TRANS_
FORMAR
O PAÍS PELA
EDUCAÇÃO
É O QUE
NOS MOVE

ecossistema
ănima

Modelos, Métodos e Técnicas da Engenharia de Software

Aula 06 e

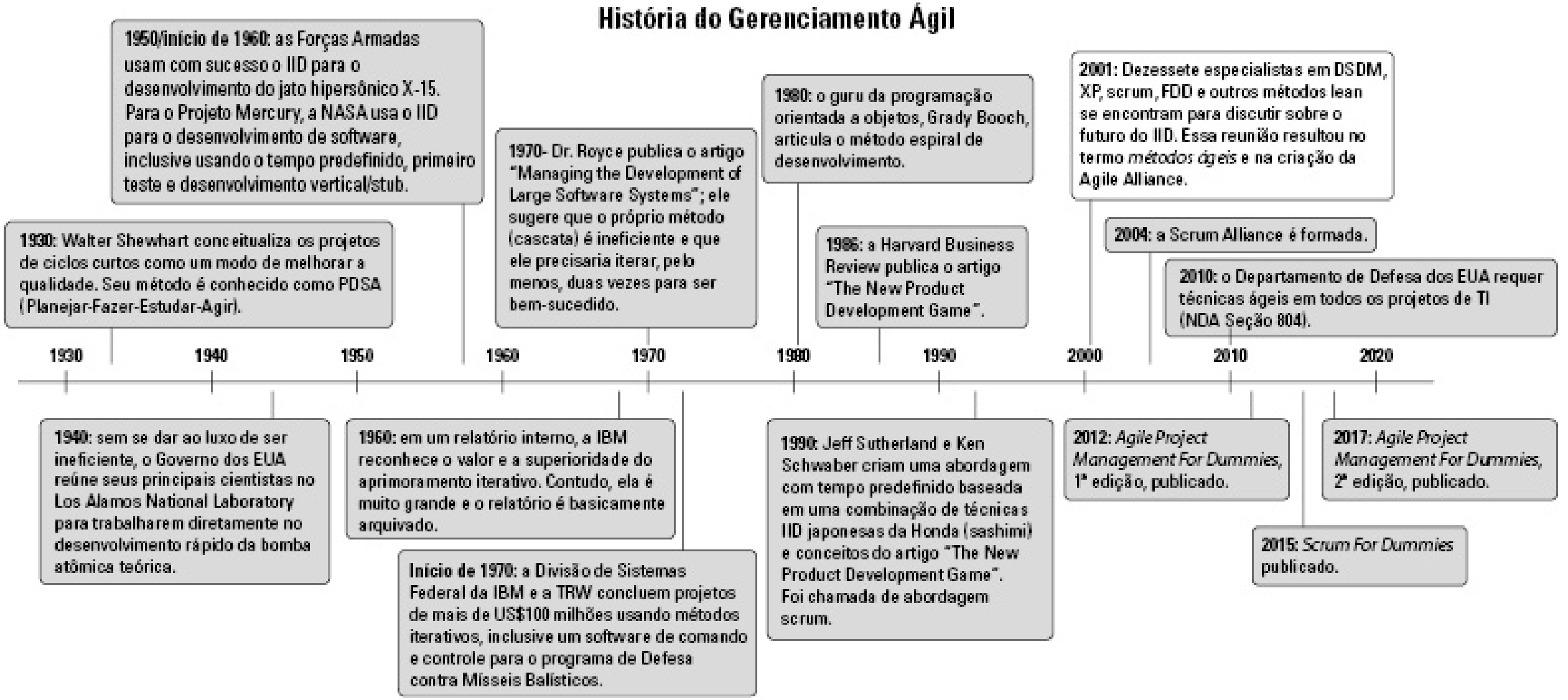
Levantamento ágil de requisitos, incluindo histórias de usuários.

Aula 07

MVPs e testes A/B. (Templates)

Prof. Luis Ybarra

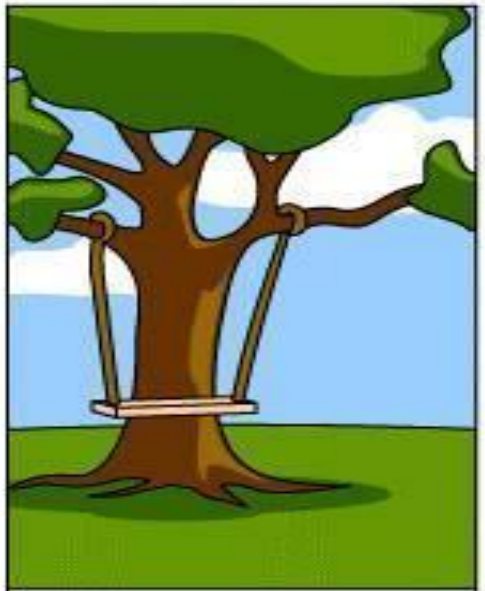
Apresentando o Gerenciamento Ágil de Projetos



Levantamento ágil de requisitos

A elicitação de requisitos é a **primeira atividade no processo de engenharia de requisitos**, na qual se busca entender quais são as necessidades do usuário que devem ser atendidas pelo software que será desenvolvido (Sommerville e Kotonya, 1998).

Como o cliente explicou...

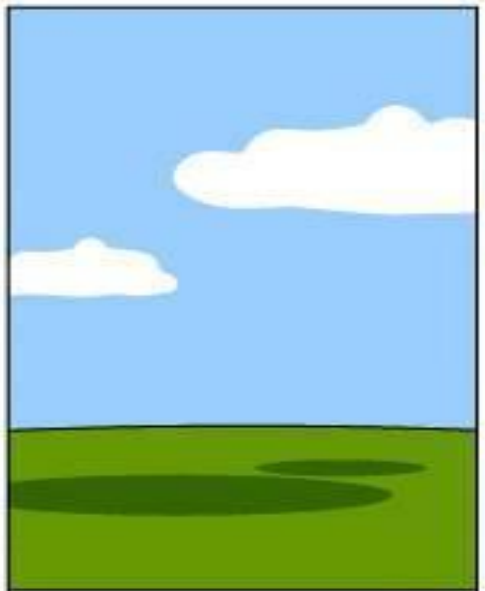


Como o líder de projeto entendeu...

Como o analista projetou...

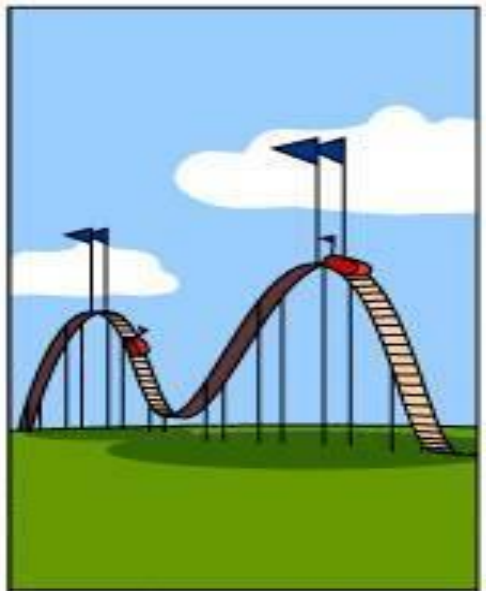
Como o programador construiu...

Como o Consultor de Negócios descreveu...

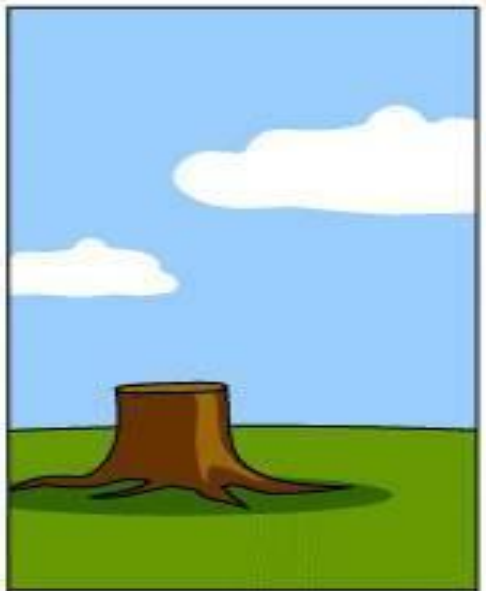


Como o projeto foi documentado...

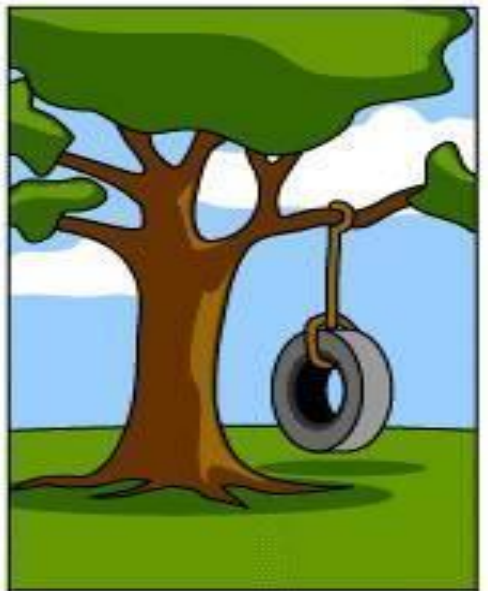
Que funcionalidades foram instaladas...



Como o cliente foi cobrado...

