso de modelos na engenharia de sistemas
- Descreva a diferença entre modelos funcionais, modelos estáticos/estruturais e modelos de comportamento.
- Enumerar as vantagens dos modelos visuais.
- Explicar a organização da UML (classificação dos diagramas)
- Caracterizar o “ponto de vista” (perspetiva) de modelação de cada diagrama da UML usado nas aulas Práticas.
- Relacionar os diagramas UML com o momento em que são aplicados, ao longo do projeto de desenvolvimento.
- Identificar os elementos comuns (dos diagramas) da UML e exemplificar a sua utilização.
- Interpretar e criar Diagramas de Atividades, Diagramas de Casos de Utilização, Diagramas de Classes, Diagramas de Sequência, Diagramas de Estado, Diagramas de Implementação, Diagramas de Pacotes e Diagramas de Componentes.
Este tópico engloba, naturalmente, o domínio da semântica dos elementos de modelação e suas relações, nos diagramas referidos.

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

Práticas de engenharia de requisitos

- Distinguir entre requisitos funcionais e não funcionais
- Apresentar técnicas de recolha de requisitos e recomendá-las para diferentes tipos de projeto.
- Distinguir entre abordagens centradas em cenários (utilização) e abordagens centradas no produto para a determinação de requisitos
- Identificar, numa lista, requisitos funcionais e atributos de qualidade.
- Justifique que “a determinação de requisitos é mais que a recolha de requisitos”.
- Identifique requisitos bem e mal formulados (aplicando os critérios S.M.A.R.T.)
- Identifique requisitos bem e mal formulados (aplicando os critérios do ISO-IEEE 29148)
- 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).
- Qual a abordagem proposta no OpenUp para a documentação de requisitos de um produto de software (*outcomes* relacionados)?
- 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.
- 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.
- Explicar o uso complementar de diagramas de casos de utilização, diagramas de atividades e narrativas de casos de utilização.
- Explicar o sentido da expressão “desenvolvimento orientado por casos de utilização”.
- Explicar os seis “Princípios para a adoção de casos de utilização” propostos por Ivar Jacobson (com relação ao “Use Cases 2.0”)
- Explicar a relação entre requisitos e os casos de utilização
- 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).
- Identifique o uso adequado de classes de associação.
- Como é que o empacotamento dos casos de uso (resultado da análise), pode contribuir para a identificação de potenciais módulos, na arquitetura?

A modelação do contexto do problema: modelo do domínio/negócio

Caraterizar os conceitos do domínio de aplicação:

- Desenhe um diagrama de classes simples para capturar os conceitos de um domínio de problema.
- Apresente duas estratégias para descobrir sistematicamente os conceitos candidatos para incluir no modelo de domínio.
- Identificar construções específicas (associadas à implementação) que podem poluir o modelo de domínio (na etapa de análise).

Caraterizar os processos do negócio/organizacionais:

- Leia e desenhe diagramas de atividades para descrever os fluxos de trabalho da organização / negócios.
- Identifique o uso adequado de ações, fluxo de controle, fluxo de objetos, eventos e partições com relação a uma determinada descrição de um processo.
- Relacione os “conceitos da área do negócio” (classes no modelo de domínio) com fluxos de objetos nos modelos de atividade.

C) Modelos no desenvolvimento

Orientação aos objetos no SDLC

- Discutir as diferenças e complementaridades entre os conceitos de orientação aos objetos na Análise (OOA), no Desenho (OOD) e na programação (OOP).
- Apresentar a distribuição de responsabilidades e a colaboração mecanismos de base na orientação aos objetos.

Modelação estrutural

- Distinguir entre a análise de sistemas baseada numa abordagem algorítmica *top-down* e baseada nos conceitos do domínio do problema.
- Explicar a relação entre os diagramas de classe e de objetos.
- Rever um modelo de classes quanto a problemas de sintaxe e semânticos, considerando uma descrição do um problema de aplicação.

- Descreva os tipos e funções das diferentes associações no diagrama de classes. Identifique o uso adequado da associação, composição e agregação para modelar a relação entre

Modelos de comportamento

- Explique o papel da modelação de comportamento no SDLC
- Explicar a complementaridade entre diagramas de sequência e de comunicação
- Relacionar a ideia essencial (na orientação aos objetos) de distribuição de responsabilidades com o diagrama de sequência (como é que ajuda?).
- Relacionar representações nos diagramas de sequência com código por objetos e vice-versa.
- Representar o ciclo de vida de uma entidade num diagrama de estados.
- Relacionar elementos presentes num D. Sequência com as entidades de um D. de Classes.
- Como é que o desenvolvimento de um modelo de colaboração entre objetos pode ser conduzido a partir dos casos de utilização? (*use case realizations*)

Vistas de arquitetura

- Explicar as atividades associadas ao desenvolvimento de arquitetura de software, no openUP.
- Explicar a relação entre requisitos e a arquitetura (como é que aqueles influenciam esta). Exemplificar requisitos com impacto na arquitetura.
- Explique a prática de “arquitetura evolutiva” proposta no OpenUp.
- Identifique as camadas e partições numa arquitetura de software por camadas.
- Usar diagramas de sequência para descrever a cooperação entre módulos/elementos de uma arquitetura.
- Identifique os três tipos de estruturas principais, constituintes de uma arquitetura (segundo L. Bass *et al*). Relacionar essas categorias com os diagramas de UML mais relevantes para as documentar.
- Discutir razões técnicas e não técnicas que justificam o desenvolvimento de uma (boa) arquitetura para o sistema a construir.

