**METODOLOGIAS DE DESENVOLIMENTO DE SOFTWARE**

**(Estudo 1ª Freq)**

**🡪Introdução (1º PDF)**

**Eng-Software:**

-Disciplina especifica

-Desenvolvimento profissional

-Economia depende de software

-Sistemas controlados por software

-Custos representam parte do PIB(Produto Interno Bruto)

-Métodos, teorias e ferramentas

-Desenvolvimento profissional

**Custos Software:**

-Parte dominante no custo de sis. Informáticos

-Custo manutenção > custo do hardware

-Eng.Software preocupa-se com criação eficaz

**Desenvolvimento profissional de software:**

-Software: programa de PC e documentação associada

-Atributos dum bom software: Entregar funcionalidades e desempenho pedidos, sustentável, confiável e utilizável

-Eng.Software: Disciplina da engenharia relacionada com tds os aspetos de produção de software

-Atividades de Eng.Software: Especificação, desenvolvimento, validação e evolução

-Eng.Software VS Computer Science: engloba também as teorias e fundamentos, porém com prrática no desenvolvimento de software útil

-Eng.Software VS Sys.Eng: Sys.Eng engloba tds os aspetos de desenvolvimento de sistemas computer-based, incluindo hardware, software… Eng.Software é parte mais geral deste processo

**Atributos de software de qualidade**

-Manutenção: software desenvolvido de forma a poder englobar futuras mudanças feitas/pedidas pelo cliente

-Confiabilidade e Segurança: inclui um conjunto de características como segurança, proteção e confiabilidade. Software confiável não causa danos físicos, económicos no sistema. Usuários mal intencionados sem acessos ao sistema.

-Eficiência: Software não deve gastar desnecessariamente memória, tempo de processamento, etc…

-Aceitabilidade: aceitável para o tipo de usuário para o qual foi projetado. Compreensível, utilizável e compatível.

**Importância da Eng.Software:**

-Crescente dependência de sistemas de software

-Custo menor no desenvolver de software usando métodos e técnicas, que simplesmente “escrever” programas.

**Problemas associados ao software:**

-Heterogeneidade

-Alterações sociais e das regras de negócio

-Segurança e confiança

-Escala (desde embededd sys to cloud-based sys)

**Diversidade da Eng.Software:**

-Muitos tipos de software

-Métodos e ferramentas da Eng.Software

**🡪Processos de Software (2º PDF)**

**Processos de Software:**

-Conjunto de atividades para desenvolver software

-Variedade de processos de software, partilhando: Especificação, implementação, validação e evolução

-Modelo de processo de software

Tipos de processos de software:

-Baseados em planos

-Processos ágeis

-Na prática

-Não existe processos correto ou errado

**Modelos de processos de software:**

-Waterfall

-Etapas: Análise requesitos -> Desenho software -> Implementação -> Integração e testes -> Operação e manutenção

-Desvantagens: etapa só começa depois da anterior terminar; dificuldade em incluir alterações

-Problemas: divisão inflexível do projeto; utilização

-Desenvolvimento incremental

-Vantagens: custo alterações reduzido; fácil obtenção de feedback; entrega e deployment mais rápido

-Problemas: processo não é visível(difícil medir progresso); degradação da estrutura a cada incremento

-Integração e Configuração

-Etapas: especificação->pesquisa e analise

->alteração/adaptação->configuração->adaptação e integração

-Vantagens e desvantagens: custos e riscos reduzidos; entregas +rápidas; compromissos com os requisitos; falta de controle sobre evolução

**Especificação de software:**

-Processo para especificar: Funcionalidades; Restrições

-Processo de Eng. de Requisitos: Análise, especificação e validação dos requisitos

