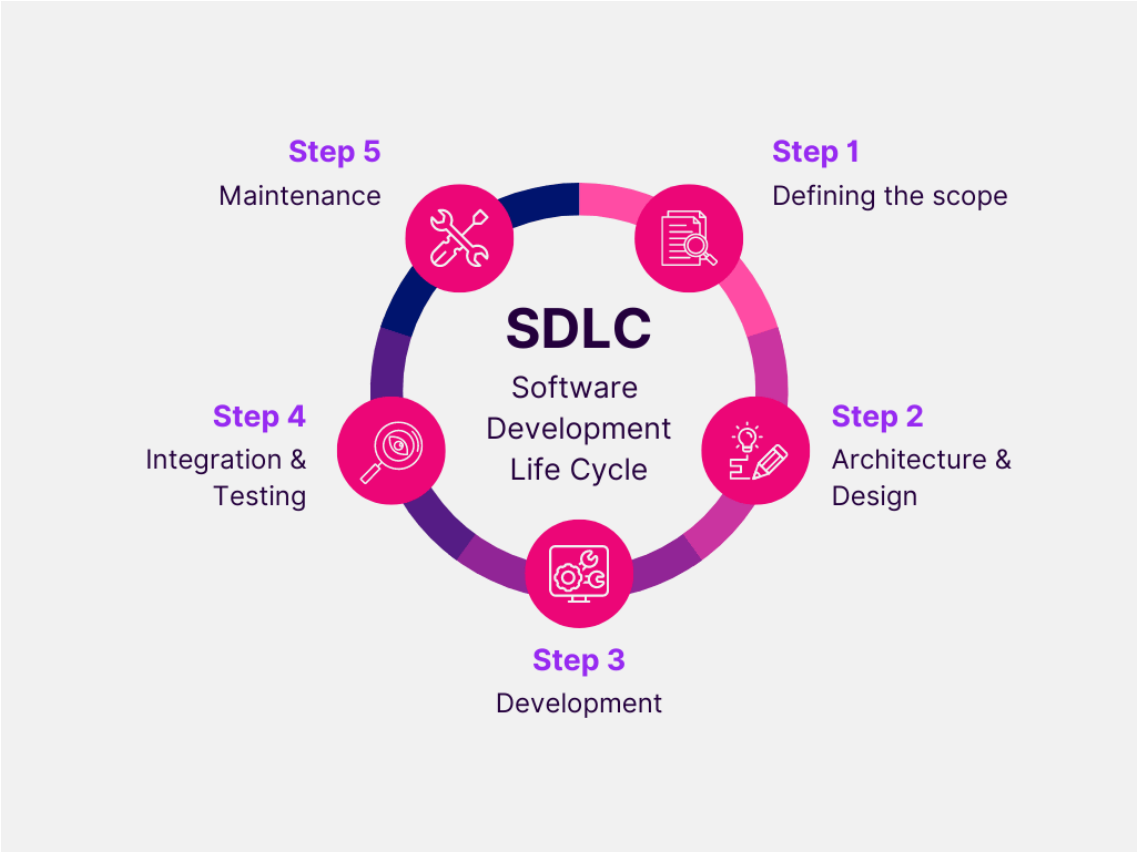
A) O que é que está incluído no SDLC?

O SDLC e o trabalho do Analista

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

SDLC ou Software Development Life Cycle é um processo que produz software com a mais alta qualidade e menor custo no menor tempo possível. SDLC fornece um fluxo bem estruturado de fases que ajudam uma organização a produzir rapidamente software de alta qualidade, bem testado e pronto para uso em produção.



• 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).

Step 1. Definir o âmbito

Durante esta fase os participantes no processo falam sobre as necessidades do produto final.

Step 2. Arquitetura e Design

Neste fase, os desenvolvedores criam um sistema de alto nível (diagramas UML por exemplo) baseado nos requisitos.

Todas as “parties” incluindo o cliente são consultadas durante.

Step 3. Desenvolvimento
A equipa começa a trabalhar depois do design ser escolhido e os requisitos estabelecidos.

Step 4. Integração e Testes
É revelado se o produto atende à qualidade exigida.

• Definir o temo “processo de software” (*software process*)

Um processo de software é o conjunto de atividades e resultados associados que produzem um produto de software.