Desenho do software (perspetiva do programador)

- Explicar como os casos de utilização podem ser usados para orientar as atividades de desenho (na perspetiva do livro do Larman).
- Explicar os princípios do baixo acoplamento e alta coesão em OO. Discutir implicações do *coupling* e *cohesion*.
- Comparar, num dado desenho por objetos, a ocorrência de maior/menor *coupling* e *cohesion*.
- Relacionar código por objetos com a sua representação em diagramas de classes da UML.
- Modelar a interação entre unidades de software (objetos) como diagramas de sequência.
- Construa um diagrama de classes e um diagrama de sequência considerando um código Java.
- Explicar as implicações no código da naveabilidade modelada no diagrama de classes.

D) Práticas selecionadas na construção do software

Garantia de qualidade

- Identifique as atividades de validação e verificação incluídas no SDLC
- Descreva quais são as camadas da pirâmide de teste
- Descreva o assunto/objetivo dos testes de unidade, integração, sistema e de aceitação
- Explique o ciclo de vida do TDD → ~~rejeite - se vários erros~~
- Descreva as abordagens “debug-later” e “test-driven”, de acordo com J. Grenning.
- Explique como é que as atividades de garantia de qualidade (QA) são inseridas no processo de

“debug-later”

código → testar → falha → debugar → corrigir



“test-driven”

teste → ~~falha~~ → código → para melhorar ↩

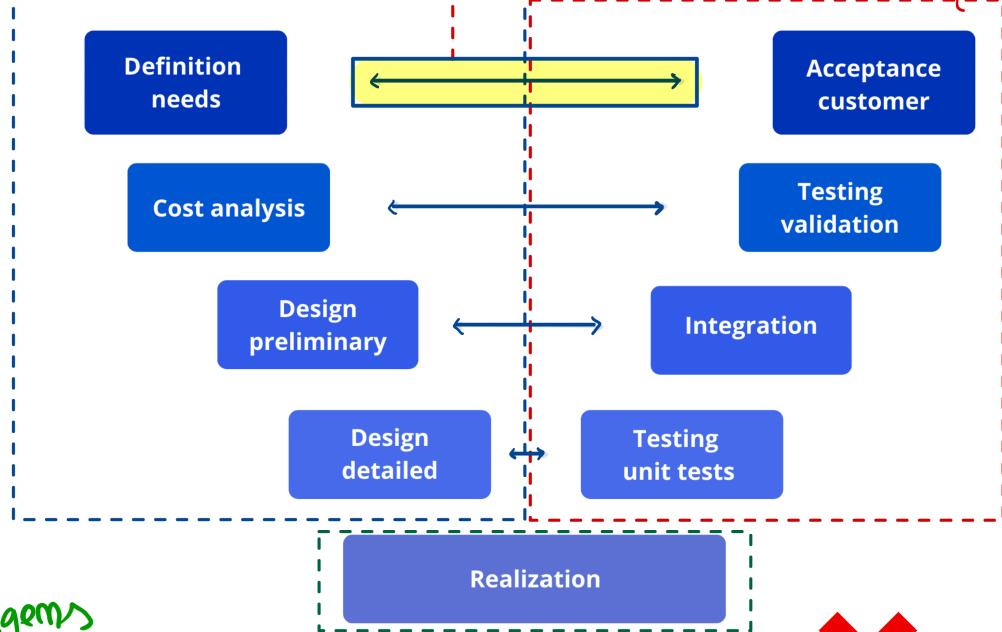
THE V MODEL METHOD



Cada fase de desenvolvimento tem
um teste associado.

Análise

Validação



Vantagens

- Fácil de entender e aplicar
- Ajuda a manter a qualidade do software
- Bom para projetos c/ requisitos bem definidos