**Desenho e implementação:**

-Conversão de especificação em sis executável/usável

-Tarefas: Desenho do software; implementação

-Desenho e implementação

**Atividades de desenho:**

-Desenho da arquitetura

-Desenho da base de dados

-Desenho da interface

-Escolha e desenho dos componentes

**Implementação do sistema:**

-Implementação

-Design e implementação

-Programação/Desenvolvimento

-Debugging

**Validação de software:**

-Objetivo

-Métodos

-Testes

**Etapas dos testes:**

-Testes de desenvolvimento ou de componentes

-Testes de sistema

-Testes de aceitação

**🡪Engenharia de Requisitos (3ºPDF)**

**Outline:**

-Requisitos de utilizador e sistema

-Requisitos Funcionais e não funcionais

-Documento

-Especificação

-Processos da Engenharia

-Identificação e análise

-Validação

-Gestão

**Processo:**

-Enumerar

**Requisitos são:**

-Descrições do sistema

-Restrições do sistema

-Identificados durante processo de Engenharia de Requisitos

**Requisitos de utilizador:**

-Frases linguagem natural

-Diagramas

-Escritos para os clientes

-Restrições de utilização

**Requisitos de sistema:**

-Documento estruturado

-Descrições detalhadas das funcionalidades dos sis e serviços

-Define o que deve ser implementado

**System stakeholders(interessados no sistema)**

-Qualquer pessoa: relacionada com o sistema, tenha interesse no sistema

-Tipos de stakeholders: utilizadores finais, gestores do sis, donos do sis, externos

**Requisitos Funcionais:**

-Descrevem as funcionalidades do sis

-Dependem de: Tipo de software, utilizadores, tipo de sistema

-Requisitos funcionais de utilizador

-Requisitos funcionais do sistema

**Imprecisão dos Requisitos:**

-Falta de precisão: surgirão problemas

-Requisitos ambíguos: diferentes interpretações(por parte dos utilizadores ou da equipa de desenvolvimento)

**Requisitos Não Funcionais:**

-Definem propriedades e restrições ao sis

-Requisitos sobre o processo de software

-Podem ser mais críticos que os funcionais

-Podem afetar o sis de forma global

-Pode dar origem a vários funcionais

**Tipos de requisitos não funcionais:**

-Requisitos do Produto: especificam qual o comportamento do sis

-Requisitos Organizacionais: resultam das politicas da organização/empresa

-Requisitos externos: relacionados com fatores externos ao sis

**Objetivos(goals) e requisitos:**

-Problema: não funcionais são muito abstratos e imprecisos; difícil verificação

-Solução: criar objetivos; associar requisitos não funcionais aos goals

-Goals: definem intenções de usos dos utilizadores

**Processos de engenharia de requisitos:**

-Variam dependendo de: domínio de app, pessoas envolvidas, entidade que desenvolve requisitos

-Atividades comuns a todos os processos: identificação, análise, validação e gestão dos requisitos

-Na prática: atividades feitas de forma iterativa e intercalada

**Identificação e análise de requisitos:**

-Conhecida por descoberta de requisitos

-Trabalho entre equipa técnica e clientes

-Pode envolver todos stakeholders

**Problemas na identificação dos requisitos:**

-Stakeholders não sabem o que querem; expressam requisitos nos seus termos próprios

-Diferentes stakeholders podem ter requisitos com conflitos

-Requisitos mudam durante o processo de análise

**Especificação de Requisitos:**

-Processo de escrever requisitos  
 -Requisitos devem ser de fácil compreensão

-Requisitos de sistema

-Podem fazer parte de um contrato para desenvolvimento do sistema

**Formas de escrever os requisitos:**

-Linguagem Natural: cada frase deve expressar um requisito; Linguagem expressiva, intuitiva, universal

-Problemas: falta de clareza; confusão de requisitos; junção de requisitos

-Linguagem Natural Estruturada: cada campo fornece info sobre um aspeto do requisito

-Linguagem de descrição de Design: usa uma linguagem tipo uma de programação, mas com aspetos mais abstratos para especificar requisitos

-Notações Gráficas: Modelos gráficos, tabelas, UML, etc.. são comummente usados

-Especificações Matemáticas: notações são baseadas em conceitos matemáticos

**Use Cases:**

-Fazem parte do UML

-Identificam: atores; interação com o sist

-Modelo gráfico de alto nível: complementado por descrição tabular

-Diagramas de Sequência: usados para detalhar use cases; mostram a sequência de eventos num sistema

