

# SDLC → System Development Life Cycle

Está dividido em 4 etapas

PADI

## Planejamento

Valor para o negócio

Decisões se o projeto deve avançar

Criação de um plano de trabalho

## Análise

Quem vai usar o sistema?

Funcionalidades do sistema

Onde e quando vai ser utilizado

## Design

Design da arquitetura

Design da UI, DB, etc

Planejamento da arquitetura e logística do SI

## Implementação

Construção de todo o sistema

Instalação de componentes

## Análise do SDLC →

- Analisa a situação de negócio
- Identifica oportunidades de evolução
- Desenha o sistema de informações

Sunge com o objetivo de criar valor para a organização

## O Unified Process / Open Up

### - Estrutura

O método Unified Process prevê quatro fases principais:  
CECT

### → Conceção

- Há uma concordância na visão do projeto?
- Deve o projeto avançar?

### Características:

- Produção de protótipos conceituais
- Levantam casos de utilização

**Milestone:** Examinar os pros / contras do projeto e decidir se o mesmo deve avançar

→ Elaboração

- Existe concordância na arquitetura a usar?
- O valor produzido e o risco são aceitáveis?

Características

- Implementações de alguns componentes
- Algum código implementado (~10%)
- Identificar dependências / requisitos

**Milestone:** Validação da arquitetura

→ Construção

- O sistema está pronto o suficiente para entregar?

Características:

- Construir, desenhar, implementar e testar

**Milestone:** O SI está pronto para ser entregue, todas as funcionalidades foram desenvolvidas e todos os testes efetuados.

O produto está pronto para o teste beta

→ Transição

O SI está pronto para ser entregue?

Características:

- Corrigir bugs existentes
- Estabilização e entrega
- Documentação produzida e organizada

**Milestone:** Aprovação do cliente após rever e aceitar o projeto

O Open Up pode ser considerado um **método ágil** que promove as melhores práticas de desenvolvimento de software:

- Trabalho em equipa
- Desenvolvimento iterativo
- Testes frequentes e adaptações às mudanças

É também orientado por casos de utilização



a equipa utiliza CAV para orientar todo o processo

ficado na arquitetura



foca-se no sentido de minimizar o risco e organizar o processo

Iterativo e incremental



Promove práticas que permitem à equipa ter contínuo feedback dos stakeholders, demonstrando assim, o valor de cada incremento

Principais características dos métodos ágeis

Os métodos ágeis (OpenUP, Scrum) surgiram como uma solução para as metodologias waterfall



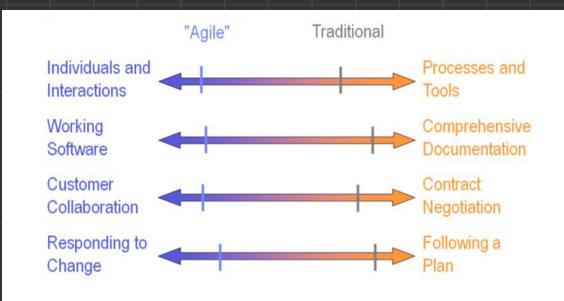
- Sequencial e linear

- Não se volta atrás

- Não há flexibilidade - Não há espaço para alterações ou erros, é tudo feito conforme planeado inicialmente

A mudança no mundo do software é inevitável e, usando métodos como o da cascata torna-se muito difícil adaptarmo-nos a estas mudanças.

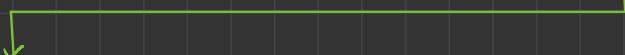
Ao fixarmo-nos com o plano inicial ao longo de toda a duração do projeto, vamos mais tarde chegar à conclusão que erros que aparecem posteriormente não poderão ser agora corrigidos



Este tipo de métodos segue uma metodologia que:

- é incremental e interativa
- Os desenvolvedores trabalham por módulos
- Ciclos curtos e entrega de valor frequente

- Tem como objetivo dar uma resposta rápida às mudanças contra-tempo
- Toda a gestão é efetuada através de um backlog



- Os items com maior prioridade encontram-se no topo
- Prioridade dos itens pode ser alterada a qualquer momento

★ O trabalho por iterações tem muitas vantagens em relação aos métodos previamente usados:

- Cada iteração pode ser testada antes da entrega
- Mais fácil integrar feedback de terceiros
- Maior capacidade de resolução de bugs

## Modelação Visual

Para além dos métodos ágeis são usados muitas formas de visualização dos seus sistemas uma vez que estes:

- Ajudam a gerir a complexidade
- Especificam estrutura e comportamento do sistema

- Serve como referência

Para isto existem 3 tipos diferentes de modelos:

- Modelos estáticos:

Representam as partes estáticas do sistema (classes, objetos e interfaces)

Diagramas característicos:

- Class
- Deployment
- Package

- Modelos funcionais:

Representam todas as funções desempenhadas pelo sistema

Diagramas característicos:

- Atividade

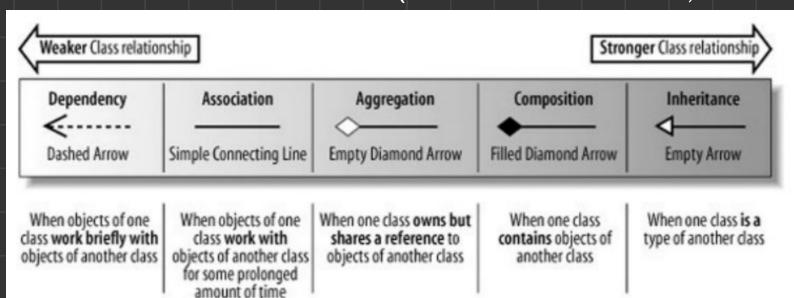
- Modelos de comportamento

Descrevem a interação no sistema. Representa interações entre os diagramas estruturais. Mostra a maturidade dinâmica do sistema

Agora em relação aos tipos de diagramas temos

Diagramas de classe →

Descreve a estrutura do sistema, mostrando as suas classes, atributos, operações e as relações entre objetos



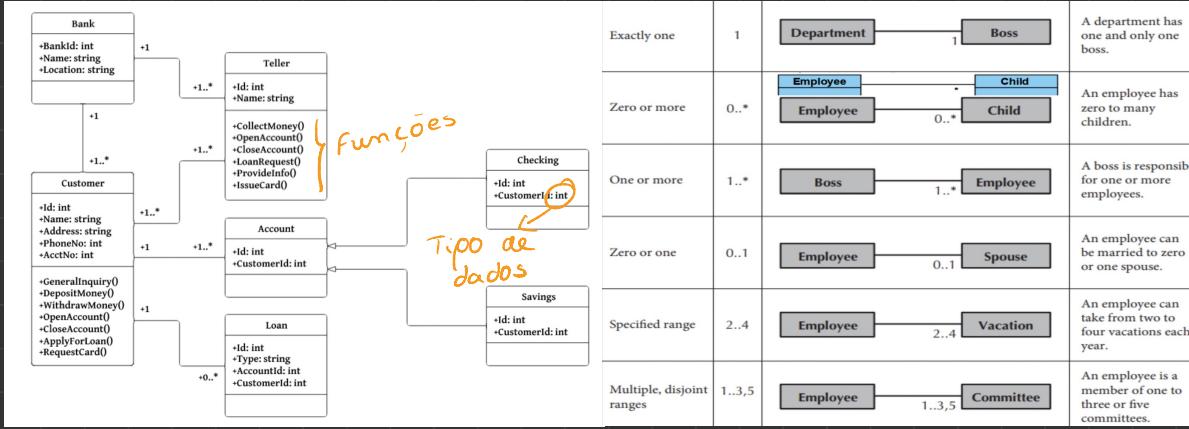
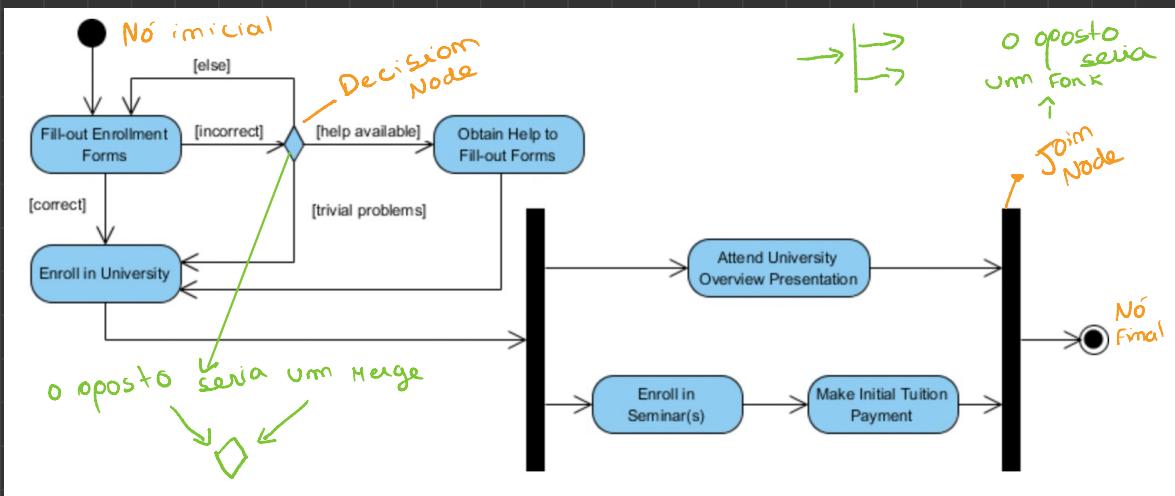


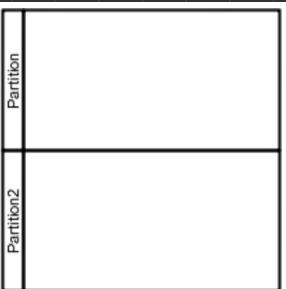
Diagrama de atividades → Modela o fluxo de uma atividade para a outra