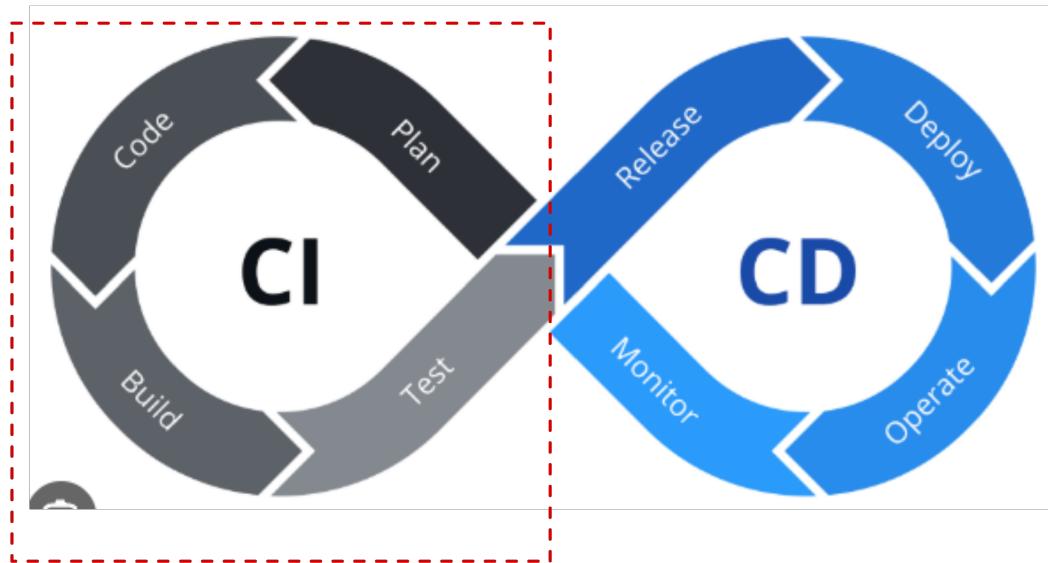
Codificação



Desvantagens

- Pouca flexibilidade mudanças

Conceito	O que é	Nível de Automatização	Entrega em Produção?
CI (Continuous Integration)	Integração contínua de código e execução automática de testes	Alta (integração + testes)	<input checked="" type="checkbox"/> Não
CD (Continuous Delivery)	Prepara automaticamente o sistema para ser lançado	Muito alta (build + testes + empacotamento)	<input checked="" type="checkbox"/> Sim, mas com aprovação manual
CD (Continuous Deployment)	Entrega contínua e automática em produção após testes	Total (inclui deploy automático)	<input checked="" type="checkbox"/> Sim, sem intervenção manual



condições adequadas (o que deve ser feito)

- desenvolvimento, numa abordagem clássica e nos métodos ágeis.
- O que é o “V-model”?
- Relacione os critérios de aceitação da história (*user-story*) com o teste *Agile*. Explique o sentido da afirmação “especificações executáveis”.
- Justifique a necessidade de testes “*developer facing*” e “*customer facing*”.

Integração contínua/Entrega Contínua

*I, conexão técnica
Ingenieros
Testes unitários*

L valida o software fog o que o cliente espera

- Identificar os passos típicos de um ciclo de CI
- Distinguir entre C. Integration, C. Deployment e C. Delivery
- Relacionar o CI/CD com a natureza iterativa e incremental dos métodos ágeis de desenvolvimento, ou seja, como é que o CI/CD ajuda na concretização de metodologias incrementais?
- Explicar as tarefas e *outcomes* englobados numa “*build*”
- Explique o sentido da prática “*continuous testing*”.

E) Práticas dos métodos ágeis

Principais características dos métodos ágeis de desenvolvimento

- Discuta o argumento: “A abordagem em cascata tende a mascarar os riscos reais de um projeto até que seja tarde demais para fazer algo significativo sobre eles.”
- Explique o sentido da frase: “O desenvolvimento iterativo centra-se em ciclos curtos e orientados para a geração de valor”
→ Não elimina, mas é mais flexível
- Discuta o argumento: “A abordagem ágil dispensa o planeamento do projeto”.
- Identifique vantagens de estruturar um projeto em iterações, produzindo incrementos funcionais com frequência.
- Caracterizar os princípios da gestão do *backlog* em projetos ágeis.
*Prioridades (+ imortante nem primeiro)
Evolução Contínua
Gendo pelo Product Owner*
- Dado um “princípio” (do *Agile Manifest*), explicá-lo por palavras próprias, destacando a sua novidade (com relação às abordagens “clássicas”) e impacto / benefício.
- Apresentar situações em que, de facto, um método sequencial pode ser o mais adequado.
- Quais os objetivos das organizações com a adoção de metodologias de desenvolvimento “iterativas e incrementais”?

Histórias (=*user stories*) e métodos ágeis

- Defina histórias (*user stories - US*) e dê exemplos.
- Como é que o conceito de US se relaciona [ou não] com o de requisito (da análise)?
- Compare histórias e casos de utilização em relação a pontos comuns e diferenças. Em que medida podem ser usados de forma complementar?
→ Até ajudam a entender o sistema
- Compare “Personas” com Ator com respeito a semelhanças e diferenças. Qual a utilidade das Personas no desenvolvimento das US?
→ Ator - Papel ou entidade externa
- O que é a pontuação de uma história e como é que é determinada?
- Descreva o conceito de velocidade da equipa (como usado no PivotalTracker e SCRUM).
- Explique a abordagem proposta por Jacobson em “Use Cases 2.0” para combinar a técnica dos *use cases* com a flexibilidade das *user stories*.
- Discuta se os casos de utilização e as histórias são abordagens redundantes ou complementares (quando seguir cada uma das abordagens? Em que condições? ...)
- Exemplifique a definição de critérios de aceitação para uma US.
→ critérios para as use-cases devem conectar
- Como é que um *story map* organiza espacialmente o *backlog*, ou seja, como se usa o *story map* no contexto dos métodos ágeis?

quantidade de trabalho que uma equipa consegue fazer em um sprint

O framework SCRUM de gestão de equipas

- Explique o objetivo da “**Daily Scrum meeting**”
- Relacione os conceitos de *sprint* e iteração e discuta a sua duração esperada.
- Explique a método de pontuação das histórias (e critérios aplicados) – estimar o esforço relativo (menor/médio/maior)
Uma história de utilizador
- Identificar os papéis numa equipa de Scrum e as principais “cerimónias” (- 10mtoz) → + simplicidade
- Relacione as práticas previstas no SCRUM e os princípios do “Agile Manifesto”: em que medida estão alinhados? (equipa, software funcional > documentação, mudança > seguir um plano)
- Pode-se considerar o SCRUM como a “silver bullet” (solução universal) para o sucesso dos projetos de desenvolvimento? **NÃO**, situações muito específicas (equipa organizada, etc...)
- Identifique alguns desafios/bloqueios à implementação eficaz do SCRUM.
 - Gestores presos aos modelos tradicionais
 - Não compreensão do framework
 - Falta do envolvimento do Product Owner
 - Premões por entregas imediatas