**Documento de Requisitos:**

-Documento oficial que indica o que é esperado do sis

-Deve incluir definção dos requisitos de utilizador bem com os requisitos do sis

-Não é documento de desenho

**🡪Introdução ao UML (4ºPDF)**

**O que é? :**

-Unified Modeling Language

-Linguagem de modelação gráfica

-Conjunto de notações gráficas

-Meta-modelo único

-Descrever e desenhar (ou modelar) sis de software

**Tipos de utilização:**

-Depende do user

-Sketch: Esboços

-Descrever detalhes do sis

-Nivel de abstração elevado

-Tipos de utilização:

-Forward engineering: desenhar diagrama antes de escrever cód; focar no que é importante; não pensar em cód

-Reverse engineering: desenhar diagrama depois de termos cód; usar sketches para explicar sis; documentação do sis

-Blueprint: detalhar um sis, ou partes

-Linguagem de programação: implementar um sis, ou partes

-Conceptual e modelação de software

**UML define:**

-Notações

-Meta-modelo

**Notações e meta-modelos:**

-Linguagens gráficas: Pouco rigor; embora muito informal

-Importância do meta-modelo: depende da utilização; sketching; blueprinting; UML as programming language

**Diagramas UML:**

-Define 13 diagramas: atividade; classe; comunicação; deployment; estrutura composta; pacote; máquina; tempo; …

**UML Válido:**

-O que é?: Respostas simples

-Na prática: não é assim tão simples; standard; pode ter diferentes interpretações; depende da utilização

-Importância: sketching ou blueprinting ; UML as programming language

**Significado do UML:**

-Existe especificação detalhada

-Significado: não existe especificação para o seu significado

-Diagrama UML: não existe tradução para source code; consegue-se ter ideia geral de como seria o código

**Análise de Requisitos:**

-Use cases: descirção da interação dos user com o sis

-Diagramas de classes: usando perspetiva conceptual

-Diagramas de atividades: fluxo de trabalho da empresa, mostando como o software interage com as atividades humanas

-Diagramas de estados: se o sis tiver life cycle interessante

-Não incluir nada técnico

**Design:**

-Diagramas de classes: idem

-Diagramas de sequência: idem

Diagramas de estados: idem

**Documentação:**

-Complemento á documentação

-Compreensão global do sis: focar aspetods importantes; não fazer diagramas detalhados do sis; documentação detalhada deve estar no cód

-Package diagram: mapa logico do sistemas

-Diagramas de classes: aspetos importantes de cada package

-Diagramas de interação: ajuda a compreender aspetos dos diagramas de classes

-Máquinas de estados: ajuda a perceber o ciclo de vida das classes

**🡪UML – Use Cases (5ºPDF)**

**O que são:**

-Técnica para capturar requisitos funcionais de um sis

-Descreve interações entre user do sis e o sis

-Mostra como é usado o sis

**Cenários:**

-Descrição de uma utilização do sis: sequência de passos; descrevem a interação entre o user e o sis

-Mesmo objetivo; exemplos diferentes, mas semelhantes

-Use case: conjunto de cenários; relacionados entre sí por objetivo comum

**Engenharia de requisitos:**

-Deve responder a: o sis serve para quê? ; Deve fazer o quê?

-Definir funcionalidades que o sis deve exibir

-E não como se implementa

**Use Cases:**

-Casos de uso

-Captar e ilustrar os requisitos do sis

-Representação do sis

-São cenários de utilização do sis

-Não existem sis “isolados”

-Definem fronteiras do sis com o exterior

-Especificam interações com o exterior

-Podem ser considerados diagramas de contexto

**Elementos Básicos:**

-Atores: users do sis ; quem o que interage com o sis

-Use Cases: funcionalidades oferecidas aos atores ; atores executa Use Cases

**Conteudo de um Use Case:**

-Como escrever

-Deve incluir: cenário principal de sucesso(caso normal, descrito por sequencia de passos numerados) ; outros cenários alternativos(através de extensões, pode ter sucessos ou falhas)

**Cada Use Case:**

-Tem um ator principal

-Podem existir outros atores

**Use Case:**

-Cada passo/etapa: representa uma interação entre ator e sis ; frases simples ; descrever de forma clara quem executa o passo ; descrever de forma clara o objetivo do ator

-Cada extensão: descrever alternativa ao caso normal ; descrever uma condição

-Como descrever extensão: indicar o passo do cenário principal que pode dar origem á extensão ; continuar lista de passos ; no fim indicar para onde deve continuar

-Excelente ferramenta para fazer brainstorming

Pode incluir: trigger(evento que inicia o use case) ; garantias(o que o sis deve garantir) ; pré- condições(o que deve ser assumido com verdade)

**Como identificar atores:**

-Quem vai beneficiar do sis?