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

As atividades de análise de domínio focam na compreensão do contexto do negócio e das necessidades dos usuários.
As atividades de especificação de software traduzem os requisitos do domínio em soluções técnicas.

• Descrever o papel e as responsabilidades do Analista no SDLC

Um analista no SDLC é crucial para atividades como: planejamento (coleta e valida requisitos), definição de requisitos (cria modelos de processos), projeto do sistema (colabora no design), desenvolvimento (participa na definição de casos de teste), teste (valida requisitos), implementação.

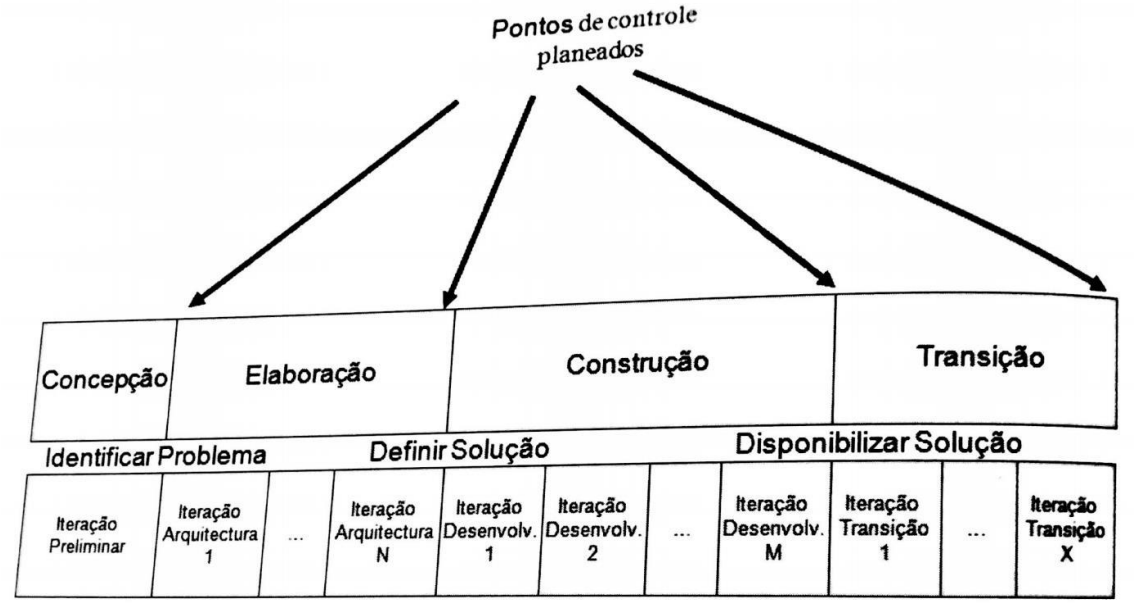
• Distinguir as competências de “análise de sistemas” das de “programação de sistemas”, em engenharia de software. Relacionar com os conceitos de “soft skills” e “hard skills”.

Análise de Sistemas:
- Hard Skills:
  - Modelagem de processos
  - Coleta e documentação de requisitos
- Soft Skills:
  - Comunição
  - Resolução de problemas

Programação de Sistemas:
- Hard Skills:
  - Linguagem de programação (ex: Java)
  - Desenvolvimento de software
- Soft Skills:
  - Atenção aos detalhes
  - Trabalho em equipe

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

Concepção: Estabelecer a visão do projeto, definir requisitos iniciais e planejar o escopo.
Elaboração: Refinar requisitos, arquitetar a solução e mitigar riscos técnicos.
Construção: Desenvolver o sistema completo, realizar testes.
Transição: Preparar o sistema para produção e garantir aceitação pelos usuários.

● O OpenUP pode ser considerado “método ágil”?

Sim, devido à sua abordagem iterativa e incremental, foco na colaboração e comunicação continuas.