Alimentar a equipa sobre o progresso do projeto

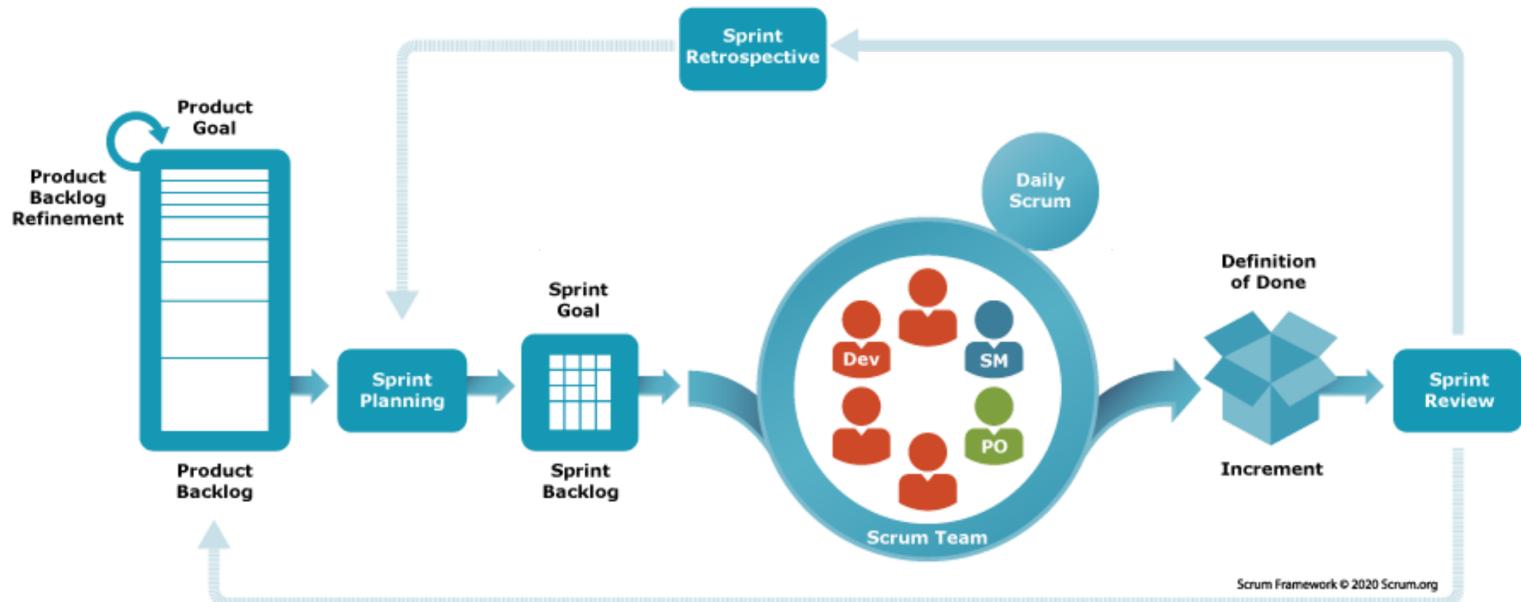
1 a 4 semanas

Sprint é uma iteração do SCRUM

estimar o esforço relativo menor/medio/maior

Uma história de utilizador

(- 10mtoz) → + simplicidade



Resumo dos Conteúdos

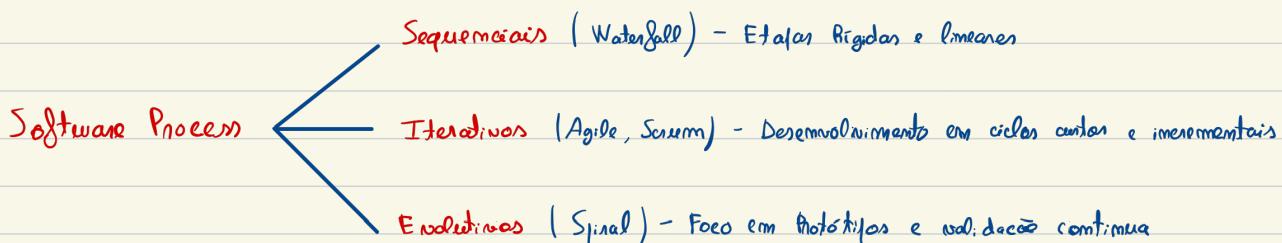
A) O que é que está incluído no SDLC?

O SDLC e o trabalho do Analista

- Explicar o que é o ciclo de vida de desenvolvimento de sistemas (SDLC) ✓
- Descrever as principais atividades/assuntos dentro de cada uma das fases do SDLC (há autores que incluem 4 fases no SDLC, outros que incluem 5 fases com a Manutenção). ✓
- Definir o tema "processo de software" (software process) ✓
- Distinguir atividades de análise do domínio (de aplicação) de atividades de especificação do (produto de) software.
- Descrever o papel e as responsabilidades do Analista no SDLC ✓
- 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" com o papel do analista. ✓



Software Process: Conjunto de atividades que estão interrelacionadas, que produzem um produto de software, envolvendo planejamento, desenvolvimento, teste, entrega e manutenção.