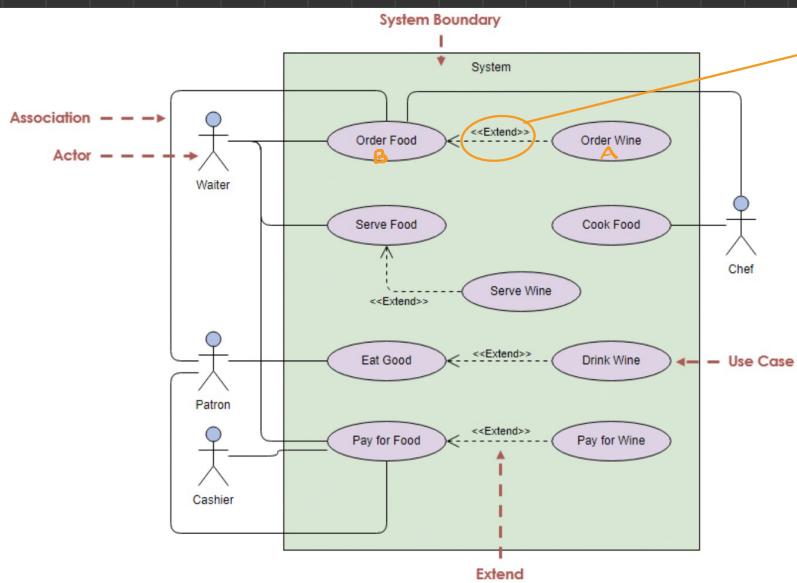
## Swimlane and Partition

A way to group activities performed by the same actor on an activity diagram or to group activities in a single thread

Também pode aparecer com várias caixas destas



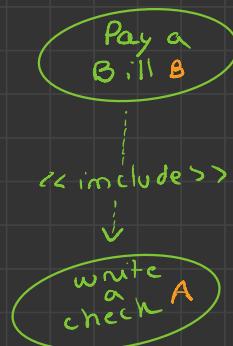
**Diagrama de Casos de Utilização** → Forma principal de identificarem requisitos de sistema / software para um novo programa de software em desenvolvimento. CaU especificam o comportamento esperado (o que), e não o método exato de fazer isso acontecer.



Significa que a história A inclui B.

comportamento de B pode incorporar comportamento de

Existe também a opção de incluir

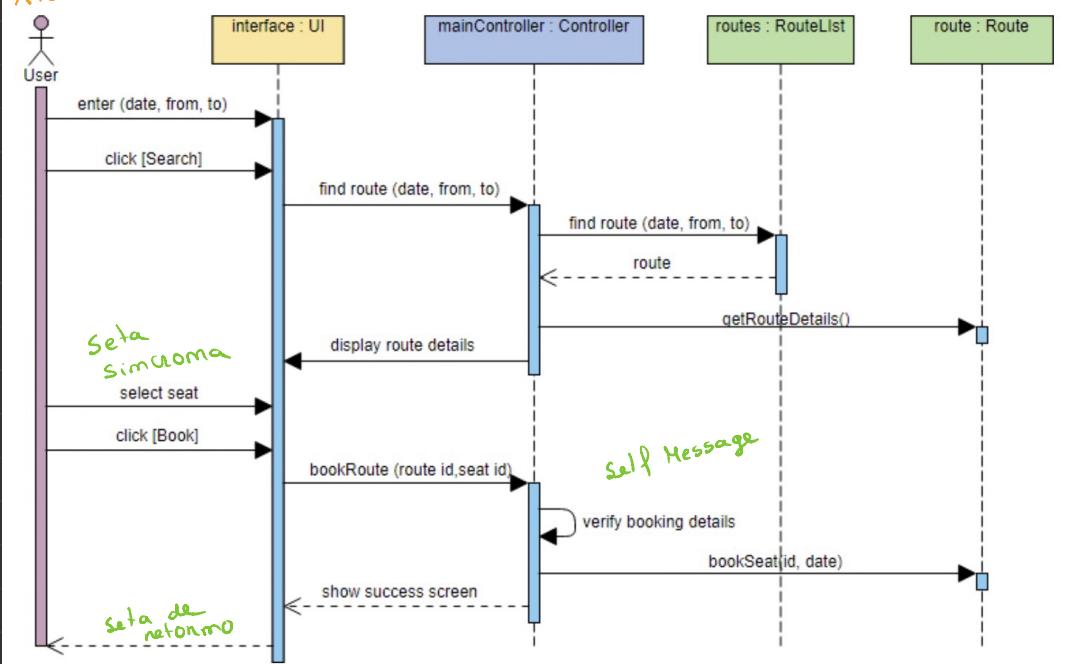


Diagramas de sequência → São diagramas que mostram em detalhe como as operações são desenvolvidas

# When to Draw Sequence Diagram?

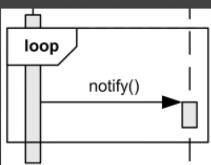
1. Model high-level interaction between active objects in a system
  2. Model the interaction between object instances within a collaboration that realizes a use case
  3. Model the interaction between objects within a collaboration that realizes an operation
  4. Either model generic interactions (showing all possible paths through the interaction) or specific instances of a interaction (showing just one path through the interaction)

## Aton

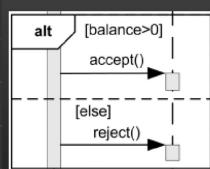
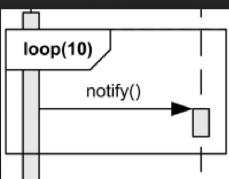