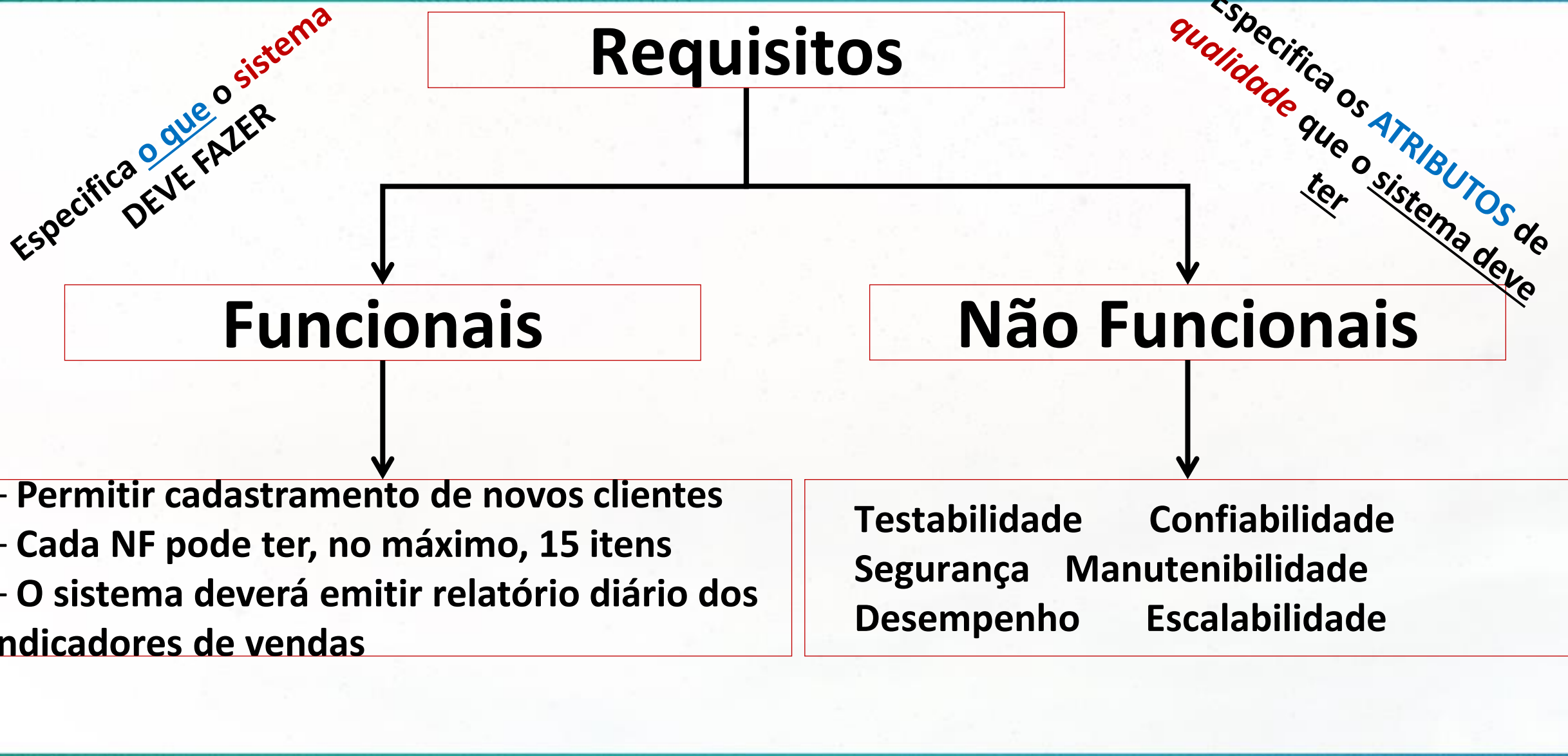


Como foi mantido...



O que o cliente realmente queria...

Elicitação de Requisitos



Engenharia de Requisitos

FONTES DE REQUISITOS:

→ Stakeholders
→ Documentos
→ Sistemas em Operação

FATORES DE SATISFAÇÃO

→ Básicos
→ Esperados
→ Inesperados

TÉCNICAS PARA COLETA DE INFORMAÇÕES

→ Entrevistas
→ Focus Groups (discussões em grupo)
→ Questionários
→ Shadowing (aprendizagem por observação)
→ Instrução dos Usuários
→ Prototipação

Engenharia de Requisitos

- Uma **ENTREVISTA** é uma reunião individual entre um membro da equipe do projeto e um usuário ou um *stakeholder*
- ***FOCUS GROUP*** é uma discussão objetiva, conduzida ou moderada que introduz um tópico a um grupo de respondentes e direciona sua discussão sobre o tema, de uma maneira não-estruturada e natural
- **QUESTIONÁRIOS** consistem em conjuntos de perguntas que são criados para reunir informações
- ***SHADOWING*** *é uma técnica em que você observa um* usuário realizar as tarefas no seu ambiente de trabalho, e pergunta ao usuário qualquer questão relativa às tarefas
- **INSTRUÇÃO DOS USUÁRIOS**: Quando você usar a técnica de instrução dos usuários, essas pessoas realmente irão treiná-lo nas tarefas que realizam
- Um **PROTÓTIPO** é uma versão inicial, em geral parcial, de um produto (ou parte do produto) do projeto

O que é Gerenciamento de Requisitos Ágil?

O gerenciamento de requisitos é o processo de identificar, documentar e gerenciar os requisitos de um projeto.
No desenvolvimento em cascata tradicional, esse processo é muito linear.
Os analistas de negócios trabalham com as partes interessadas para reunir todas as informações necessárias e, em seguida, transmiti-las aos desenvolvedores que começam a codificar.
Esse processo pode consumir muito tempo e geralmente leva a prazos perdidos e clientes insatisfeitos.

No desenvolvimento ágil, os requisitos são gerenciados de uma maneira muito diferente.

A filosofia ágil é baseada em quatro princípios fundamentais:

1. indivíduos e interações acima de processos e ferramentas;
2. software que trabalha sobre uma documentação completa;
3. colaboração do cliente sobre a negociação do contrato;
4. e responder às mudanças ao invés de seguir um plano.

O gerenciamento de requisitos ágil reflete esses princípios por ser rápido, flexível e focado na entrega de valor ao cliente.

ă Como funciona o gerenciamento de requisitos ágil?

O gerenciamento de requisitos ágil é uma abordagem iterativa e incremental ao gerenciamento de requisitos que se caracteriza por sua flexibilidade e capacidade de resposta à mudança.

No desenvolvimento tradicional em cascata, os requisitos de um projeto são reunidos de uma só vez no início do projeto e, em seguida, repassados aos desenvolvedores que iniciam a codificação.

Isso pode levar a problemas porque é muito difícil prever todas as mudanças que ocorrerão ao longo de um projeto longo.

O gerenciamento ágil de requisitos, por outro lado, funciona em ciclos curtos chamados sprints.

No início de cada sprint, a equipe identifica em quais requisitos eles trabalharão durante esse sprint.
À medida que o sprint avança, eles podem descobrir que alguns dos requisitos mudaram ou que novos requisitos surgiram.
A equipe pode então adaptar seus planos de acordo.
Essa flexibilidade torna o gerenciamento ágil de requisitos muito mais responsivo a mudanças e ajuda a evitar os problemas que podem ocorrer no desenvolvimento tradicional em cascata.

ă

ă INTRODUÇÃO - Histórias de Usuário (User Stories)

O Modelo Ágil surge com a proposta de reduzir essa documentação. O foco é voltado para a colaboração e comunicação entre as partes envolvidas, buscando entregar valor de forma contínua e responder rápido às mudanças que surgirem no meio do caminho.

Ao invés de uma documentação extensa e exaustiva, podemos lançar mão de algumas ferramentas mais sucintas e que cumprem muito bem o objetivo quando bem aplicadas.

Uma delas é a escrita de histórias de usuário.

Histórias de Usuário (User Stories)

As Histórias de Usuário comumente se baseiam no template:

COMO UM <USUARIO>

DESEJO <NECESSIDADE>

PARA QUE <OBJETIVO>

Não é a única ferramenta que podemos usar para escrever requisitos ágeis, mas as Histórias de Usuário são muito interessantes por gerar alinhamento entre os clientes e o time de desenvolvimento.

Elas fazem com que todos conversem na mesma linguagem e a necessidade possa ser transmitida com clareza.
Para isso, devemos lembrar sempre que as histórias precisam ser criadas seguindo o conceito INVEST, ou seja:

- **Independent (Independente):** pode ser implementada em qualquer ordem
- **Negotiable (Negociável):** pode ser removida a qualquer instante se descobrir que não é útil
- **Valuable (Valorosa):** entregar valor
- **Estimable (Estimável):** capaz de ser estimada
- **Small (Pequena):** caber em uma Sprint (duas a quatro semanas)
- **Testable (Testável):** possível de ser validada

Ainda que sejam interessantes, claras aos usuários e sucintas o suficiente para que todos consigam rapidamente ler e entender do que se tratam, as histórias geralmente não fornecem subsídios suficientes para que o time de desenvolvimento possa de fato codificar e desenvolver a solução.

Neste ponto surgem os Critérios de Aceitação, que são uma lista de itens que expressam como o comportamento do negócio é validado, maximizando o entendimento do mesmo.

Determinando os requisitos do produto e criando histórias do usuário

Depois de ter identificado os diferentes clientes, comece a determinar os requisitos do produto e criar histórias dos personagens.
Uma boa maneira de criar histórias do usuário é reunir os envolvidos em uma sessão de criação da história. (Layton, 2019).