Análise de Domínio VS Especificação do Produto (Software)

Resumo da distinção

Aspecto	Análise de Domínio	Especificação do Produto de Software
Foco	Entendimento do problema e do contexto	Definição do que o software deve fazer
Perspectiva	Visão ampla do domínio	Visão detalhada do sistema a ser desenvolvido
Resultado típico	Modelos conceituais, glossário, processos	Requisitos funcionais e não funcionais
Interlocutores principais	Especialistas do domínio, usuários	Clientes, analistas de requisitos, desenvolvedores
Tempo no processo	Antes da especificação	Após a análise de domínio

Análise de Domínio:

- Entender profundamente o contexto e o ambiente em que o software será usado

- Foco no mundo real, não no sistema ainda

Especificação do Produto de Software:

- Definir exatamente o que o sistema deve fazer (suas funções, restrições e interfaces)

- Foco no sistema que será construído, não no domínio em geral

Papel do Analista no SDLC:

1. Planejamento

- Participar da análise de viabilidade técnica e econômica.
- Contribuir com a definição do escopo.
- Ayudar a identificar stakeholders e suas expectativas.

2. Análise de Requisitos

- Realizar levantamento de requisitos por meio de entrevistas, workshops, observações e questionários.
- Conduzir a análise de domínio, identificando entidades, processos e regras de negócio.
- Redigir a Especificação de Requisitos de Software (ERS).

3. Modelagem e Design

- Ajuda a modelar os casos de uso, fluxos de dados, diagramas de entidade-relacionamento (ER), BPMN etc.
- Esclarecer regras de negócio para os arquitetos e designers de sistema.



4. Desenvolvimento

- Atuar como referência técnica dos requisitos: tirar dúvidas, revisar implementações e ajustes.
- Gerenciar mudanças de requisitos, se necessário.

5. Teste e Validação

- Ayudar a definir critérios de aceitação com base nos requisitos.
- Revisar os casos de teste.
- Participar de testes de aceitação do usuário (UAT).

6. Implantação e Manutenção

- Acompanhar o comportamento do sistema em produção.
- Coletar feedback dos usuários e relatar melhorias ou problemas.
- Apoiar na priorização de correções ou novas funcionalidades.

Análise de Sistemas VS Programação de Sistemas

■ Objetivo Principal

Análise de Sistemas

Entender e documentar o que o sistema deve fazer, considerando o contexto do negócio e as necessidades dos usuários.

Programação de Sistemas

Implementar e otimizar o código-fonte para que o sistema execute as funcionalidades definidas pelo analista.

HARD SKILLS	Análise de Sistemas	Programação de Sistemas
Ferramentas	UML, BPMN, Mina, Trello	IDE's (VSCode, ...) GITHUB...
Técnicas	Enumerações de Requisitos, elaboração de use cases	Estrutura de dados, Linguagens (Python, C, C++), testes unitários
Padrões	Padrões de especificação ex (IEEE 830)	Padrões de Projeto, Primeiros SOLID, Clean Code

SOFT SKILLS	Análise de Sistemas	Programação de Sistemas
Comunicação	Felicitações de Workshops, entrevistas c/ stakeholders	Colaboração em code reviews
Negociação	Gerenciadas estatísticas de várias áreas (negócio, TI, usuários)	Definir prioridades do implementação com a equipe
Pensamento Crítico	Traduzir holofitas de negócios em requisitos técnicos coerentes	Diagnosticar holofitas de performance e bugs complexos.

Processo de software e o Unified Process/OpenUP

- Descrever a estrutura do UP/OpenUP (fases e objetivos; iterações)
- Descrever os objetivos e principais atividades de cada fase do UP/OpenUP
- O OpenUP pode ser considerado “método ágil”?
- Porque é que o UP se assume como “orientado por casos de utilização, focado na arquitetura, iterativo e incremental”?
- Identificar características distintivas dos processos sequenciais, como a abordagem waterfall.
- Identificar as práticas distintivas dos métodos ágeis (o que há de novo no modelo de processo, comparando com a abordagem “tradicional”?).
- Distinguir projetos (de desenvolvimento de software) sequenciais de projetos evolutivos.

Resumo: Processo de Software e OpenUP

- Estrutura do UP/OpenUP:

O Unified Process (UP), e sua versão open-source OpenUP, organiza o desenvolvimento em quatro fases principais: Iniciação, Elaboração, Construção e Transição. Cada fase tem objetivos bem definidos e contém várias iterações. A estrutura permite mitigar riscos desde cedo, promover a reutilização de componentes e adaptar o projeto à medida que o entendimento do problema evolui. O modelo é iterativo, incremental e dirigido por casos de uso, com forte ênfase em arquitetura desde o início.

- Objetivos e atividades principais de cada fase:

Iniciação: estabelecer a visão do sistema, identificar stakeholders e principais riscos.

Elaboração: detalhar requisitos críticos, definir arquitetura de base e planejar o projeto.

Construção: desenvolvimento iterativo da funcionalidade, integração e testes contínuos.

Transição: entrega do produto ao utilizador final, correções finais e avaliação do sistema.

- O OpenUP pode ser considerado ágil?

Sim. O OpenUP incorpora várias práticas dos métodos ágeis como desenvolvimento incremental, foco no envolvimento do utilizador, comunicação contínua, e entregas frequentes de software funcional. No entanto, mantém também a disciplina e estrutura típica de processos mais formais, sendo por isso considerado um processo híbrido.

- Orientação por casos de utilização e foco arquitetural:

O UP promove o uso de casos de utilização como principal técnica para capturar requisitos. Estes guiam o desenvolvimento e ajudam a manter o foco nas funcionalidades relevantes para os utilizadores. A arquitetura do sistema é tratada como prioridade desde as primeiras fases, permitindo decisões técnicas robustas e sustentáveis ao longo do tempo. O modelo iterativo permite ajustar o sistema com base no feedback contínuo.