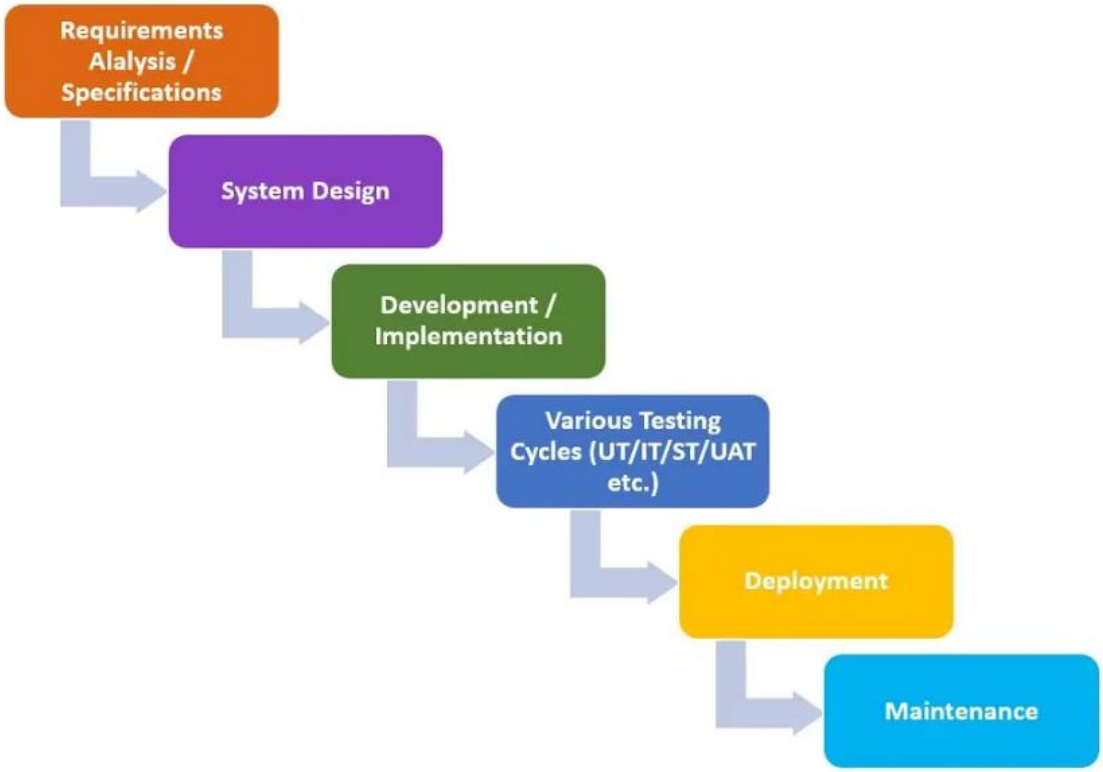
● Porque é que o UP se assume como “orientado por casos de utilização, focado na arquitetura, iterativo e incremental”?

-Casos de Utilização: São usados para capturar requisitos de forma clara e centrada no usuário.

-Arquitetura: É definida precocemente para mitigar riscos técnicos e orientar o desenvolvimento.

-Iterativo e Incremental: Divide o projeto em ciclos curtos de desenvolvimento, permitindo adaptação contínua e entrega incremental de funcionalidades.

● 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”?).

-Iterações curtas e sucessivas, entregando ao cliente de forma incremental ao invés de uma entrega única final.

-Flexibilidade para responder a mudanças nos requisitos.

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

Modelação visual e a UML

● Justifique o uso de modelos na engenharia de sistemas

-Facilitam a compreensão e comunicação das complexidades do sistema.
-Permitem análise e verificação antes da implementação física.

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

-Modelos Funcionais:
- Focam no que o sistema deve fazer
- Diagramas de Casos de Uso, fluxo de atividades, Diagramas de sequência

-Modelos Estáticos:
- Descrever a estrutura estática do sistema
- Diagramas de classes, Diagramas de Componentes, Diagramas de pacotes.

-Modelos de Comportamento:
- Como o sistema se comporta e interage com os usuários.
- Diagramas de sequência, Diagramas de estados, Diagramas de atividades.

● Enumerar as vantagens dos modelos visuais.

- Servem como uma linguagem comum entre diferentes partes interessadas, melhorando a comunicação

● Explicar a organização da UML (classificação dos diagramas)

- Diagramas de Estrutura:

- Diagrama de Classes: Descreve a estrutura estática do sistema, mostrando classes, atributos, operações e seus relacionamentos.

- Diagrama de Objetos: Representa uma instância particular de uma classe e seus relacionamentos em um momento específico.

- Diagrama de Componentes: Mostra as partes físicas do sistema e suas inter-relações, destacando a implementação física e a dependência entre os componentes.