Faça com que os envolvidos escrevam o máximo de requisitos em que puderem pensar, usando o formato de história do usuário.
Uma história para o projeto e personagens das seções anteriores pode ser assim:

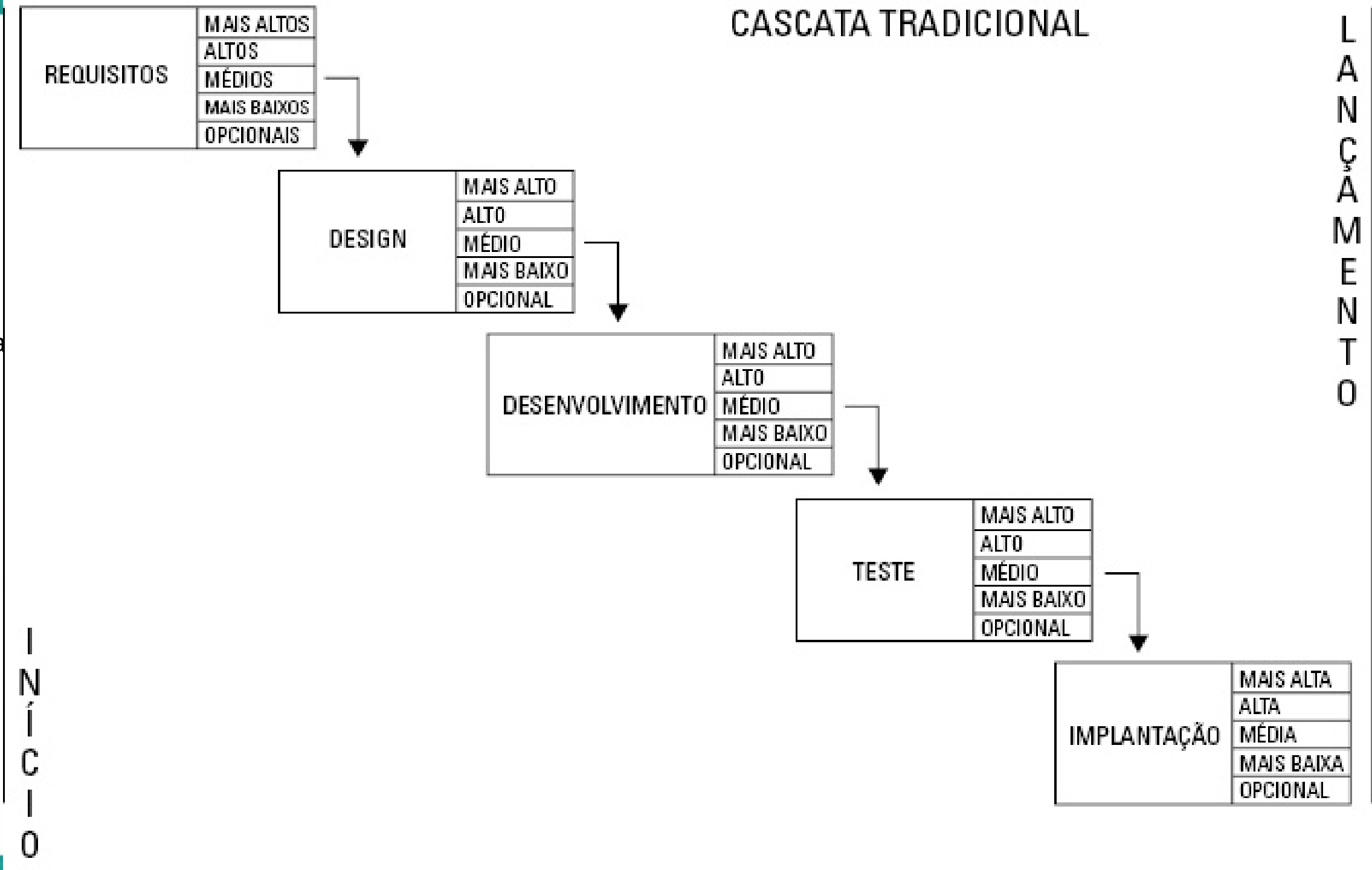
Frente da ficha:

Título Ver o saldo da conta bancária Como José, quero ver o saldo de minha conta corrente no smartphone para poder ver quanto dinheiro tenho

Verso da ficha:

Quando entro no aplicativo do Banco XYZ, o saldo da contacorrente aparece no topo da página.
Quando entro no aplicativo do Banco XYZ, depois de fazer uma compra ou depósito, o saldo da contacorrente reflete isso. (Layton, 2019).

FIGURA 3-2: O ciclo do projeto em cascata é uma metodologia linear. (Layton, 2019).



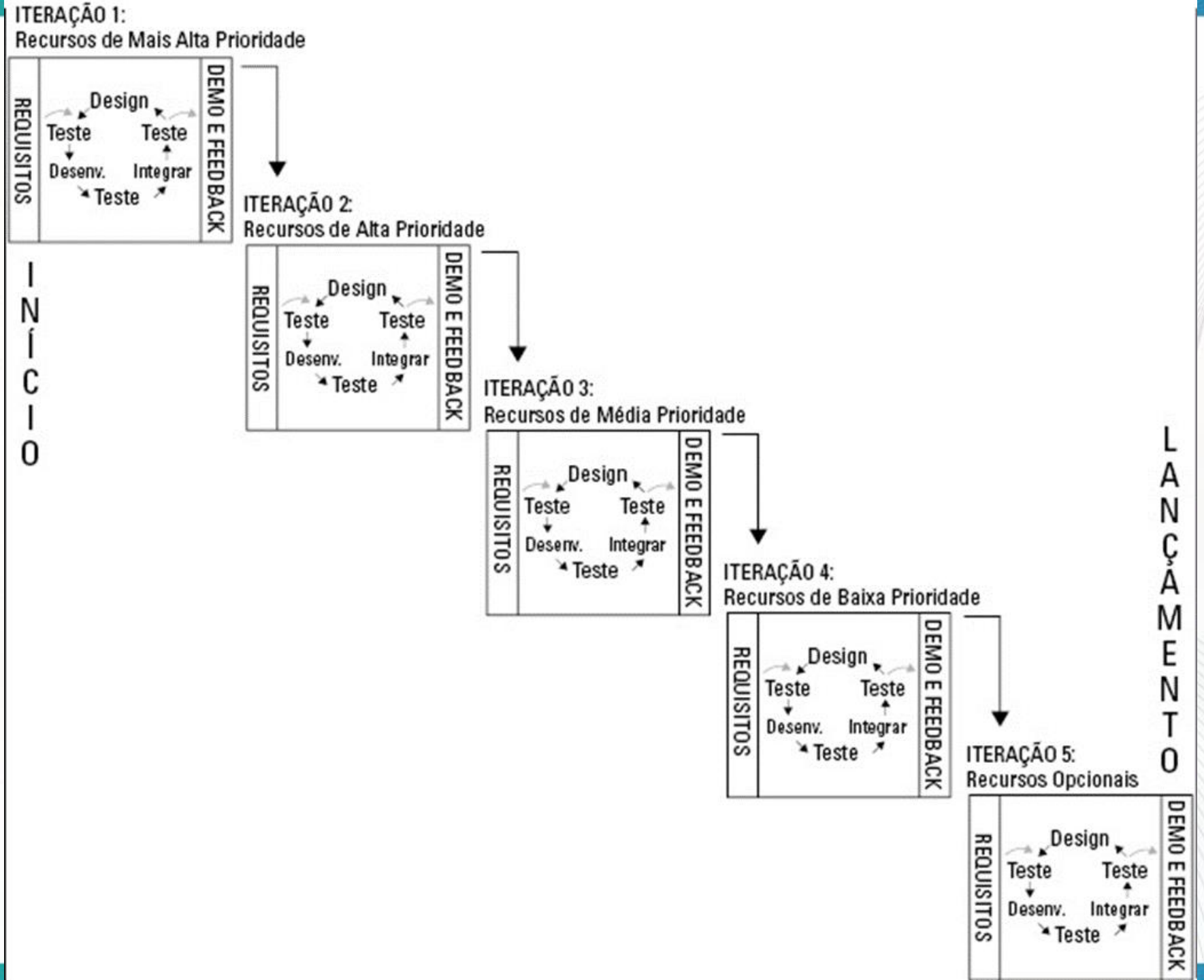


FIGURA 3-3:
As abordagens ágeis têm um ciclo de projeto iterativo. (Layton, 2019).