-Quem está interessado em certo requisito do sis?

-Onde, na organização, é o que sis é usado?

-O sis usa algum recurso externo?

…

**Identificar Use Cases:**

-Um use case deve corresponder a uma grande funcionalidade bem definida do sis

-Numa ATM não interessa dizer “uma grande …”

-Pode ser: levantar dinheiro; pagar um conta; ver o saldo; ativar a via verde; …

**Diagramas de Use Cases:**

-UML: não indica conteúdo de um use case; especifica diagrama para apresentar os use cases

-Diagramas de Use Cases: uteis; não obrigatórios; tabela gráfica de conteúdos; diagrama de contexto

-Mostram: relações atores-use cases; que atores invocam quais use cases; que use cases usam outros use cases; fronteiras do sis

-Relações entre use cases: include ou uses; extends; herança

**Outras utilizações:**

-Conceber e validar arquitetura do sis

-Podem ser uma base para os testes do sis

-Ponto de referência em todas a etapas de desenvolvimento

**IMPORTANTE:**

-Use cases descrevem sis a um nível muito elevado de abstração

-Não diz como são implementados

-Disciplina de análise difícil

-Fácil resvalar para a tecnologia

-Fácil descer de nível de abstração

-Cuidado com especializações, extends ou include/uses

**🡪Diagramas de Atividades (6ºPDF)**

**-Modelação de fluxo?**

-Diagramas de interações: objetos ; mensagem trocadas entre objetos

-Diagramas de Gantt: objetos; desempenham atividade ao longo do tempo

-Diagramas de atividades: focam as atividades ; entre objetos

-Comportamento do sis

-Modelação dinâmica do sis

-Modelar/descrever: logica procedimental; processos de negócio; fluxos de trabalho

-Semelhantes aos fluxogramas

-Representam atividades

**Elementos:**

-Nó inicial: ponto de partida do diagrama

-Nó final: ponto final do diagrama

-Ações: ação concreta; realizada por um objeto

-Fork: fluxo de entrada; vários fluxos de saída concorrentes; ações em paralelo

-Join: vários fluxos de entrada; um fluxo de saída; marca o fim de um fork; apenas se avança depois de todas as entradas terem terminado

-Decisões: comportamentos condicionais; fluxo de entrada; vários fluxos de saída alternativos(apenas 1 é seguido); cada fluxo de saída tem uma guarda(bool condition)

-Merge: vários fluxos de entrada; um fluxo de saída; marca o fim de uma decisão

**Decomposição de ações:**

-Ações podem ser decompostas: em sub atividades

-Cada sub atividade: é uma atividade; com as suas ações

-Ligação entre ação e a sua decomposição: símbolo rake(ancinho) no diagrama

**Partições:**

-Diagramas de atividades mostram o que acontece: mas não mostram quem faz o quê

-Partições: dividir diagrama de atividades; mostrar quem faz o quê; Swim lanes

**Sinais:**

-Diagramas de atividades: ponto de entrada bem definido (invocação de método/rotina ; provoca transição de atividade)  
 -Sinais: atividades podem responder a sinais; eventos externos ao processo; atividade constantemente á escuta

-Representações gráficas: sinais temporais; atividade envia sinal e recebe sinal

**🡪Diagramas de Interação (7ºPDF)**

**Diagramas de interação:**

-Aspetos dinâmicos do sis

-UML fornece vários diagramas: de sequencia ; de colaboração

-Descrevem/capturam comportamento

-Num único cenário de utilização

**Diagramas de Sequência:**

-Focam-se no tempo

-Ordenação temporal das mensagens

-Mostram: interação e mensagens entre participantes

-Cada participante: representado por uma lifeline(linha vertical) ; mensagens de/para linhas de vida; ordenadas de cima para baixo