Diagramas de sequência podem também ter loops e if's

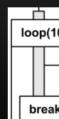
### Fragments para mostrar loops



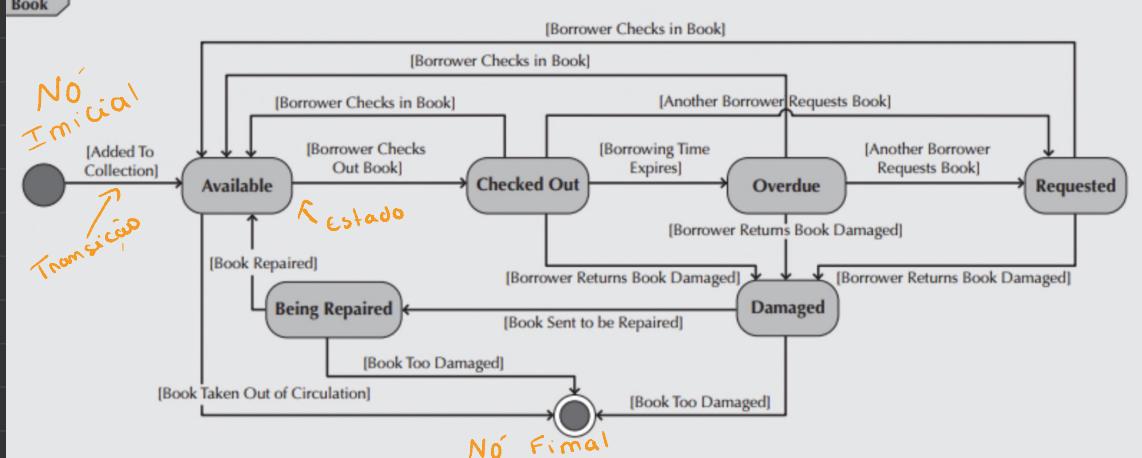
```
i=0;
While( i<10 )
{
    otherObj.Add();
    If ( y>0 ) break;
    i++
}
```



```
If ( balance>0 )
    otherObj.Accept()
Else
    otherObj.Reject()
```



- **Diagrama de Estado** → Mostra diferentes estados de uma entidade. Podem também mostrar como uma entidade responde a vários eventos mudando de um estado para o outro



## Modelos de análise

### Práticas de engenharia de requisitos

Como falo anteriormente, nos métodos ágeis temos dois tipos de requisitos que devem ser levantados elaboração

#### Requisitos funcionais

Refere-se a um processo computacional ou tratamento de dados

- O sistema deve permitir pesquisar os pacientes pelo nome

#### Requisitos não-funcionais:

Diz respeito a uma qualidade do sistema

- As pesquisas de utentes por nome, devem retornar os resultados em < 1 seg.

O conceção do desenvolvimento de requisitos é a elicitação, o processo de identificação das necessidades e restrições de um si  
Elaboração é o processo colaborativo de coletivizar, descobrir, extrair e definir requisitos

Podem, porém, existir requisitos bem e mal formulados, sendo que, para os distinguir podemos usar os critérios SMART

Specific → Target a specific area for improvement

Measurable → Quantify or at least suggest an indicator of progress

Assignable → Specify who will do it

Realistic → State what results can realistically be achieved

Time-related → Specify when the results can be achieved

Exemplos de maus requisitos:

- "O sistema deve ser fácil de usar"
- "O sistema deve estar disponível 24x7"

↓  
Enquanto se escrever "O sistema tem de existir"

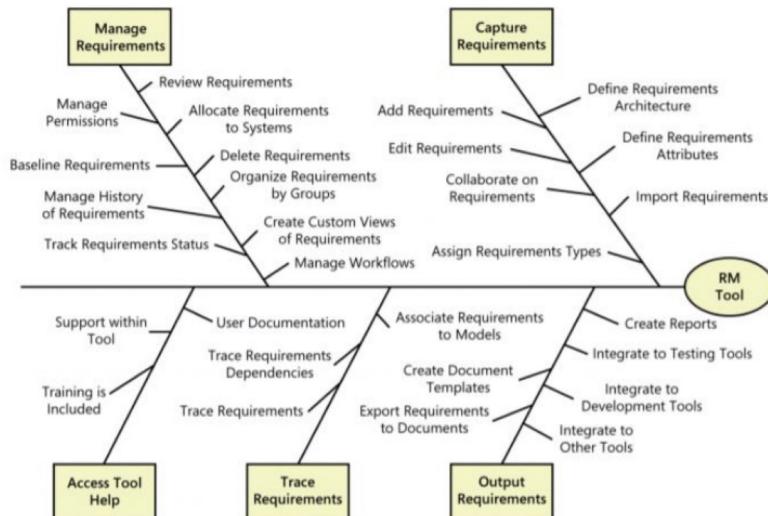


FIGURE 30-1 Common RM tool features.