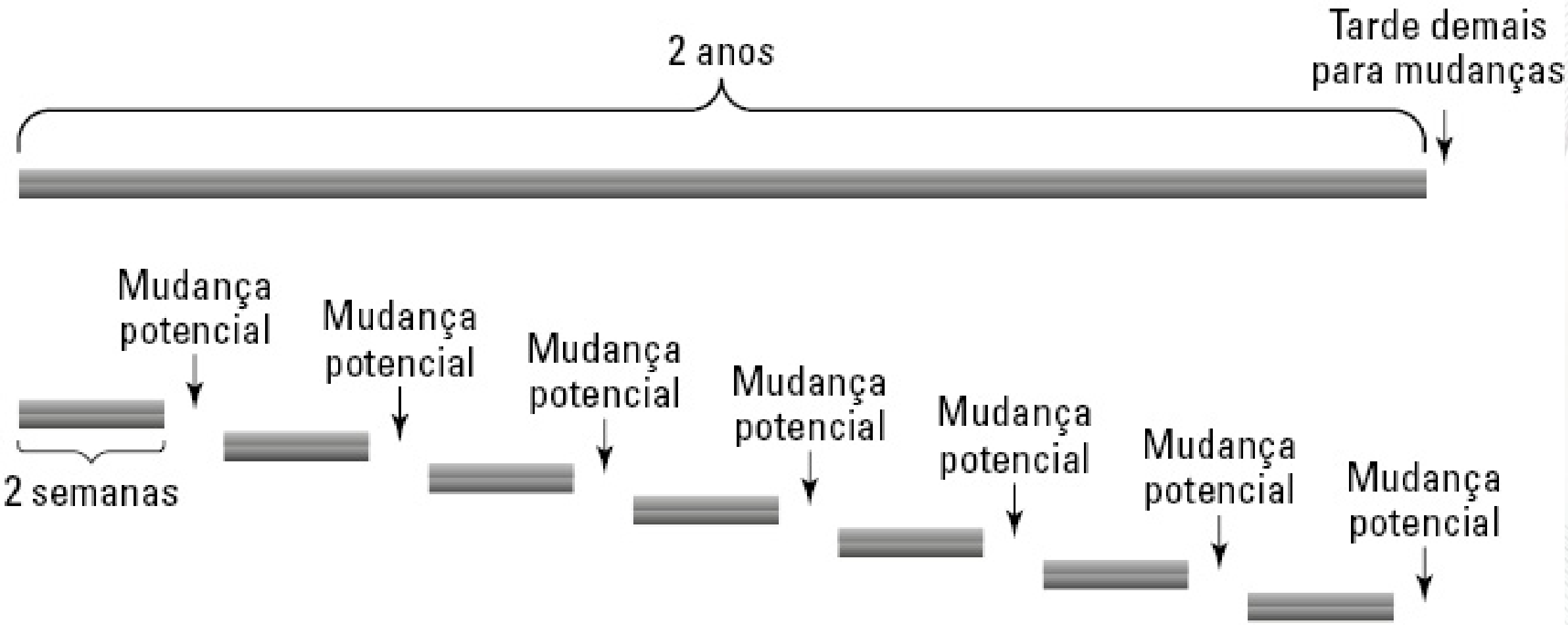
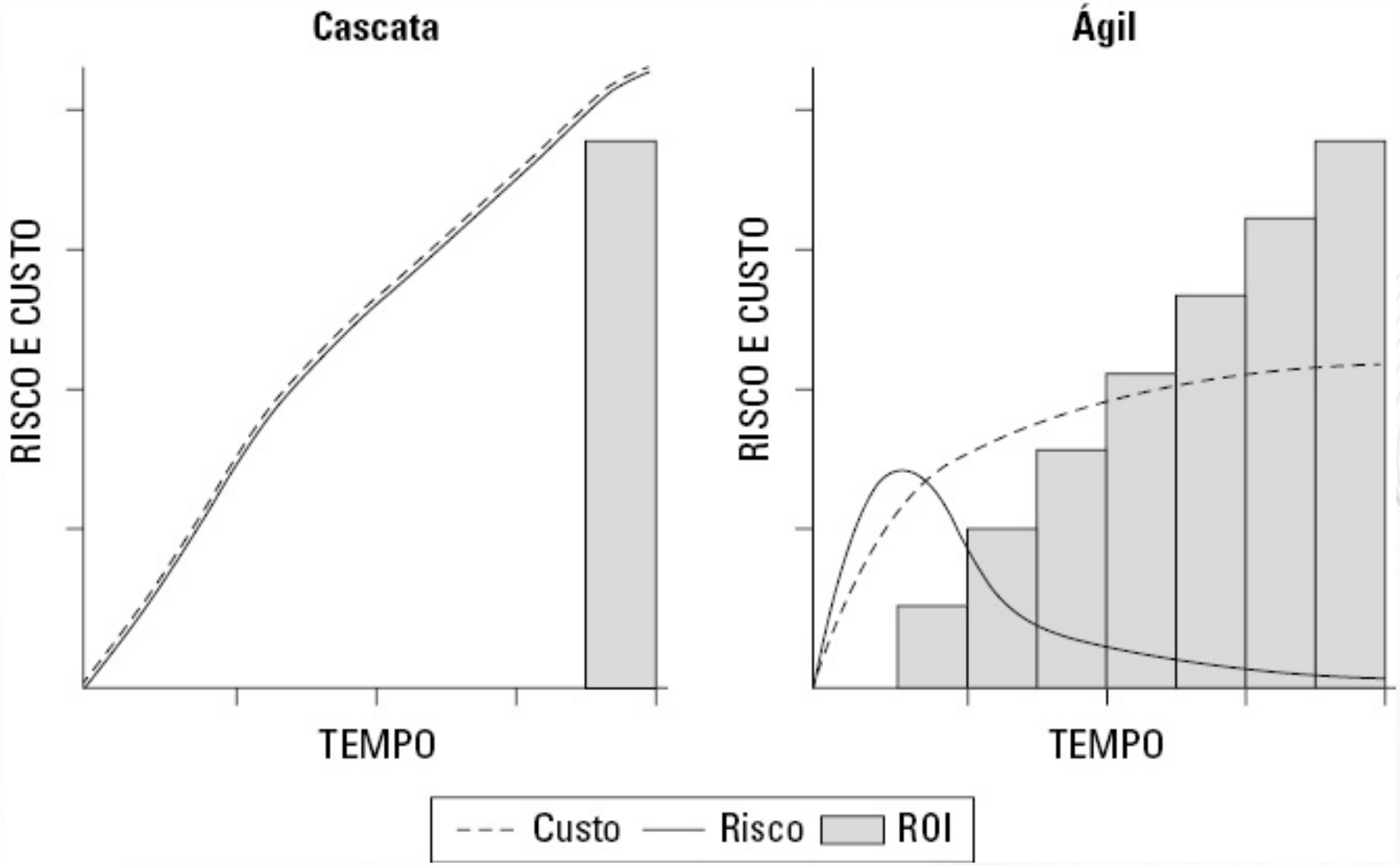


FIGURA 3-4: Estabilidade na flexibilidade nos projetos ágeis. (Layton, 2019).



Retorno no investimento (ROI)

FIGURA 3-5:
Um gráfico de risco e investimento comparando as metodologias em cascata e ágil. (Layton, 2019).

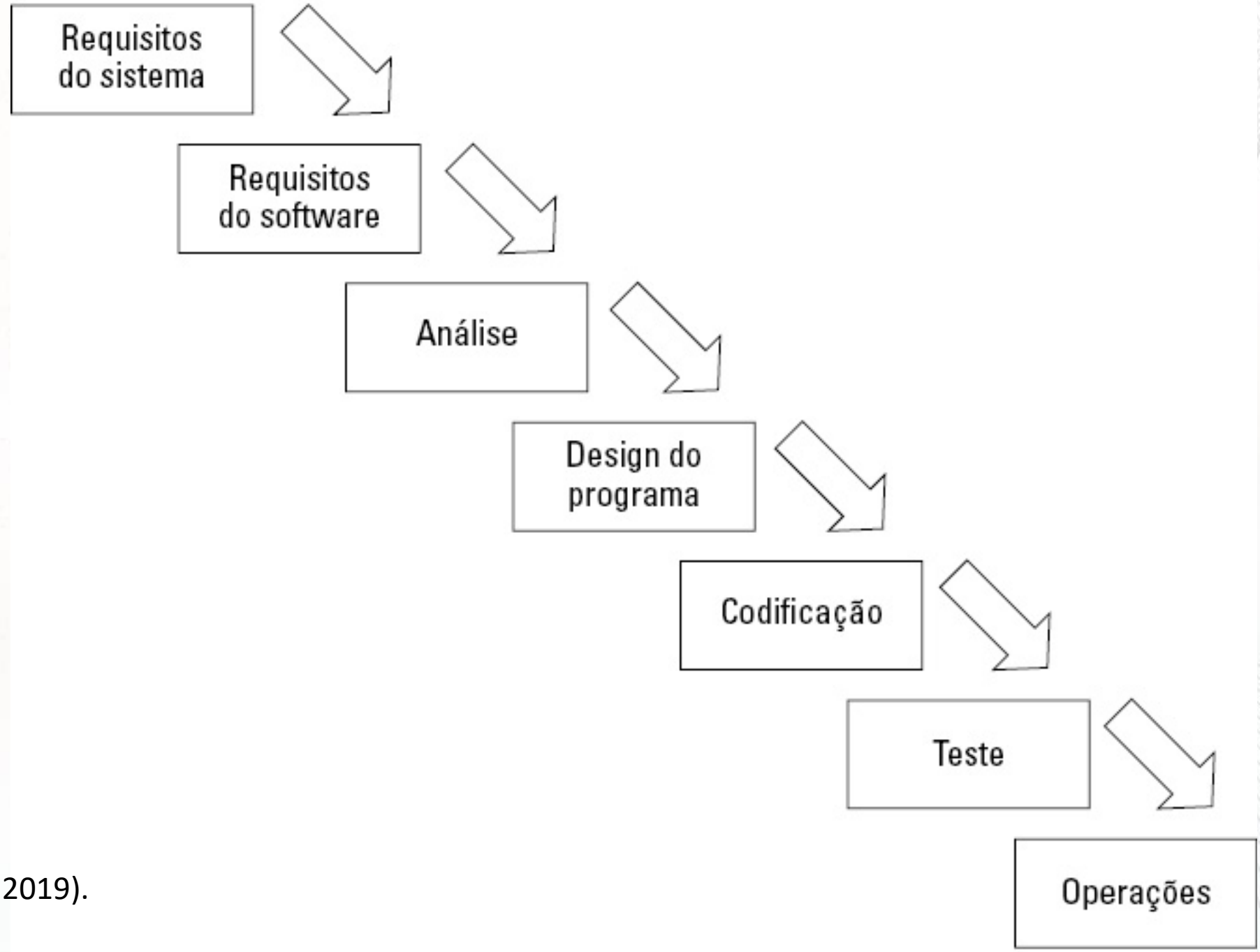


FIGURA 4-2:
As origens da cascata. (Layton, 2019).