- Diagrama de Pacotes: Organiza elementos do modelo em grupos (pacotes), mostrando a estrutura de agrupamento e dependências entre os pacotes.

-Diagramas de Comportamento:

- Diagrama de Casos de Uso: Descreve funcionalidades do sistema a partir da perspetiva do usuário, mostrando atores, casos de uso e suas interações.

- Diagrama de Sequência: Representa interações entre objetos em uma sequência temporal, mostrando como mensagens são trocadas ao longo do tempo.

- Diagrama de Estados: Descreve estados possíveis de um objeto e transições entre esses estados em resposta a eventos.

- Diagrama de Atividades: Modela o fluxo de trabalho do sistema, mostrando atividades e a sequência de ações necessárias para realizar uma atividade específica.

- Diagrama de Máquina de Estados: Mostra como um objeto responde a eventos ao longo do tempo, especificando estados, transições e ações associadas.

● Caraterizar 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.

- Fase Inicial:
    - Diagrama de casos de uso
- Fase Design:
    - Diagrama de classes
    - Diagrama de sequência
    - Diagrama de estado
    - Diagrama de atividades
- Fase de Implementação:
    - Diagrama de Componentes

● 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 Implementação1, Diagramas de Pacotes1 e Diagramas de Componentes.

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

Práticas de engenharia de requisitos

● Distinguir entre requisitos funcionais e não funcionais

- Requisito Funcional: As tarefas que o sistema é capaz de realizar.

- Requisito Não Funcional: O quão bem deve o sistema fazer a operação.

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

- Projetos Pequenos: Entrevistas, Análise de Documentação, Histórias de Usuário.
- Projetos Médios a Grandes: Workshops, Questionários.
- Projetos Inovadores: Protótipos, User Stories.

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

- Abordagem centrada em cenários: Para garantir necessidades reais do usuário.
- Abordagem centrada em produto: Para cobertura completa de requisitos técnicos.

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

- Atributos de qualidade = requisitos não funcionais

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

- A recolha de requisitos envolve apenas a coleta das necessidades e desejos dos stakeholders. Enquanto que a determinação de requisitos vai além da coleta, inclui entender profundamente o problema, analisar, validar e priorizar requisitos.

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

- S.M.A.R.T. -> Specific, Measurable, Attainable, Relevant and Time-sensitive
- Para ser bem formulado tem de seguir o SMART.

● Identifique requisitos bem e mal formulados (aplicando os critérios do ISO-IEEE 29148)

- Tem de ser satisfeita por um sistema.
- 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].

https://readcache.xyz/api/p?url=https%3A%2F%2Fmedium.com%2Fanalysts-corner%2Ften-cosmic-truths-about-software-requirements-edd33292a456

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

- As regras de negócio orientam o comportamento da organização e são independentes da implementação técnica (como se fosse dito por uma pessoa normal)
- Conceito de requisitos são descrições especificas sobre como o sistema deve ser implementado para atender às necessidades (do tipo “o sistema tem de”, nenhuma pessoa fala assim)

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

- Usar Use Cases

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

- É desafio de interação humana devido à necessidade de compreender profundamente as necessidades dos usuários. A habilidade de comunicar e colaborar efetivamente com stakeholders é essencial para capturar requisitos de forma precisa e relevante.

Modelação funcional com casos de utilização

● Descrever o processo usado para identificar casos de utilização.

1. Identificar Atores
2. Identificar Funcionalidades
3. Identificar Casos de Uso

● 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”.

- Desenvolvimento orientado por Use Cases” significa que o processo de desenvolvimento de software é guiado pela identificação, descrição e implementação dos casos de uso que descrevem como os usuários interagem com o sistema.

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

- Foco em Valor para o Usuário:

-Os casos de uso devem priorizar a entrega de valor direto aos usuários finais. Isso significa concentrar-se nas funcionalidades que realmente importam para os usuários, contribuindo para a satisfação e eficácia do sistema.

- Iteração e Evolução Contínua:

-Adotar uma abordagem iterativa e incremental no desenvolvimento de casos de uso, permitindo melhorias contínuas e adaptações às mudanças nos requisitos e nas necessidades dos usuários ao longo do tempo.

- Foco em Resultados:

    - Definir claramente os resultados esperados de cada caso de uso. Isso envolve não apenas descrever as interações entre atores e sistema, mas também especificar quais mudanças concretas ocorrerão no ambiente de negócios ou no sistema após a execução do caso de uso.