## Modelação funcional com casos de utilização

Os Cau são modos importantes de organizar informação, sendo que devem ser identificados pela seguinte ordem:

1. Identificam a fronteira do Sistema
2. Identificam os atores que, de alguma forma, interagem com o Sistema
3. Identificam os objetivos / motivações de cada ator
4. Definir os Cau que satisfazem os objetivos dos atores

Estes são usualmente usados pois:

- Fornecem o contexto para a descoberta
- Um modelo de um Cau são todas as maneiras úteis de usar um SI. Tornando a aprendizagem do mesmo mais rápida

Ivan Jacobson propôs os seis Princípios para a adoração dos Cau

### 1. Mantém a simplicidade contando histórias

Para compreender um caso de uso, contamos histórias. Estas cobrem como alcançar o objetivo e como lidar com os problemas que ocorrem

### 2. Entender a grande figura

É uma maneira simples de apresentar a visão geral dos requisitos de um SI

### 3. Foco no valor

É sempre mantido o foco no valor que o SI proporcionará aos seus utilizadores e stakeholders

### 4. Constrói o sistema em fatias

O SI é construído em várias "peças" ao longo de várias iterações

### 5. Entregar o sistema em incrementos

O SI evolui ao longo de várias entregas, sendo que cada incremento fornece uma versão demonstrável ou utilitária do Sistema. É assim que todos os sistemas devem ser produzidos

## 6. Adaptação para atender as necessidades da equipa

Equipas diferentes e situações diferentes exigem estilos diferentes e diferentes níveis de detalhe

### Modelação estrutural

Num conceito de análise de sistemas podemos efetuar dois tipos de decomposição

#### Decomposição algorítmica

Destaca a ordenação de eventos

#### Decomposição orientada a objetos

Destaca os agentes que provocam a ação ou que são assuntos em que essas operações atuam

Ao projetar um sistema de software complexo, é essencial decompô-lo em partes mais pequenas, cada uma delas podendo ser então refinada de forma independente. Em vez de tentar compreender o sistema como um todo, precisamos apenas de compreender algumas partes.

Para isso usamos diagramas de classe e diagramas de objetos

Diagramas de classe → Mostram em que consistem os objetos do sistema

Diagramas de Objetos → Mostram como os objetos interagem uns com os outros

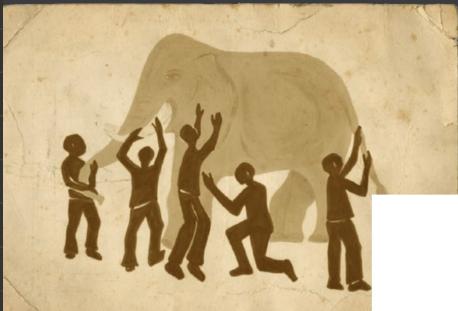
### Modelação de comportamento

Os diagramas comportamentais UML visualizam, especificam, construem e documentam os aspectos dinâmicos de um sistema, fazendo a representação dos detalhes de um processo

#### Exemplos:

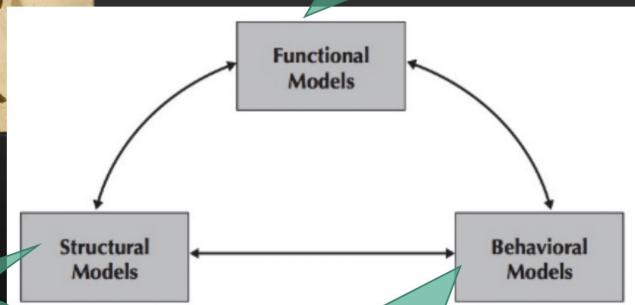
- Diagramas de sequência

## Resumo dos Modelos :



...

O que é que o Sistema deve fazer?  
[do ponto de vista do observador  
externo;  
caixa fechada]



Quais são as partes/"coisas"  
que constituintes e o  
relacionamento estrutural  
entre elas ?

I Oliveira

Como é que as parte  
colaboram/interagem ao  
longo do tempo? [perspetiva  
dinâmica]

4

## Modelos no desenho e implementações

### Vistas de arquitetura

Uma arquitetura é o conjunto de decisões significativas em relação à organização de um sistema de software

Consiste na seleção dos elementos estruturais e as suas interfaces pelas quais o sistema é composto, juntamente com o seu comportamento

Esta arquitetura é constituída por dois elementos abstratos

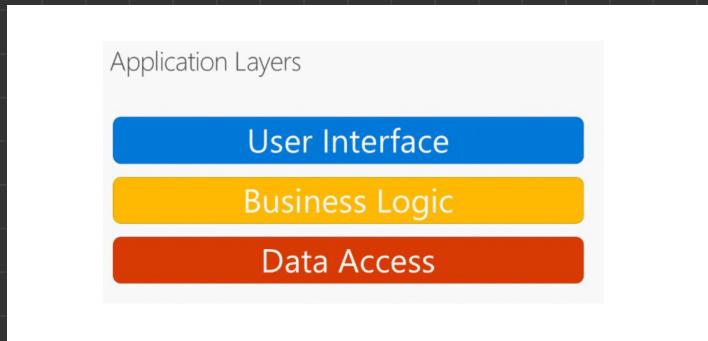
modes | class  
module  
package  
layer  
Sub system  
System

interconnections

communications  
control

Um dos conceitos principais da arquitetura é a organização do SI por camadas, sendo que:

- São sobrepostas
- São especializadas
- Não se pode sair das camadas



Os utilizadores fazem solicitações através da UI, que interage com a BL e, esta, por sua vez, se precisar interage com a DA

## Classes e desenhos de métodos

As classes estão interligadas umas com as outras, sendo que podemos classificar o seu tipo de ligação em:

Coupling → Mede a força da dependência de uma classe a outra. A classe  $C_1$  está acoplada a  $C_2$  se precisa de  $C_2$

Coesão → Mede a força do relacionamento dos elementos de uma classe entre si.