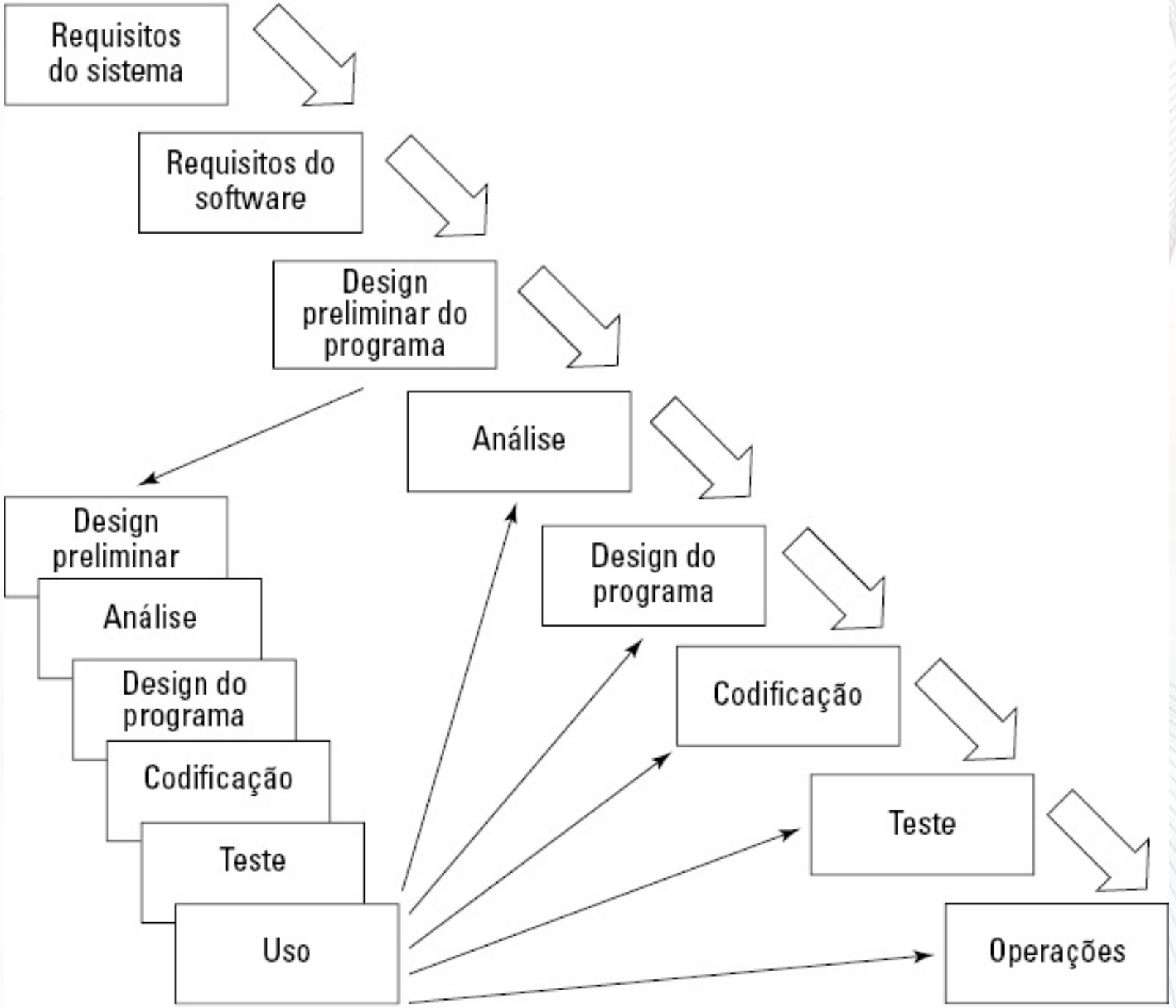


FIGURA 4-3:
Iteração em cascata. (Layton, 2019).

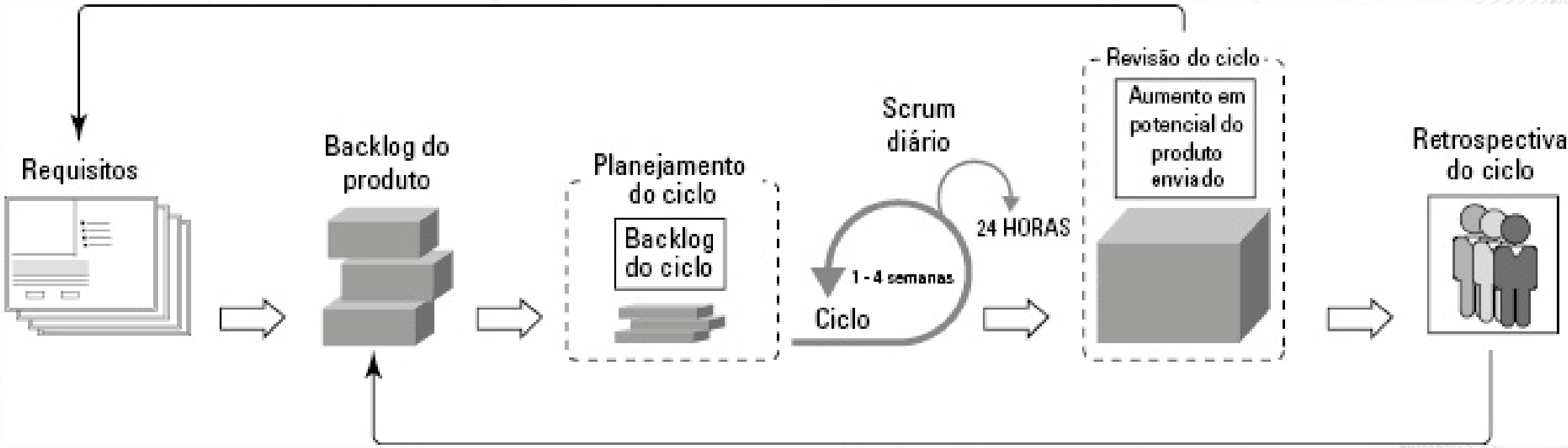


FIGURA 4-4:
Abordagem scrum. (Layton, 2019).

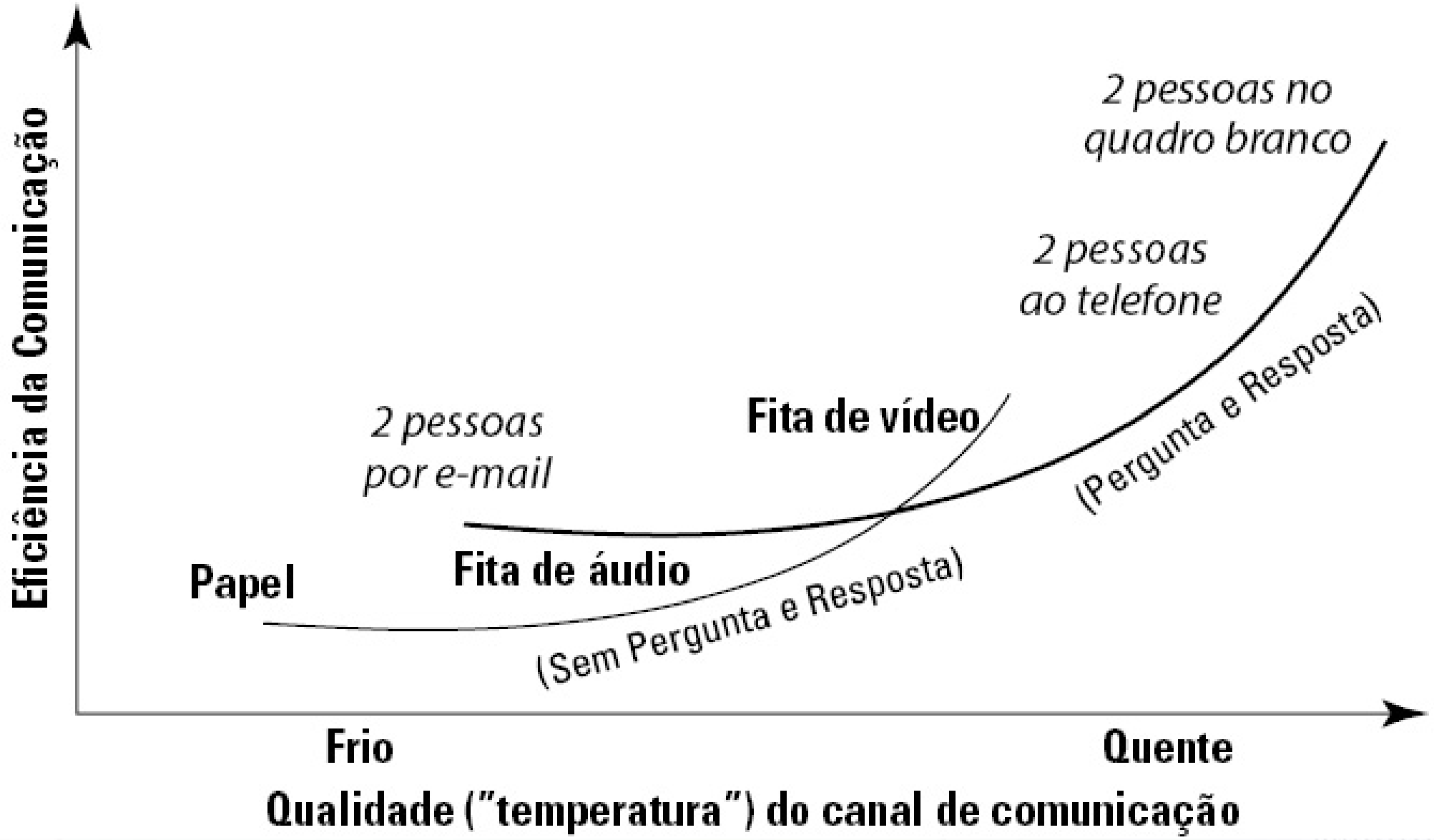


FIGURA 5-1: Melhor comunicação com o local partilhado. (Layton, 2019).