-Participantes: representam objetos de classe

-LifeLine dos participantes: representa o tempo de vida no cenário modelado; caixa ou barra de ativação

**Diagramas de comunicação:**

-Ou diagramas de colaboração

-Focam-se na organização dos objetos e nos dados

**Controlo centralizado VS distribuído:**

-Centralizado: processamento feito no mesmo local; mais simples

-Distribuído: processamento distribuído por todos os participantes; preferível; “Junta” dados com comportamentos; muito orientado a objetos

**Criar e Apagar Participantes:**

-Criar: mensagem diretamente para a caixa do participante (seta, nome da mensagem: “new”)

-Apagar: marcado com um X; uma mensagem para o X indica que o participante apaga outro; X no fim da linha significa que o participante se apagou a ele próprio

**Quando apagar participantes? :**

-Em ambiente garbage colletion

-Para fechar ou terminar operações

**Ciclos e condições:**

-Diagramas de sequencia: mostrar interações entre objetos

-Como modelar: interaction frames; marcar parte do diagrama; cada frame tem um operador e uma “guarda”

-Operadores: tipo de operação associada ao frame; ciclo(loop); condições(alt)

-Guarda: quando o frame é executado

-Ciclos: 1 frame; operador(loop); guarda(controlo do ciclo)

-Condições: vários frames; operador(alt); guarda(condição associada ao frame)

**Operadores comuns:**

-alt: múltiplos frames alternativos, apenas cujos a condição seja verdadeira serão executados

-opt: frames opcionais, frame executado apenas se a condição for verdadeira

-par: frames executados em paralelo

-loop: ciclo, frame executa varias vezes, guarda indica interação feita

-region: região crítica, apenas permite executar uma thread ao mesmo tempo

-neg: interação que não é válida

**Passagem de Dados:**

-Opção: setas de retorno; através de parâmetros nas mensagens

-Alternativamente: data tadpoles; indicam o movimento dos dados

**Mensagens síncronas e assíncronas:**

-Sincronas: chamador da mensagem e espera operação terminar; necessita de esperar mensagem; notação(seta fechada)

-Assincrona: chamador passa imediatamente apos enviar mensagem; não precisa esperar resposta; usada em sis multi-thread: notação(seta aberta)

**Diagramas de comunicação:**

-Foco nos links de dados entre os participantes

-Participantes: colocação no diagrama

-Links: entre participantes; mostram como cada um se liga a outro

-Podem representar: instâncias de associação entre classes; links transientes

-Mensagens entre os participantes: links, numeração de mensagens, etc

-Não tem notação precisa para especificar lógica de controlo

**Sequência VS Comunicação:**

-Equivalentes entre si: a partir de um, consegue-se outro; diagramas de comunicação não tem notação para logica de controlo  
 -Sequência: usar quando for focar a sequência de chamadas

-Comunicação: quando focar ligação entre participantes; bom para explorar diferentes alternativas (facilidade na alteração)

**🡪Desenvolvimento Ágil de Software**

**Desenvolvimento rápido de software:**

-Desenvolvimento e entrega rápida de software é muito importante: dinâmica de negócios é rápida (requisitos alteram-se; difícil produção de requisitos estável); software tem de adaptar-se rapidamente

-Métodos baseados em planos: essenciais para alguns tipos de software; pouco adequados para alguns negócios

-Métodos Ágeis de Desenvolvimento: objetivo de reduzir drasticamente o tempo de entrega do software

-Especificação, desenho e implementação intercalada

-Sistemas são desenvolvidos como uma série de versões: interessados participam na avaliação das versões

-Entregas frequentes de novas versões: avaliação

-Ferramentas de apoio ao desenvolvimento

-Documentação mínima

**Planos e métodos ágeis:**

-Desenvolvimento baseado em planos: etapas de desenvolvimento separadas; pode não ser Waterfall; iterações ocorrem dentro de cada atividade

-Métodos ágeis: especificação, desenho e implementação intercalados, output decidido num processo de negociação

**Métodos ágeis:**

-Insatisfação com as metodologias pesadas

-Objetivo: reduzir peso do processo de desenvolvimento(reduzir documentação); responder de forma rápida ás alterações dos requisitos