Código Java  $\Rightarrow$  Diagramas de Classes / Diagramas de Sequência

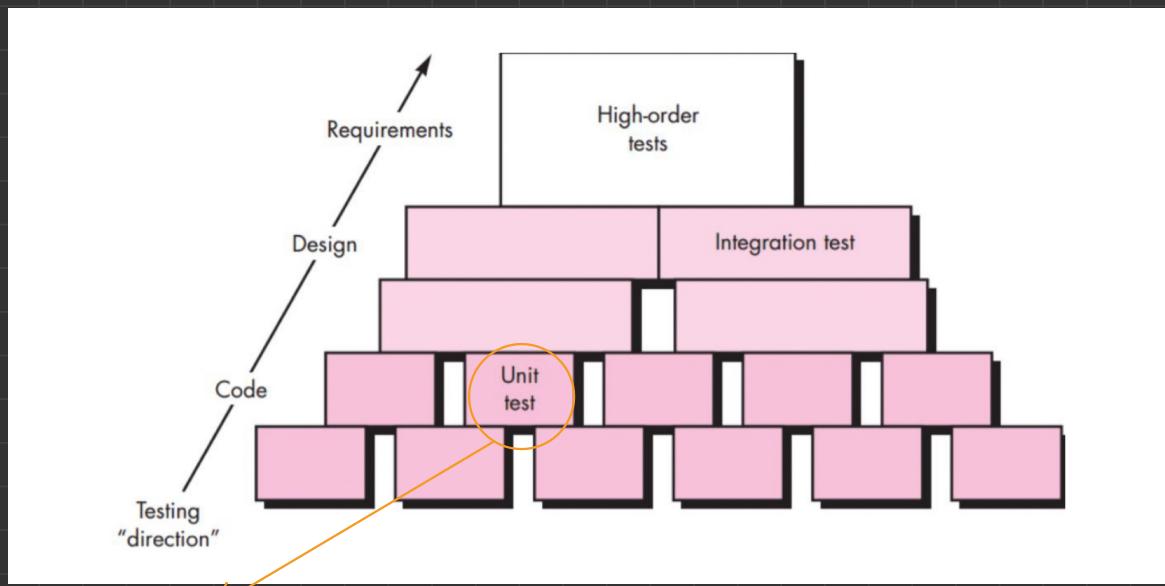
P  
Práticas Selecionadas na Construção de Software

## Garantia e gestão de qualidade

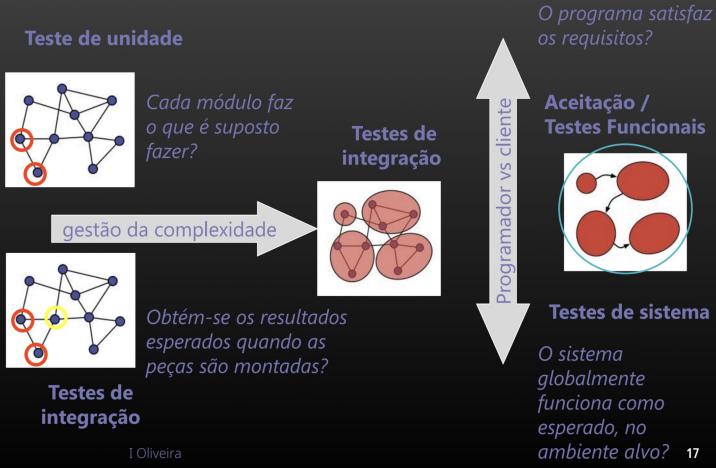
A garantia da qualidade ao longo do desenvolvimento do projeto é algo que deve ser da preocupação de qualquer engenheiro para isso deve

- Verificam os produtos de trabalho contra as suas especificações
- Verificam a consistência dos módulos
- Verificam através das melhores práticas da indústria
- Verificam se o resultado encontra-se às expectativas do utilizador e stakeholders

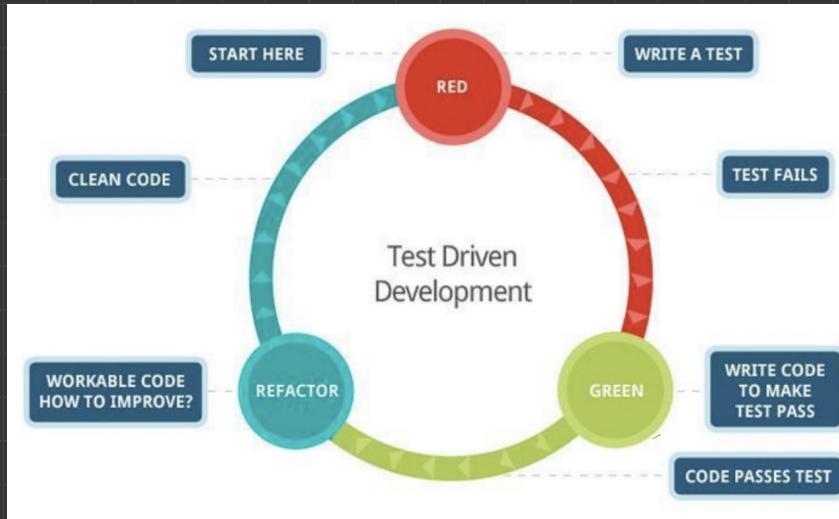
No entanto, os testes não podem ser realizados de qualquer maneira e, por isso devem seguir a seguinte estrutura