- Engajamento Colaborativo:

    - Incentivar a colaboração ativa entre todos os stakeholders envolvidos no processo de especificação e implementação dos casos de uso. Isso inclui desenvolvedores, analistas de negócios, usuários finais e outras partes interessadas relevantes.

- Visualização e Comunicação Eficaz:

    - Utilizar técnicas visuais e ferramentas apropriadas para representar os casos de uso de forma clara e compreensível. Isso facilita a comunicação entre todos os envolvidos e ajuda a garantir uma interpretação consistente dos requisitos.

- Foco em Arquitetura:

    - Considerar os casos de uso como uma parte integrante da arquitetura do sistema. Isso implica projetar o sistema de forma que os casos de uso sejam suportados de maneira eficiente e que a arquitetura geral do sistema permita uma implementação robusta dos requisitos.

● Explicar a relação entre requisitos e os casos de utilização

● Identificar as disciplinas e atividades relacionadas aos requisitos no OpenUP

- Disciplinas:
    - Gerenciamento de Requisitos
    - Análise e Design
- Atividades:
    - Análise de requisitos
    - Priorização de Requisitos

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

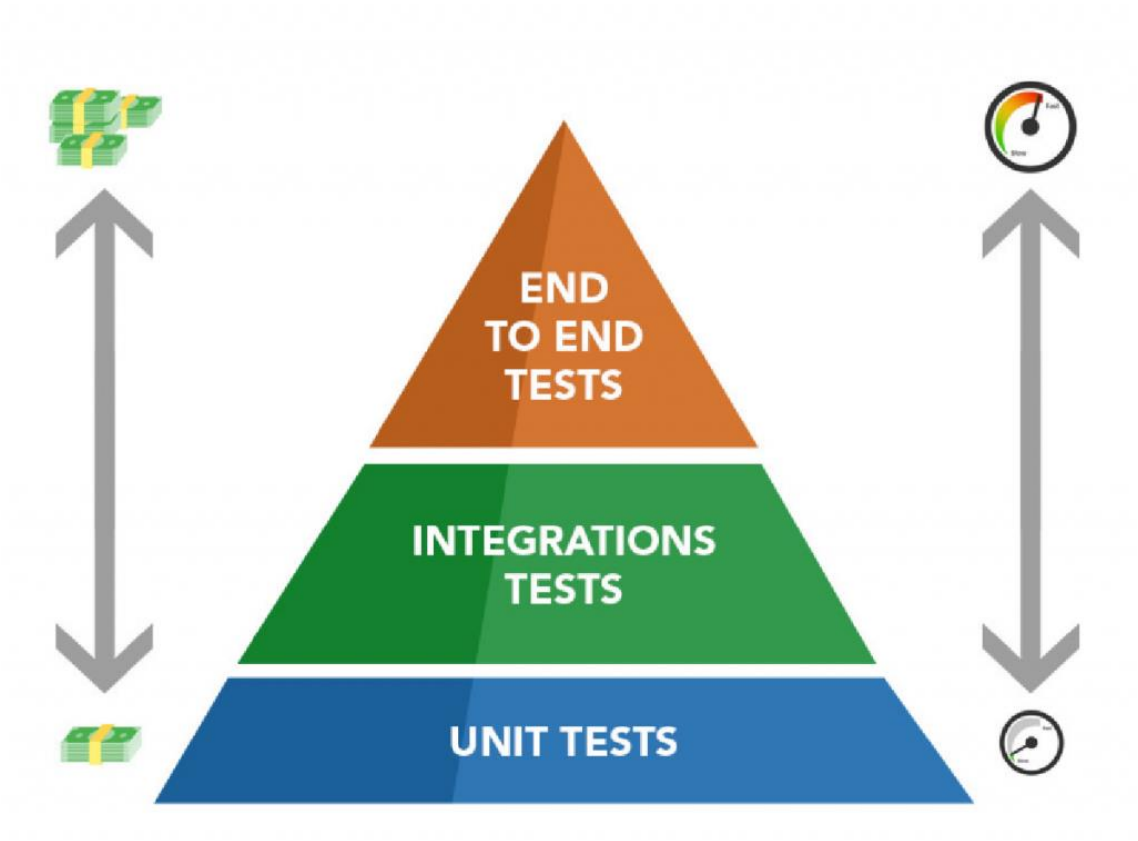
● Identifique o uso adequado de classes de associação.