**Agile Manifesto (Princípios):**

-Envolvimento do cliente

-Entrega incremental

-Pessoas não processam: habilidades da equipa de desenvolvimento devem ser exploradas e reconhecidas. Devem adoptar as suas maneiras de trabalhar

-Abrace a mudança: projetar o sistema para acolher futuras mudanças

-Simplicidade: sempre tentar eliminar a complexidade do sistema

**Aplicação de métodos ágeis:**

-Desenvolvimento de software de pequena/média escala

-Desenvolvimento de sistemas á medida

-Foco em equipas pequenas e muito bem integradas

**Técnicas de desenvolvimento ágil:**

-Extreme programming: método ágil; leva abordagem de desenvolvimento iterativo ao extremo(varias releases diárias; incrementos entregues a cada 2 semanas, …)

-Boas práticas: Planejamento incremental; pequenas versões; design simples; Test-First Development, Refactoring, Pair programming, Colletive ownership, Continuous integration, Sustainable pace, On-site costumer

**User stories como requisitos:**

-Em XP, cliente ou utilizador faz parte da equipa de desenvolvimento

-Requisitos de utilizador expresos através de cenários/user stories

-Clientes escolhe as histórias para incluir na próxima release

**Refactoring:**

Consenso generalizado em Eng.Software: desenhar software para ser alterado facilmente; gastar tempo e esforço e antecipar alterações; reduz custos

-XP: defende não antecipar alterações; impossível antecipar alterações de forma fiável; melhorias constantes do código

-A equipa procura possíveis melhorias no software e implementa-as

-Melhora a compreensão do software, reduzindo necessidade de documentação

-Alterações mais fáceis de implementar

**Testes antes da implementação:**

-Testes em XP: executados sem qualquer alteração; conceito central

-Características dos testes em XP: antes do desenvolvimento; incrementais criados apartir de cenários; participação dos clientes nos testes; ferramentas para execução dos testes automática

**Participação do cliente:**

-Papel no processo de testes: desenvolver testes de aceitação para os cenários possíveis

-Faz parte da equipa de desenvolvimento: escreve teste á medida que é desenvolvido o software; todo o código é validado caso seja o que o cliente precisa

**Automação de testes:**

-Testes escritos como componentes executáveis: antes da implementação das funcionalidades; devem ser isolados, verificar se output está correto

-Usar um framework de testes: facilita criação e execução de testes

-Com testes automáticos: executar testes de forma rápida e fácil

**Programação Por Pares:**

-Programadores trabalham aos pares

-Código partilhado por vários elementos da equipa

-Processo informal para rever código

-Encoraja o refactoring, pois todos beneficiam

**Gestão de Projetos Ágeis:**

-Responsabilidade dos gestores de projeto

- Gestão de projetos típica: baseada em planos, gestores criam plano

-Gestão de projetos ágeis: obriga abordagem diferente

**Scrum:**  
 -Método ágil de desenvolvimento de software: foco na gestão do desenvolvimento iterativo

-Composto por 3 etapas: Fase incial, Sprint Cycles, Fim do projeto

-Terminologia: Develpoment team, Product backlog, Product owner, Scrum, Scrum Master, Sprint, Velocity

Beneficios: produto partido por partes pequenas e de fácil compreensão; requisitos não estáveis não afetam progresso; clientes tem entregas incrementais a tempo; aumento da confiança entre clientes e equipa

**Escalar Métodos ágeis:**

-Métodos ágeis provaram ser bem sucedidos: equipas pequenas, projetos pequena/media dim)

-Razão para o sucesso: melhoria na comunicação, facilita colaboração

-Escalar métodos ágeis: projetos maiores e maior duração; várias equipas;

**Scaling Out e Scaling Up:**

-Scaling Up: usar métodos ágeis; projetos não podem ser desenvolvidos por equipas pequenas

-Scaling Out: introdução de métodos ágeis em organizações grande

**Métodos ágeis-Problemas:**

-Muito informal: imcompativel com alguns aspetos legais relacionados com contratação de serviços

-Apropriados para software novo: não recomendados para manutenção; grande parte dos custos de software está associado á manutenção de software existente

-Desenhados para equipas pequenas