Os testes devem ser realizados em módulos



Com base no conceito de testes constantes surgiu o  
Test Driven Development



Conceitos :

- Build, then fix
- Small tests
- Automation

Com base neste conceito surgiu um novo termo: "Debug - later"

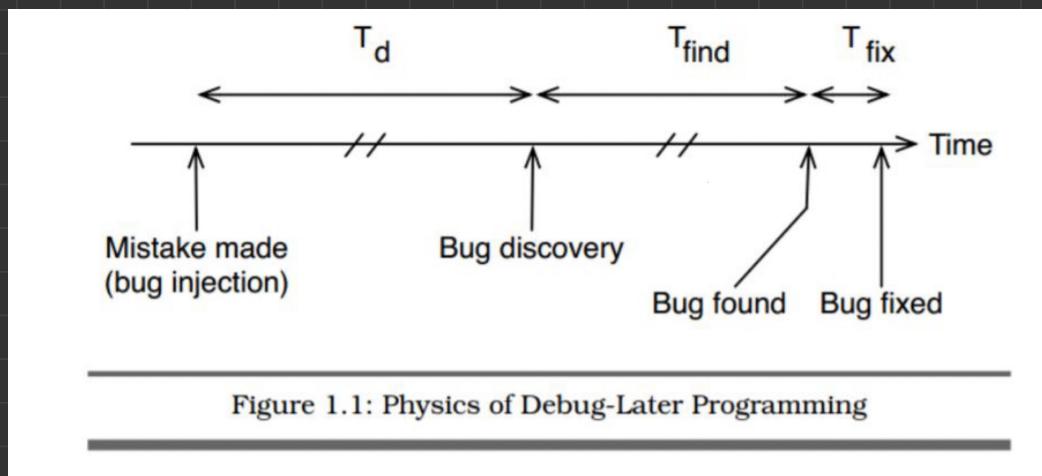
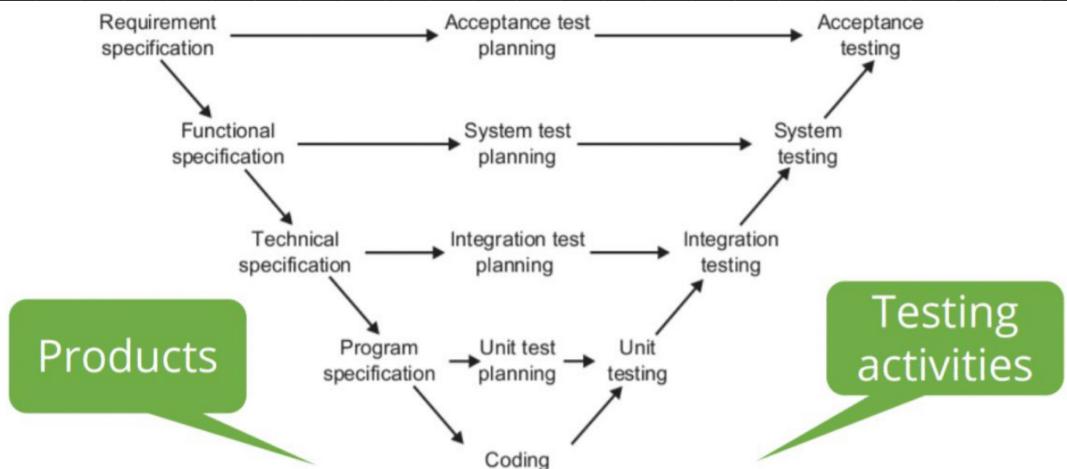


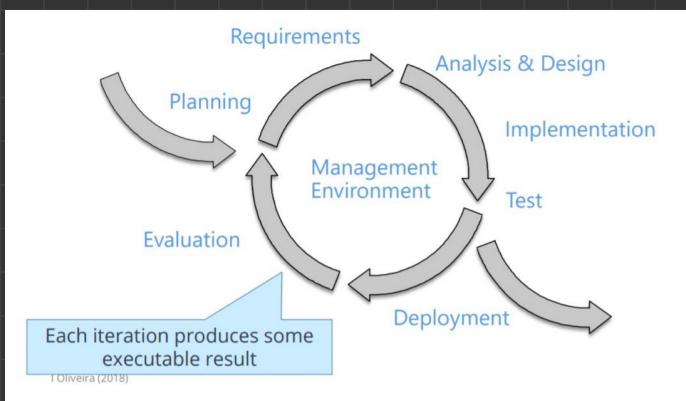
Figure 1.1: Physics of Debug-Later Programming

Agora é jogar com a lógica. Se  $T_d$  for maior,  $T_{find}$  também será geralmente maior, uma vez que, quando  $T_d$  é baixo significa que o programador cometeu o erro há pouco tempo e, por isso, consegue localizá-lo mais rapidamente



Products

Testing activities



Cada iteração fornece algum software funcional que foi totalmente testado. A equipa entrega um conjunto de histórias e estas são testadas antes da entrega.