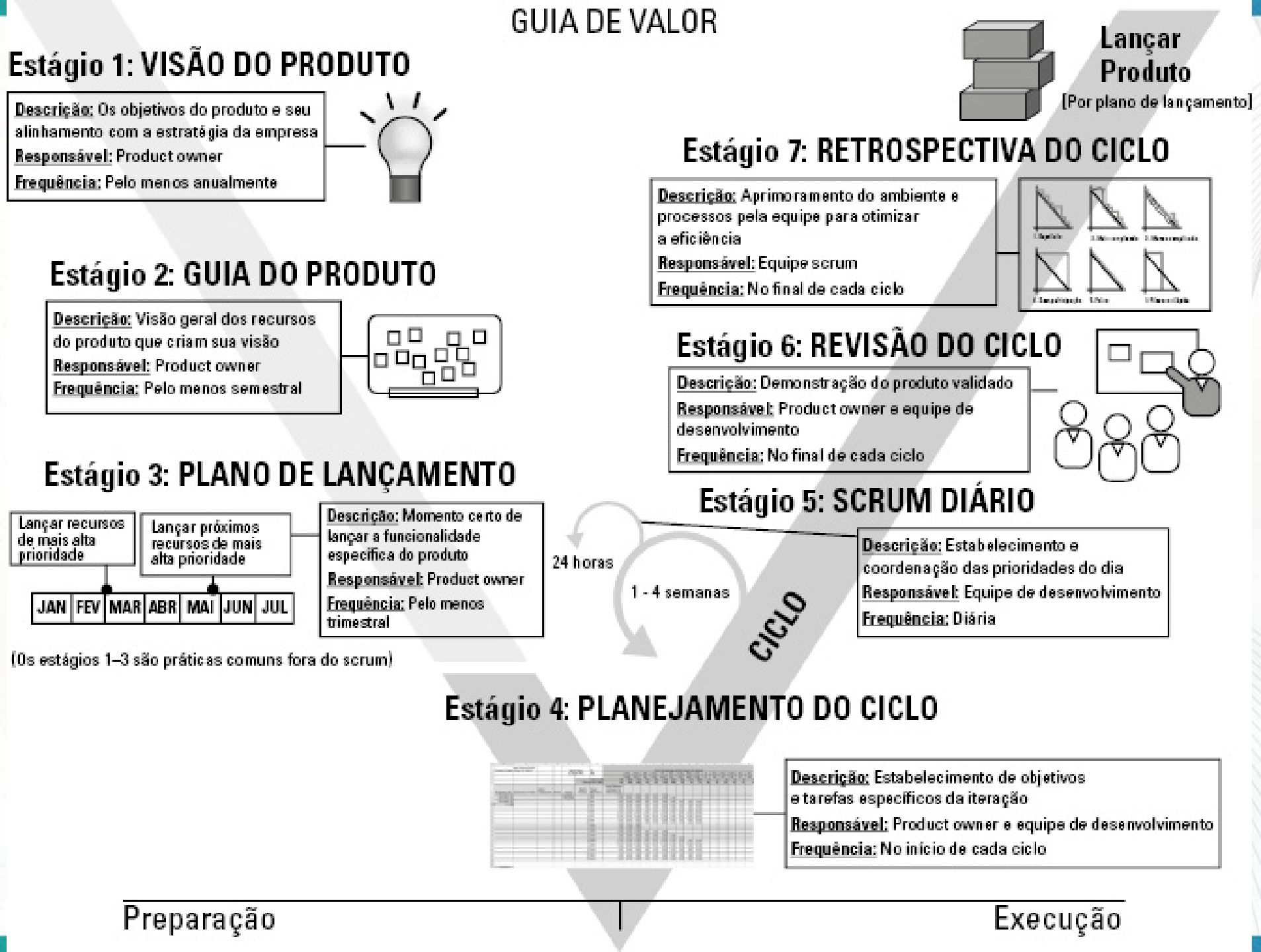


FIGURA 7-1:
Estágios do planejamento ágil e execução com o Guia de Valor. (Layton, 2019).

Modelos, Métodos e Técnicas da Engenharia de Software

Aula 07

Continuação

MVPs e testes A/B. (Templates)

Prof. Luis Ybarra

Padrões arquiteturais MVC, MVP e MVVM

Introdução

No desenvolvimento de projetos, a preocupação com a arquitetura da aplicação é fundamental.

É preciso saber definir a plataforma, apontar os componentes que serão desenvolvidos e demonstrar como serão organizados, além de seguir os requisitos do sistema e focar no desenvolvimento.

Quando se trata de arquitetura de software, o mais importante é entender os mecanismos utilizados em cada camada, seus benefícios e seus impactos. (Zenker, 2019).

Existem padrões que dão referência, servem de modelo para a aplicação, normalizam o código, criam boas práticas, comunicam e organizam os componentes do sistema.
Os padrões servem como ponto de partida para o desenvolvimento e aumentam a qualidade e a produtividade do sistema.
Desenvolvedores experientes, já conhecendo os padrões e as boas práticas, tendem a optar diretamente pelas melhores soluções para cada situação.
você vai entender a importância e os benefícios proporcionados ao utilizar os padrões model–view–controller, model–view–presenter e model–view–view-model.
Você vai analisar as diferenças entre esses padrões e vai identificar a aplicação de cada um deles

Importância e benefícios de adotar padrões MVC, MVP ou MVVM

A arquitetura define os elementos de software e como eles interagem entre si, além das suas conexões e da sua persistência.

Pode ser definida como a forma ou estrutura composta pelo projeto.

Os padrões de arquitetura e padrões de projeto (design patterns) facilitam a implementação e resolver problemas por meio de soluções conhecidas, desenvolvidas e já testadas.

Para Martin (2017), um dos principais objetivos dos padrões arquiteturais é reduzir a necessidade de recursos humanos para construir e manter um sistema.

Para utilizar padrões de arquitetura, é necessário muito conhecimento, pois, assim como no uso de padrões de projetos, são muitos os padrões disponibilizados para o desenvolvimento de software.

A escolha depende do tipo de projeto em questão, da equipe de desenvolvimento e dos recursos oferecidos pela empresa.

Entre os padrões, existem alguns que foram desenvolvidos para solucionar problemas corriqueiros, como a organização de sistemas em camadas e a ligação das classes.

Utilizar padrões traz alguns benefícios para a aplicação.
Entre eles, podemos citar:

- aumento de produtividade;
- padronização na estrutura do software;
- facilidade ao manter a aplicação;
- redução de complexidade no código;
- documentação implementada ou facilitada;
- vocabulário comum de projeto entre desenvolvedores;
- permite a reutilização de módulos do sistema em outros sistemas;
- ajuda a construir softwares confiáveis com arquiteturas testadas;
- reduz o tempo de desenvolvimento de um projeto.

Entre os padrões arquiteturais existentes, há três padrões utilizados com o intuito de separar as responsabilidades e organizar o código de interface de usuário, separando o que é modelo, o que é visão e o que é mediador e como eles trabalham.

São eles:

- MVC, ou model–view–controller, que significa modelo–visão–controlador;
- MVP, ou model–view–presenter, que significa modelo–visão–apresentador;
- MVVM, ou model–view–view–model, que significa modelo–visão–visão–modelo.

(Zenker, 2019).

Os três padrões têm objetivos semelhantes, mas são implementados de formas diferentes.
Os seus objetivos são:

- modularidade;
- flexibilidade;
- testabilidade;
- manutenibilidade.

Em geral, cada padrão descreve uma abordagem para o desenvolvimento de software, auxiliando na organização e interação das classes, separando as classes, reduzindo o acoplamento e aumentando a coesão das mesmas. Além disso, esses padrões facilitam muito a manutenção do código e sua reutilização em outros projetos. (Zenker, 2019).

ă Diferenças entre as arquiteturas MVC, MVP e MVVM

As arquiteturas MVC, MVP e MVVM são padrões arquiteturais para a organização e ligação de componentes no desenvolvimento de software.
Os padrões implementam os seguintes componentes ou camadas:

- **modelo (model)** — apresenta a funcionalidade principal e os dados da aplicação (em geral, dados e comportamentos);
- **visão (view)** — trabalha sobre a camada de apresentação ao usuário;
- **controlador (controller), visão-modelo (view-model) ou apresentador (presenter)** — atua no contexto geral para interligar dados e comportamentos à camada de interface do usuário, cada qual com suas responsabilidades e sua implementação lógica.

Padrão model–view–controller

O MVC é um padrão de arquitetura de software utilizado para a organização dos subsistemas de um sistema de software; ele divide uma aplicação em três partes interconectadas. O MVC utiliza uma solução já definida para separar partes distintas do projeto, reduzindo suas dependências ao máximo. Cada parte do padrão — o modelo, o controlador e a visão — executa o que lhe é definido e possui suas responsabilidades.

Segundo Gamma et al. (1994), o padrão MVC é um dos mais conhecidos entre os padrões arquiteturais. O autor explica que esse padrão é dividido entre as funcionalidades principais (modelo) e os dados. A visão mostra as informações aos usuários e os controladores manipulam a entrada de dados do usuário.
Ao utilizar um padrão MVC, as regras de negócio ficam isoladas da lógica de apresentação (interface com o usuário).

Fluxo de controle do MVC

O fluxo de controle do MVC é um evento de interação do usuário, em que:

- o controlador manipula o evento e o converte em ação do usuário para que o modelo entenda, sendo responsável por controlar e mapear as ações;
- o modelo gerencia o comportamento e os dados de domínio da aplicação, sendo responsável pela manutenção dos dados;
- a visão é responsável por incluir os elementos de exibição do cliente.
- Esse padrão foi originalmente criado em 1979 por Trygve Reenskaug, com o intuito de interagir com uma máquina da XEROX que tinha um mouse.

O controlador era o responsável por interagir com os dispositivos externos, fazendo a visão e o modelo serem sincronizados. Desde então, diversas variações foram criadas para acompanhar novas demandas na interação com o usuário.

Entre as suas características está a aplicação a vários tipos de projetos, como desktop, web e mobile.

A Figura 1 apresenta o padrão MVC, em que “a classe Model representa o estado das coisas que tem atributos e operações, como: Produto, Pessoa, Usuário, etc. Todos os exemplos como representação das entidades do banco de dados”, conforme descreve Wilson (2015, documento on-line).

A view contém os componentes visuais e o código com a interface a ser mostrada ao usuário. Também é responsável pela interação do usuário com a aplicação, como a interação do usuário com botões, links, input e outras funções. O componente controller é o responsável por implementar a regra do negócio do software.

Na prática, é responsável por interpretar e responder a requisição do usuário; ou seja, ele analisa uma solicitação do usuário e determina o que será feito a partir disso, conforme aponta Wilson (2015).

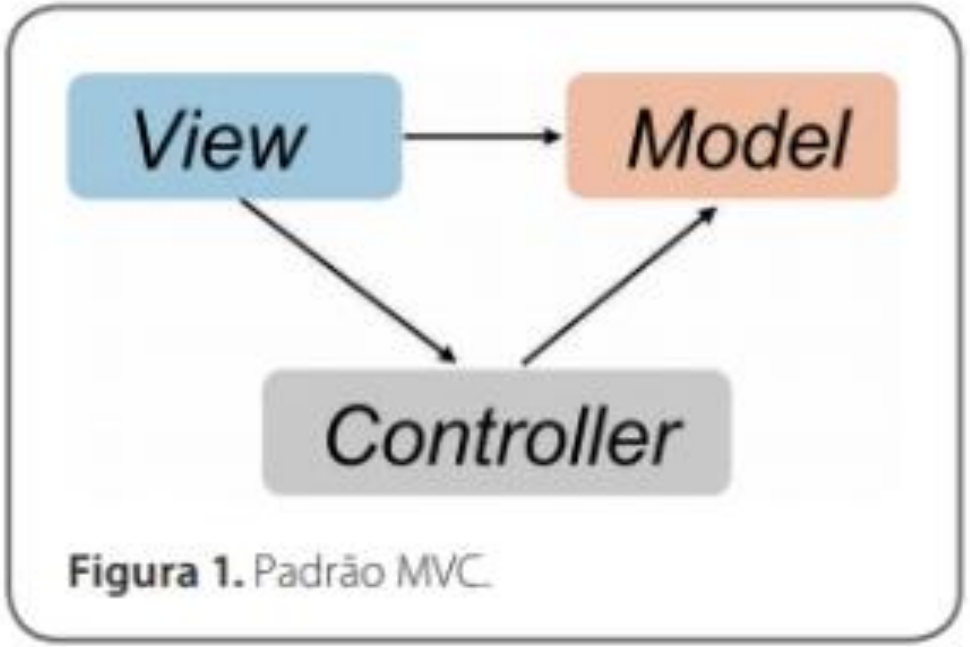


Figura 1. Padrão MVC

Wilson (2015) afirma que a model não pode ser referida como sendo a camada de persistência de dados.
A camada de persistência de dados é a camada responsável pelos códigos, em que persistem os dados das entidades.
A view mostra os dados para o usuário e não processa nada além de um loop, para mostrar os dados na interface gráfica definida ao usuário.
A mesma não faz nenhuma pesquisa diretamente no banco, ela apenas recebe, manipula e mostra os dados ao usuário.
A view não deve ter nenhum tipo de lógica de negócios