- Processos sequenciais (ex: waterfall) vs iterativos:

Processos sequenciais como o Waterfall seguem uma ordem fixa (análise -> design -> implementação -> testes). Isso torna-os rígidos e pouco adaptáveis a mudanças. Já os processos iterativos (como o OpenUP)

Resumo: Processo de Software e OpenUP

favorecem a entrega contínua de valor, permitindo revisões e adaptações frequentes, o que reduz riscos e melhora a qualidade final.

- Práticas distintivas dos métodos ágeis:

Métodos ágeis introduzem práticas como trabalho em equipa auto-organizado, reuniões diárias, backlog de produto, iterações curtas com entregas incrementais, testes automatizados e forte envolvimento do cliente.

Comparado ao modelo tradicional, o ágil privilegia adaptabilidade e colaboração contínua em detrimento de planeamentos extensos e documentação exaustiva.



- Projetos sequenciais vs evolutivos:

Projetos sequenciais são definidos por etapas claras e cronológicas, mas assumem que os requisitos não mudam. Projetos evolutivos, como os baseados no OpenUP, reconhecem que mudanças são inevitáveis e estruturam o trabalho para acomodar aprendizagem contínua, entregando valor de forma incremental e adaptativa.

Modelação visual e a UML

- Justifique o uso de modelos na engenharia de sistemas
- Descreva a diferença entre modelos funcionais, modelos estáticos/estruturais e modelos de comportamento.
- Enumerar as vantagens dos modelos visuais.
- Explicar a organização da UML (classificação dos diagramas)
- Caracterizar o “ponto de vista” (perspetiva) de modelação de cada diagrama da UML usado nas aulas Práticas.
- Relacionar os diagramas UML com o momento em que são aplicados, ao longo do projeto de desenvolvimento.

Resumo: Modelação Visual e UML

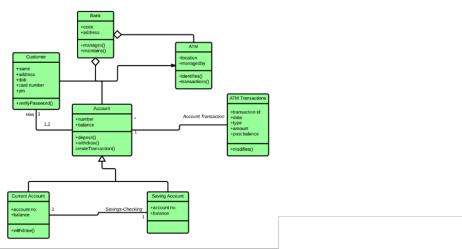
- Justificação do uso de modelos na engenharia de sistemas:

Modelos ajudam a representar visualmente conceitos complexos, facilitando a comunicação entre os membros da equipa. Permitem antecipar problemas, alinhar requisitos e validar soluções antes da implementação.

- Diferença entre modelos funcionais, estruturais e de comportamento:

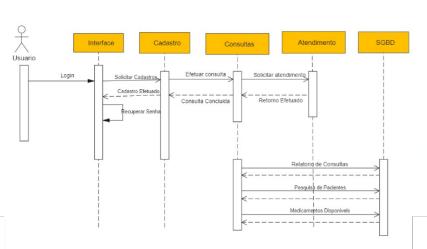
MODELOS ESTRUTURAIS

mostram a estrutura estática do sistema, como classes e relações (ex: diagramas de classes)



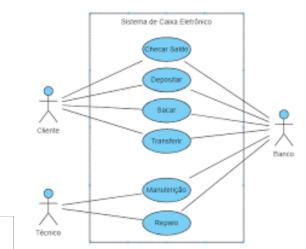
Modelos de comportamento

representam como os elementos interagem e se comportam ao longo do tempo (ex: diagramas de sequência)



MODELOS FUNCIONAIS

focam nas funções ou processos realizados pelo sistema (ex: casos de uso)



- Vantagens dos modelos visuais:

- Melhor compreensão dos requisitos e arquitetura.
- Comunicação eficaz com stakeholders.
- Detecção precoce de inconsistências.
- Base para documentação e manutenção futura.

- Organização da UML:

A UML é composta por diversos diagramas, classificados em três grupos:

- **Estruturais:** descrevem a estrutura estática (ex: classes, componentes).
- **Comportamentais:** descrevem comportamentos do sistema (ex: casos de uso, estados).
- **De interação:** focam na troca de mensagens entre objetos (ex: sequência, comunicação).

Unified Modeling Language

- Perspectiva de modelação de cada diagrama da UML:

- Casos de utilização: visão funcional do sistema.
- Classes: visão estrutural.
- Sequência: interação entre objetos no tempo.
- Atividades: lógica de processos.

UNIFIED
MODELING
LANGUAGE™



Resumo: Modelação Visual e UML

- Estados: ciclos de vida de entidades.
- Componentes/Pacotes: organização física/modular do software.

- Aplicação dos diagramas UML no projeto:

use cases

- Início: diagramas de **casos de utilização** para capturar requisitos.
- Análise: **diagramas de classes** e de atividades.
- Design: **diagramas de sequência**, componentes, pacotes.
- Implementação: componentes e deployment.

Cada tipo de diagrama é útil em momentos específicos, ajudando a garantir coerência ao longo do projeto.

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

Práticas de engenharia de requisitos

- Distinguir entre requisitos funcionais e não funcionais
- Apresentar técnicas de recolha de requisitos e recomendá-las para diferentes tipos de projeto.
- Distinguir entre abordagens centradas em cenários (utilização) e abordagens centradas no produto para a determinação de requisitos
- Identificar, numa lista, requisitos funcionais e atributos de qualidade.
- Justifique que “a determinação de requisitos é mais que a recolha de requisitos”.
- Identifique requisitos bem e mal formulados (aplicando os critérios S.M.A.R.T.)
- Identifique requisitos bem e mal formulados (aplicando os critérios do ISO-IEEE 29148)
- 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).
- Qual a abordagem proposta no OpenUp para a documentação de requisitos de um produto de software (*outcomes* relacionados)?
- Comentar a afirmação “o processo de determinação de requisitos (*requirements elicitation*) é primeiramente um desafio de interação humana”.