A história não está completa até que passe em todos os testes e seja aceite pelo cliente.

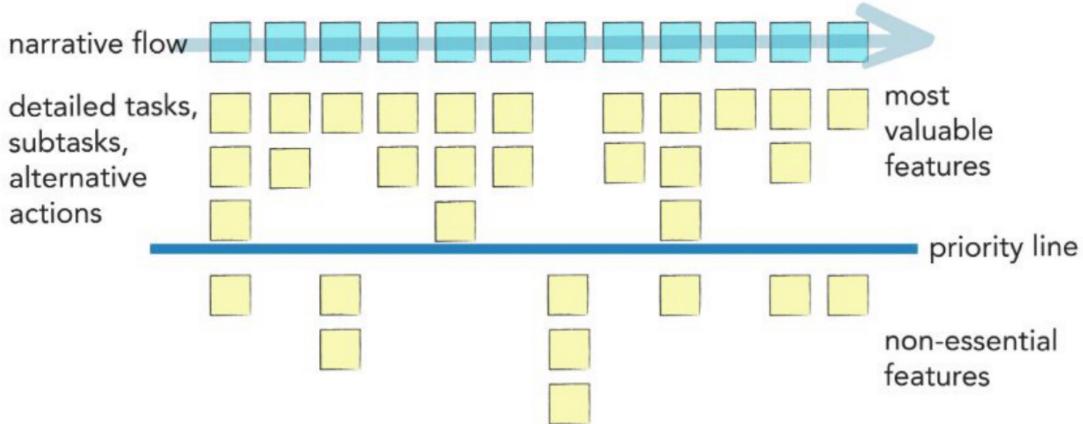
## Abordagens Complementares

User Stories e métodos ágeis

Uma user story é uma anotação que captura o que um utilizador faz ou precisa de fazer como parte do seu trabalho.

Ao contrário da enumeração tradicional de requisitos as stories concentram-se nas necessidades do utilizador.

# User Story Map



- O utilizador cria uma conta na plataforma da Arrive

Sendo o Utilizador um visitante do site da Arrive,  
Quero criar uma conta na Plataforma

De modo a poder comprar um Pacote de Viagem

Exemplo de  
uma user  
Story

## Cenário 1: Credenciais criadas com sucesso

Dado que estou na página de login da Arrive

E ainda não tenho uma conta criada na plataforma

Quando selecionei o botão "Não tem conta?" Registe-se"

Então insiro os dados pessoais necessários para criar uma conta.

## Cenário 2: Não foi possível criar a conta

Dado que estou na página de login da Arrive

E ainda não tenho uma conta criada na plataforma

Quando selecionei o botão "Não tem conta?" Registe-se"

Então insiro mal os dados pessoais e o meu email pessoal necessário para criar uma conta

Então a plataforma avisa-me que não foi possível criar a minha conta

As user stories permitem  
simular a preparação de  
um si para um certo  
tipo de pessoas

Estas têm sempre de falar  
ma primaína pessoa. como  
se fossemos o utilizador final

As user stories embora tenham variações semelhanças com os  
casos de utilização são bastante diferentes

Semelhanças: - Contêm tipos semelhantes de informação

Diferenças: - Os user stories são mais detalhados  
que os CaU

- As user stories são destinadas a provocar  
conversas ao fazermos - se perguntas durante  
reuniões de Scrum, em vez de ter uma  
especificação de requisitos antecipada  
mais detalhada, como nos CaU

Uma das principais diferenças é a de **Persona vs Actor**

As personas são utilizadas em U.S. e são geralmente mais robustas que os atores. Uma persona é projetada para ajudar a equipa a compreender o destinatário do sistema.

Os atores são utilizados em CaU e são usados como uma ferramenta para desenvolver requisitos. São projetados para ajudar a validar projetos e como ferramenta para conduzir atividades de teste.

**Pontuação →** As User Stories podem também ser pontuadas com base na dificuldade da tarefa, como fizemos no backlog do projeto.

No backlog do projeto, são então inseridas as user stories com os seus diferentes pontos de dificuldade. Estas serão depois organizadas automaticamente pelo Tracer que, por sua vez calcula a velocidade média no projeto, ou seja a velocidade com que as tasks de diferentes graus de dificuldade são resolvidas.

## Metodologia Scrum

O Scrum é uma maneira de uma equipa se organizar, definir tarefas e partilhar ideias.

A Daily Scrum meeting não é usada como solução de problemas ou reunião de resolução dos mesmos, esses são levantados off-line e tratados depois da reunião. Durante esta reunião a equipa responde às seguintes perguntas:

- O que foi feito ontem?
- O que vai ser feito hoje?
- Há algum obstáculo?

Sprint → Tempo que a equipa demora a completar uma tarefa

É de notar que todas sprints são iterações, mas nem todas as iterações são sprints