ă

Nesse sentido, é possível que um model seja representado por várias views.
Um exemplo é um model de usuário ter uma view que mostra a listagem de usuários e outra view que cria um usuário.
Essas duas views estarão ligadas com o mesmo model, pois tratam e mostram o mesmo tipo de dado, conforme lecionam Gamma et al. (1994).
O fluxo de comunicação da view com o controller também é válido, pois recebe as informações de uma view, como algum input feito pelo usuário, e traduz e envia esses dados para um model específico, que vai persistir no banco da aplicação, conforme aponta Wilson (2015).

Padrão model–view–presenter

O MVP, assim como os demais padrões de arquitetura de software, possui a finalidade de separar a camada de apresentação das camadas de dados e regras de negócio.
Segundo Kouraklis (2016), esse modelo foi inspirado no padrão MVC e surgiu na década de 1990; nele, o controlador é substituído pelo apresentador.
As responsabilidades dos componentes mudaram, e a sincronia da visão com o modelo é executada pelo apresentador.

- **Visão (view):** um objeto de visão implementa uma interface, que expõe campos e eventos aos usuários; sua responsabilidade é comum a todos os padrões arquitetônicos.
- **Modelo (model):** são os objetos com dados e ações que serão manipulados.
- **Apresentador (presenter):** é a ligação entre visão e modelo; possui papel de mediador entre eles. Sua função é atualizar a visão quando o modelo é alterado e sincronizar o modelo em relação à visão.

Esse modelo possui o objetivo de separar a camada de apresentação das camadas de dados e regras de negócio. Basicamente, a diferença entre os dois padrões (MVC e MVP) está no conceito, pois as funções do apresentador e do controlador são semelhantes.

A Figura 2 apresenta o padrão MVP. Kouraklis (2016) afirma que a view não está ciente da model, e o inverso também se aplica; isso é chamado comumente de passive view.

O autor também afirma que o presenter recebe entradas do usuário a partir do componente view, manipula o mapeamento entre a view e o model e realiza lógica de negócios complexa

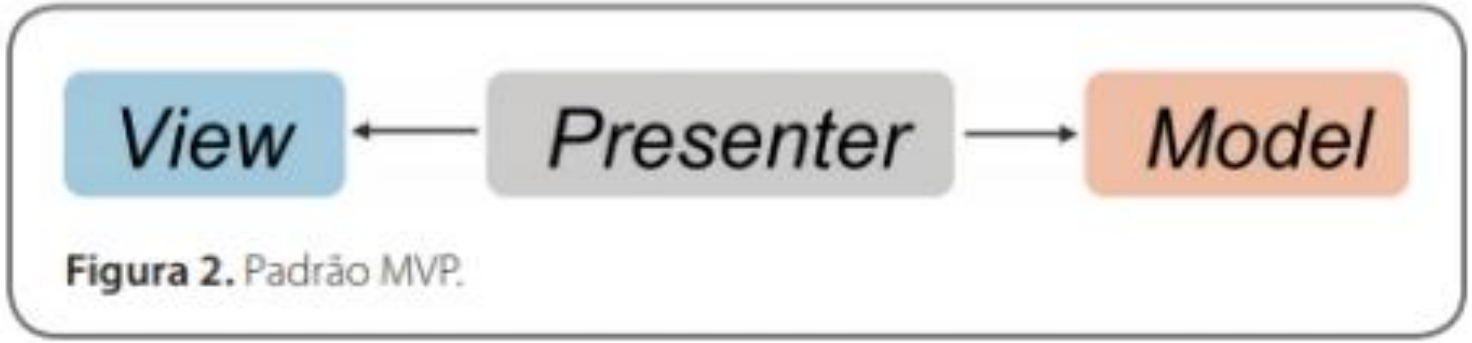


Figura 2. Padrão MVP.

O fluxo de funcionamento do MVP apresentado na Figura 2 mostra a diferença entre o MVC e o MVP.

No padrão MVP, o usuário interage com o sistema por meio de telas disponibilizadas pela view. As requisições do usuário são pegas na view e repassadas ao presenter, que faz o pedido ao model.

O model retorna todos os pedidos ao presenter, que, por fim, encaminha para a view; por meio dela, os pedidos chegam ao usuário.

O usuário interage com a interface acessando, por exemplo, um cadastro de usuário.

ă

O presenter observa os eventos disparados pela view e requer os dados para efetuar o cadastramento, requerendo os mesmos ao model.

O model retorna todos os dados que deverão ser preenchidos para efetuar corretamente o cadastro, como: nome, telefone, e-mail, login e senha.

O presenter retorna a view, que atualiza a tela com os campos que o usuário deverá preencher.

EMPRESAS QUE IMPLEMENTARAM E DEU CERTO O MVP

1° Spotify – musicas – testar o streaming de musica era a cor feature – feito para Desktop e dando certo foi para aplicativos de celular;
2º Groupon- entrega de voucher em pdf para os clientes;
3° AirBed&Breakfast – aluguel de casa como se fosse hotel
4° Facebook- de diretório de alunos de Harvard e, passou a ser utilizado como rede social.
5° Uber- Nasceu em São Francisco, devido a não gostar de taxi muito caro.
6° Aplle iPhone 2G - usuário se interessaria em teclado virtual, internet e multiplataforma.

ă

Cinco passos para criar um MVP

"MVVM é um padrão de projeto baseado em UI, ele é uma aplicação do MVP, que é uma derivação do MVC. Estes padrões de projeto (MVC, MVP e MVVM) procuram atingir os mesmos objetivos mas com soluções diferentes."

Padrão model–view–view-model

O MVVM é um padrão muito semelhante ao MVC e ao MVP. O padrão MVVM foi criado por John Gossman em 2005.
John é arquiteto da Microsoft Azure e foi um dos arquitetos responsáveis pela criação do Windows Presen-tation Foundation (WPF) e da Silverlight.
Pode-se dizer que o MVVM é uma especialização do MVP.
Segundo Kouraklis (2016), o componente view-model substitui os componentes presenter e controller dos outros dois modelos vistos anteriormente.
O model e a view continuarão responsáveis pela mesma área apresentada no padrão MVP.

- **Modelo (model)** — assim como no MVC, o mesmo gerencia o comportamento e os dados de domínio da aplicação. É responsável pela manutenção dos dados.
- **Visão (view)** — age da mesma forma do MVC, interagindo com o usuário por meio de telas. É responsável por incluir os elementos de exibição do cliente.
- **Visão–modelo (view-model)** — é um modelo da visão, que complementa o modelo com comportamentos que a visão possa utilizar, associando dados entre visão e modelo.

Esse vínculo entre visão e modelo permite que a visão passe eventos para o modelo. É responsável pela camada de negócios, manipulando a lógica de negócios da aplicação, armazenando e processando o estado dos widgets da interface do usuário.

Por fim, vincula os dados do modelo para o modo de exibição da visão.

De acordo com a Microsoft (2018), o modelo de exibição também pode definir membros para manter o controle de dados que são relevantes para a interface do usuário, mas não para o modelo, como a ordem de exibição de uma lista de itens. O modelo de exibição também serve como um ponto de integração com outros serviços, como o código de acesso de banco de dados. Para projetos simples, talvez não seja necessária uma camada de modelo separada, mas apenas um modelo de exibição que encapsula todos os dados necessários. Visão e modelo possuem comportamentos iguais aos demais padrões.

A Figura 3 apresenta o padrão MVVM. Nesse padrão, é feito um mapeamento da view, fazendo com que o view-model seja responsável por criar toda a lógica de negócios e testar o estado da view, interagindo com a mesma sempre que precisar de sincronia dos dados e estados. Assim, a view-model interage com o model, possibilitando que a view da camada de apresentação possa interagir com os dados do model.

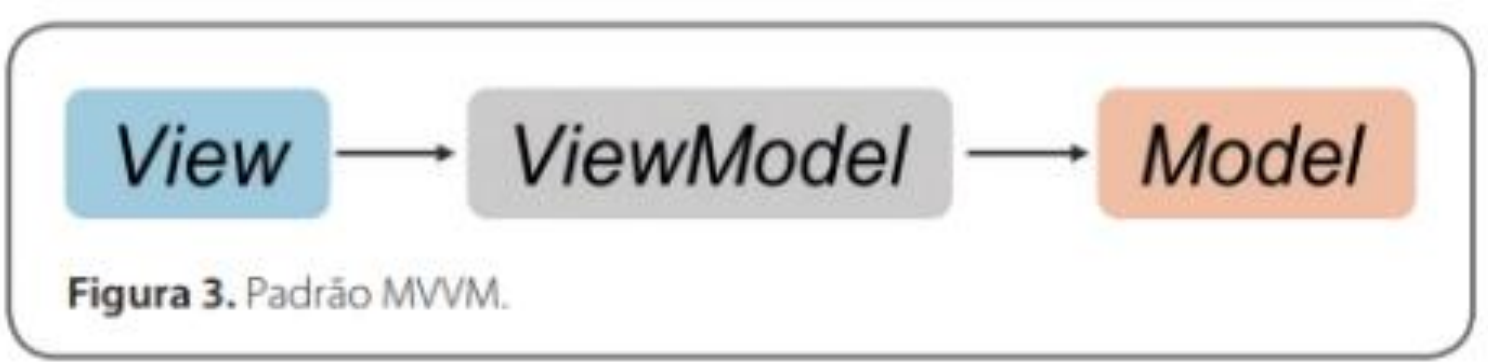


Figura 3. Padrão MVVM.