Resumo: Engenharia de Requisitos

- Requisitos funcionais vs não funcionais:

o que o sistema faz → *funcionalidades específicas que suportam*

Requisitos funcionais descrevem o que o sistema deve fazer, como funcionalidades específicas que suportam

os objetivos do utilizador (ex: 'O sistema deve permitir ao utilizador alterar a palavra-passe.').

Já os não funcionais definem critérios de qualidade do sistema (ex: desempenho, escalabilidade, usabilidade).

São também chamados de requisitos de qualidade ou atributos do sistema.

- Técnicas de recolha de requisitos:

- **Entrevistas:** úteis quando há acesso direto aos stakeholders.
- **Questionários:** para recolher dados de muitos utilizadores de forma eficiente.
- **Observação direta:** analisar o contexto real de uso.
- **Workshops:** promovem colaboração e consenso.
- **Análise de documentos:** útil quando substitui ou complementa conhecimento tácito.

- Abordagens centradas em cenários vs produto:

- centrada no utilizador*
- **Abordagem centrada em cenários:** explora interações reais e ajuda a descobrir requisitos implícitos.
 - **Abordagem centrada no produto:** começa pelas funcionalidades pretendidas do sistema, independentemente de quem as usa.
 - A abordagem por cenários é mais centrada no utilizador e é típica de métodos ágeis.

- Requisitos funcionais e atributos de qualidade:

- funcionalidades do sistema*
- **Requisitos funcionais** estão diretamente ligados às funcionalidades do sistema.
 - **Atributos de qualidade** incluem fiabilidade, segurança, facilidade de manutenção, desempenho e usabilidade.
 - Ambos são fundamentais e complementares no sucesso do sistema.

Resumo: Engenharia de Requisitos

- Determinação de requisitos vai além da recolha:

- Recolher requisitos é apenas a primeira etapa.
- Análise, validação, negociação, documentação e gestão fazem parte do ciclo completo.
- Exige habilidades técnicas e interpessoais.

Soft Skills

- Requisitos bem e mal formulados (S.M.A.R.T.):



- S.M.A.R.T.: Específico, Mensurável, Atingível, Relevante e Temporal.
- ⚠️ - Requisitos mal formulados geralmente carecem de clareza ou precisão, e são difíceis de testar.
- Ex: 'O sistema deve ser rápido' é vago; 'O sistema deve responder em menos de 2 segundos' é SMART. ✓

- Requisitos bem e mal formulados (ISO-IEEE 29148):

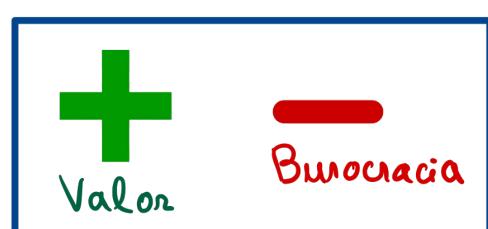
- Padrão internacional que define boas práticas para especificação de requisitos.
- Requisitos devem ser completos, consistentes, verificáveis, rastreáveis e compreensíveis.
- Linguagem ambígua é um erro comum (ex: 'fácil de usar').
↳ Não usar! ⚡

- Regras de negócio vs requisitos:

- Regras de negócio são restrições lógicas da organização (ex: 'clientes devem ter mais de 18 anos').
- Requisitos derivam dessas regras, mas não as substituem. (ex. "Validar a idade no formulário")
- Regras podem ser reutilizadas em vários sistemas ou módulos.

- Abordagem OpenUP para documentação:

- OpenUP propõe artefactos leves, úteis e evolutivos. → Só o necessário!
- Inclui Casos de Utilização, Glossário, Requisitos Suplementares e Visão do Sistema.
- A documentação é feita com foco em valor e não em volume.
↳ Atualização Contínua



Resumo: Engenharia de Requisitos

- Requirements elicitation como desafio humano:

- Envolve comunicação com pessoas de diferentes perfis e interesses.
- Exige empatia, escuta ativa, mediação de conflitos e adaptação da linguagem.
- Requisitos mal definidos geralmente decorrem de má comunicação e falta de envolvimento dos stakeholders.

Não é só recolher dados → é compreender as pessoas

Modelação funcional com casos de utilização

- Descrever o processo usado para identificar casos de utilização.
- 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.
- Explicar o uso complementar de diagramas de casos de utilização, diagramas de atividades e narrativas de casos de utilização.
- Explicar o sentido da expressão “desenvolvimento orientado por casos de utilização”.
- Explicar os seis “Princípios para a adoção de casos de utilização” propostos por Ivar Jacobson (com relação ao “Use Cases 2.0”)
- Explicar a relação entre requisitos e os casos de utilização
- 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).
- Identifique o uso adequado de classes de associação.
- Como é que o empacotamento dos casos de uso (resultado da análise), pode contribuir para a identificação de potenciais módulos, na arquitetura?

Resumo Elaborado: Modelação Funcional com Casos de Utilização

Identificar casos de utilização

Parte da compreensão dos objetivos dos utilizadores e das interações com o sistema.

Atores são entidades externas (utilizadores, sistemas, organizações) que interagem diretamente com o sistema.