C) Práticas selecionadas na construção do software

Garantia de qualidade

● Identifique as atividades de validação e verificação incluídas no SDLC

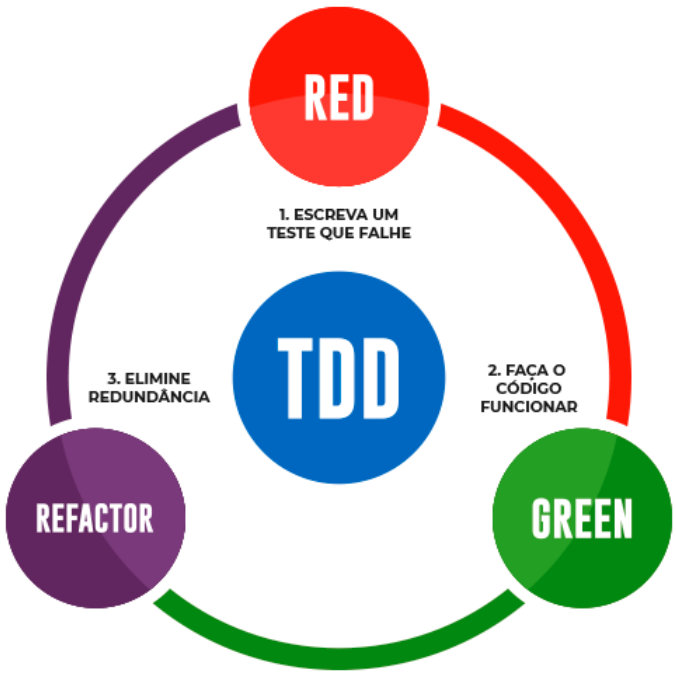
● Descreva quais são as camadas da pirâmide de teste



● Descreva o assunto/objetivo dos testes de unidade, integração, sistema e de aceitação

- Testes Unitários: Testam componentes individuais de forma isolada
- Testes de Integração: Verificam a interação entre módulos
- Testes de Sistema: Testam o sistema como um todo

● Explique o ciclo de vida do TDD

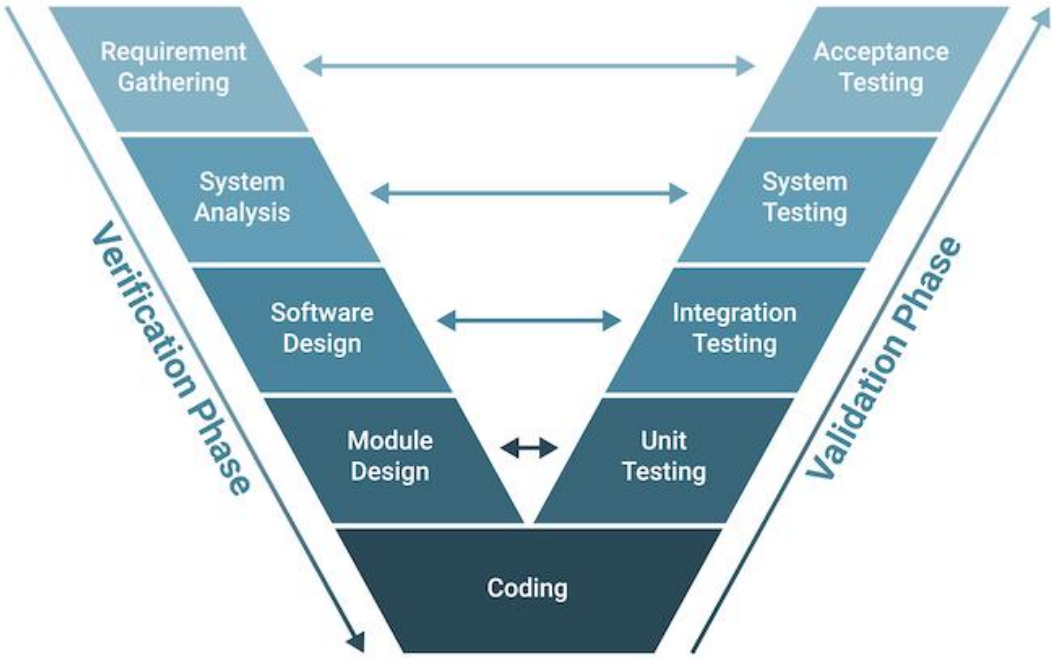


● Descreva as abordagens “debug-later” e “test-driven”, de acordo com J. Grenning.

- Debug-later: escrever primeiro o código e depois testar
- Test-Driven: TDD

● Explique como é que as atividades de garantia de qualidade (QA) são inseridas no processo de desenvolvimento, numa abordagem clássica e nos métodos ágeis.

● O que é o “V-model”?



O Modelo V [2]é um modelo conceitual de Engenharia de Sistemas/Desenvolvimento de Produto visto como melhoria ao problema de reatividade do modelo em cascata.

Os testes têm resultados de maior efetividade, uma vez que são testados contra requisitos e não contra especificações;

● 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 “*custommer facing*”.

Integração contínua/Entrega Contínua

● Identificar os passos típicos de um ciclo de CI

● Distinguir entre C. Integration, C. Deployment e C. Delivery

- Continuous Integration (CI):

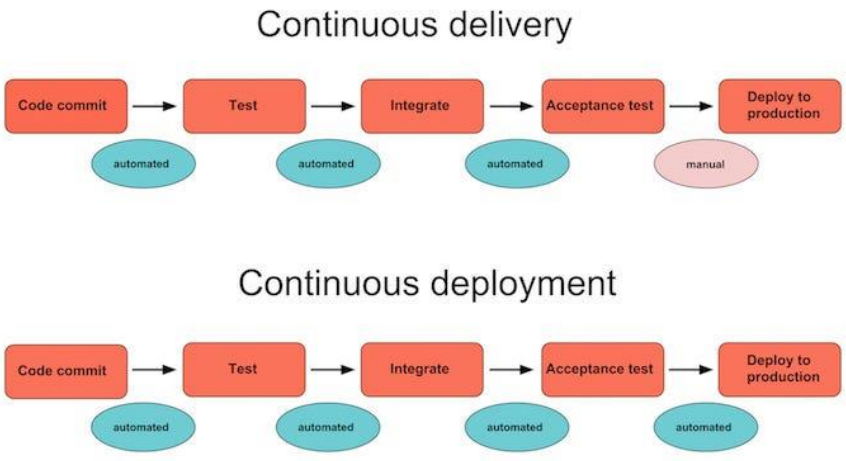
Prática de integrar frequentemente o código ao repositório (github) compartilhado e realizar testes automatizados para detectar problemas de integração.

- Continuous Deployment (CD):

Extensão do CI onde cada alteração de código aprovada é automaticamente implantada em produção.

- Continuous Delivery (CD):

Abordagem para garantir que o software seja sempre implantável em produção, permitindo entregas frequentes, mesmo que a implantação seja manual.



● 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?

- Detectar e corrigir problemas de integração rapidamente.
- Facilita entregas frequentes e previsíveis

● Explicar as tarefas e *outcomes* englobados numa "*build*"

- Outcomes vão ser os resultados das tarefas da build

● Explique o sentido da prática "*continuous testing*".

- Envolve a integração continua de testes automatizados.

D) 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."

- É verdade pois a abordagem em cascata prioriza decisões e compromissos precoces, dificultando correções significativas em fases posteriores.

● Explique o sentido da frase: "O desenvolvimento iterativo centra-se em ciclos curtos e orientados para a geração de valor"

- Cada ciclo visa produzir valor tangível ao cliente, permitindo adaptações baseadas em feedback contínuo.

● Discuta o argumento: "A abordagem ágil dispensa o planeamento do projeto".

- Falso, o planejamento é feito de forma iterativa e adaptativa.

● Identifique vantagens de estruturar um projeto em iterações, produzindo incrementos funcionais com frequência.

- Permite ajustes contínuos com base no feedback do usuário.

● Caracterizar os princípios da gestão do *backlog* em projetos ágeis.

● 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.

- Situações que os requisitos são bem definidos desde o inicio do projeto, ou há pouco espaço para mudanças ao longo do desenvolvimento.

● 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.

- Coisas que o utilizador quer fazer, do ponto de vista do utilizador.

● 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?

● Compare “Persona” com Ator com respeito a semelhanças e diferenças. Qual a utilidade das Personas no desenvolvimento das US?

● 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.