Cada objetivo importante de um ator representa um caso de uso candidato.

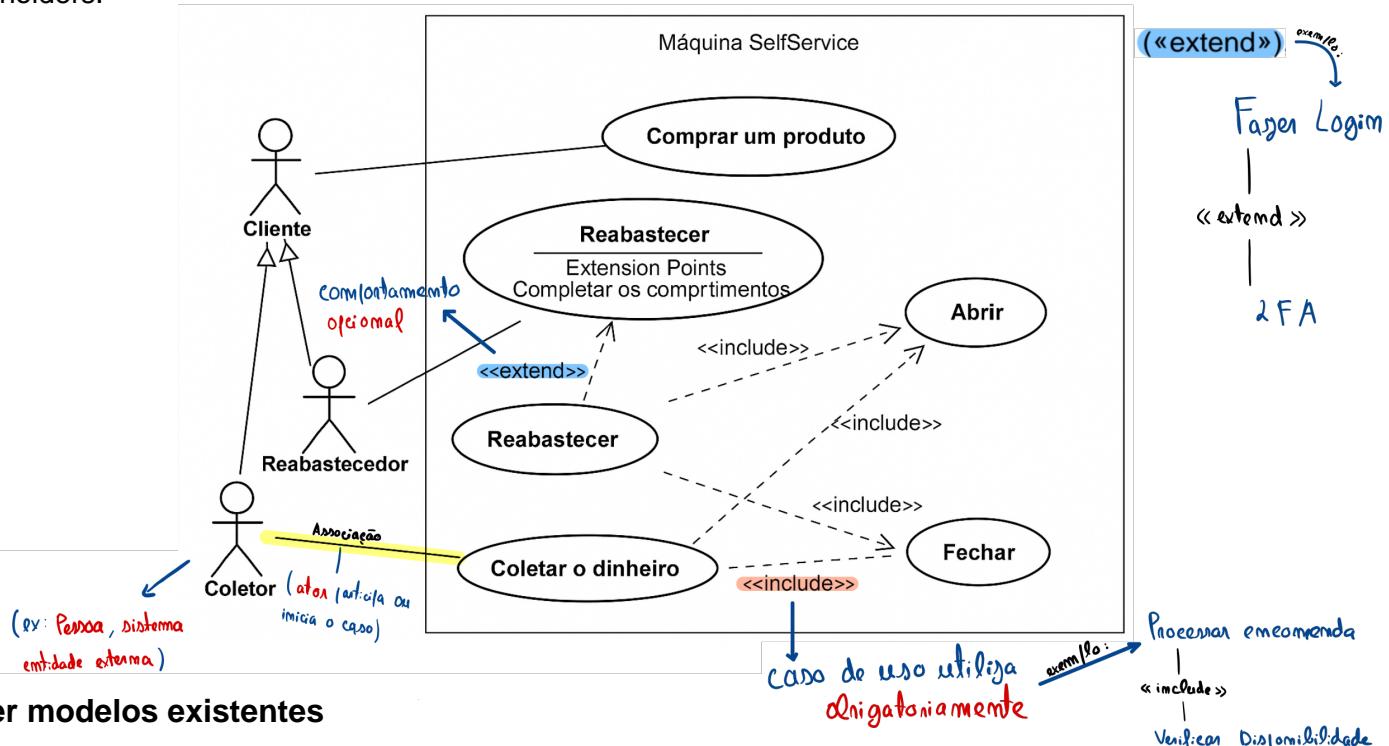
Técnicas de identificação incluem entrevistas, análise de processos existentes e observação de tarefas reais.

Ler e criar diagramas de casos de utilização

Os diagramas incluem atores (figuras externas), casos de uso (ações representadas em ovais), e relacionamentos:

- **Associação**, Inclusão («include»), Extensão («extend»), Generalização.

Esses diagramas mostram interações de alto nível, úteis para delimitar o sistema e comunicar com stakeholders.



Rever modelos existentes

É fundamental validar a coerência e correção dos diagramas:

- **Semântica:** os casos fazem sentido no contexto?

- **Sintaxe:** estão de acordo com a notação UML?

Verifica-se nomes claros, ausência de duplicações e ligações corretas entre atores e casos de uso.

Resumo Elaborado: Modelação Funcional com Casos de Utilização

Elementos essenciais de uma especificação de caso de uso :

Cada caso deve incluir:

- Nome claro e descritivo (ex: 'Emitir Fatura');
- Objetivo principal do ator;
- Fluxo principal de eventos (normal);
- Fluxos alternativos e exceções;
- Pré-condições (estado necessário antes de começar);
- Pós-condições (estado esperado no final da execução).

A modelação do contexto do problema: modelo do domínio/negócio

Caraterizar os conceitos do domínio de aplicação:

- Desenhe um diagrama de classes simples para capturar os conceitos de um domínio de problema.
- Apresente duas estratégias para descobrir sistematicamente os conceitos candidatos para incluir no modelo de domínio.
- Identificar construções específicas (associadas à implementação) que podem poluir o modelo de domínio (na etapa de análise).

Caraterizar os processos do negócio/organizacionais:

- Leia e desenhe diagramas de atividades para descrever os fluxos de trabalho da organização / negócios.
- Identifique o uso adequado de ações, fluxo de controle, fluxo de objetos, eventos e partições com relação a uma determinada descrição de um processo.
- Relacione os “conceitos da área do negócio” (classes no modelo de domínio) com fluxos de objetos nos modelos de atividade.