Utilização dos padrões MVC, MVP ou MVVM

Os padrões MVC, MVP e MVVM têm como objetivo isolar ao máximo a camada de apresentação de um sistema.
As letras M e V, similares nos três padrões, referem-se às camadas de modelo (representação de dados e ações do sistema) e às camadas de visão (representação gráfica do modelo, interface), respectivamente.
As demais letras — C, P e VM — se referem a controlador, apresentação e visão-modelo, respectivamente, que exercem, em geral, a função de ligar logicamente as demais camadas.
Porém, cada padrão exerce essa função de forma diferente, com enfoque em um determinado tipo de sistema.

Para saber avaliar qual é o melhor para a aplicação, é necessário verificar a testabilidade, a capacidade de manutenção, o desempenho e a funcionalidades dos três padrões.

Realizar os seguintes questionamentos facilita a decisão:

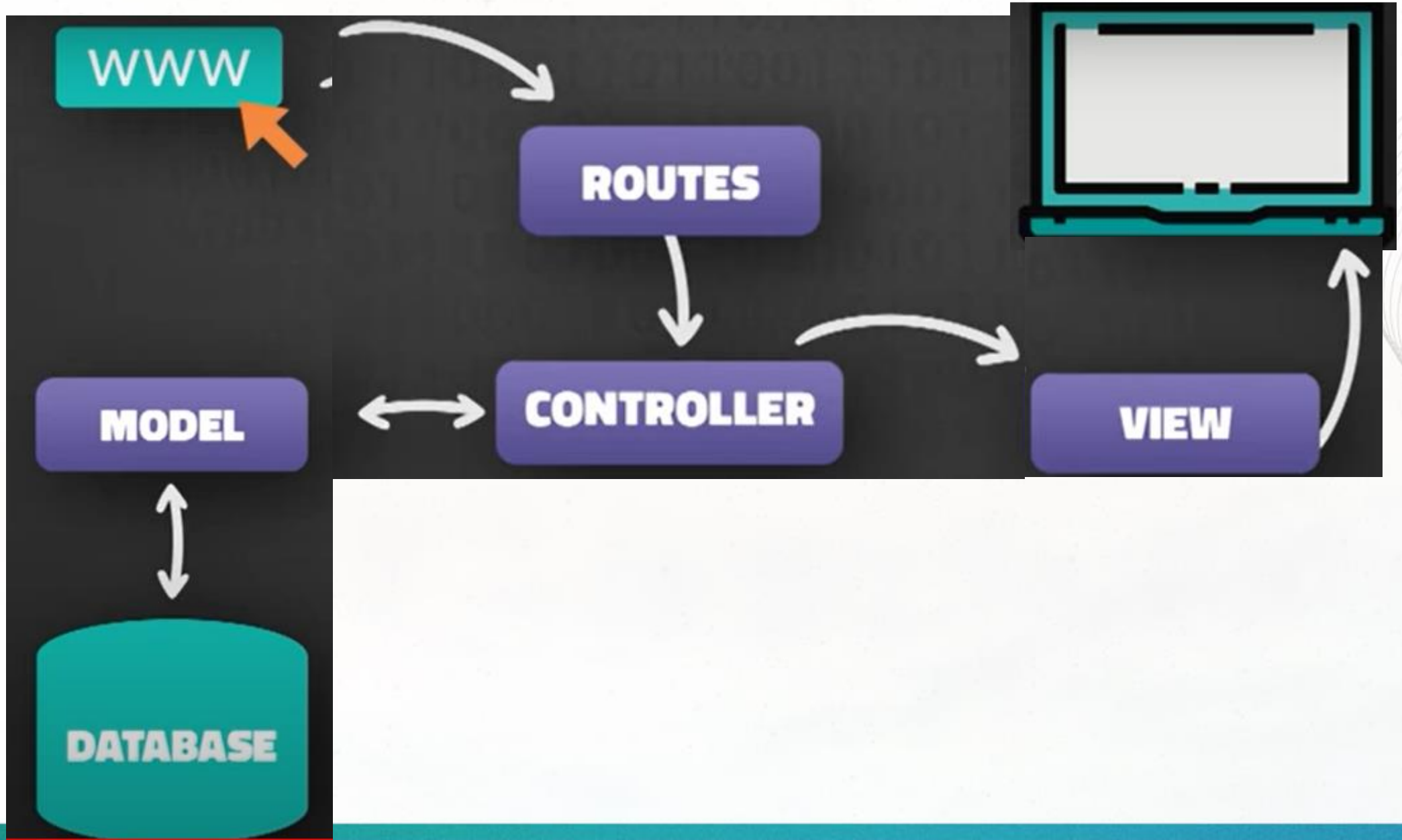
- Qual é o desempenho do sistema após a implementação? A aplicação utiliza quanto de CPU/GPU, alocação de memória, entre outros aspectos?
- Quão testável é a aplicação?
- Qual é a facilidade para fazer modificações e alterações no código?

ă

A seguir, vejamos as finalidades de cada padrão.

- **MVC** — para uma aplicação muito simples com apenas uma ou duas telas, o MVC pode funcionar bem. Em aplicações mais complexas, quando suas atividades e classes de fragmentos tendem a crescer, torna-se difícil para o aplicativo evoluir. Esse padrão não facilita a manutenção; em caso de alteração da view, por exemplo, deve-se retornar e alterar o controlador também. Além disso, o controlador dificulta os testes de unidade.
- **MVP** — para aplicações com lógica de estado da interface do usuário. Com o MVP, é muito mais fácil testar a unidade e simular a visualização. O mesmo traz como benefício a melhor testabilidade.
- **MVVM** — para aplicações com uso de associação de dados (data binding) bidirecional. O MVVM desempenha um ótimo papel quando se deseja que o código seja mais bem dimensionado no futuro. O mesmo permite uma resposta mais rápida às alterações de projeto, além de facilitar a utilização de testes unitários. (Zenker, 2019).

Arquitetura de Software MVC



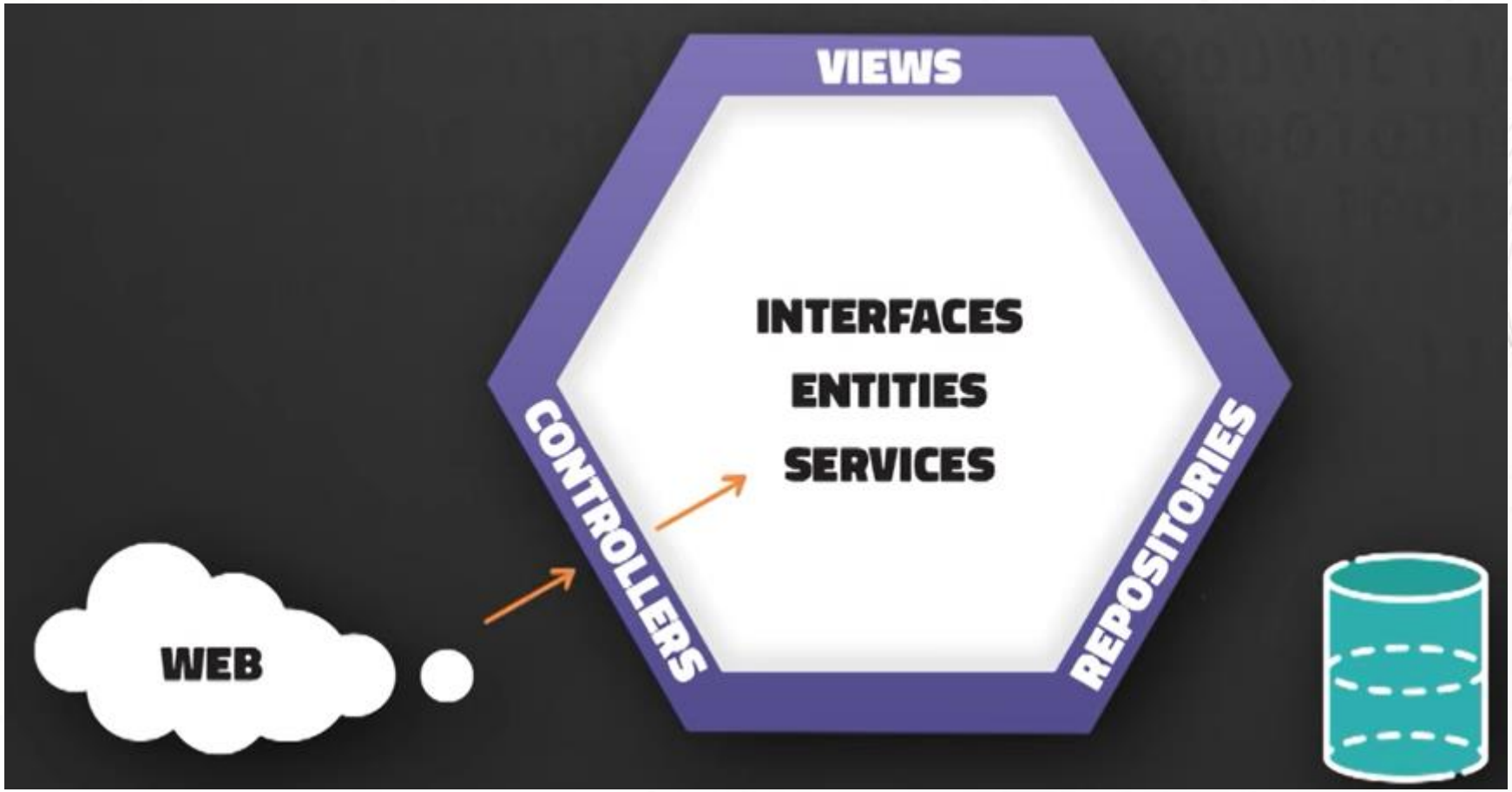
```php
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Categoria extends Model { }
```

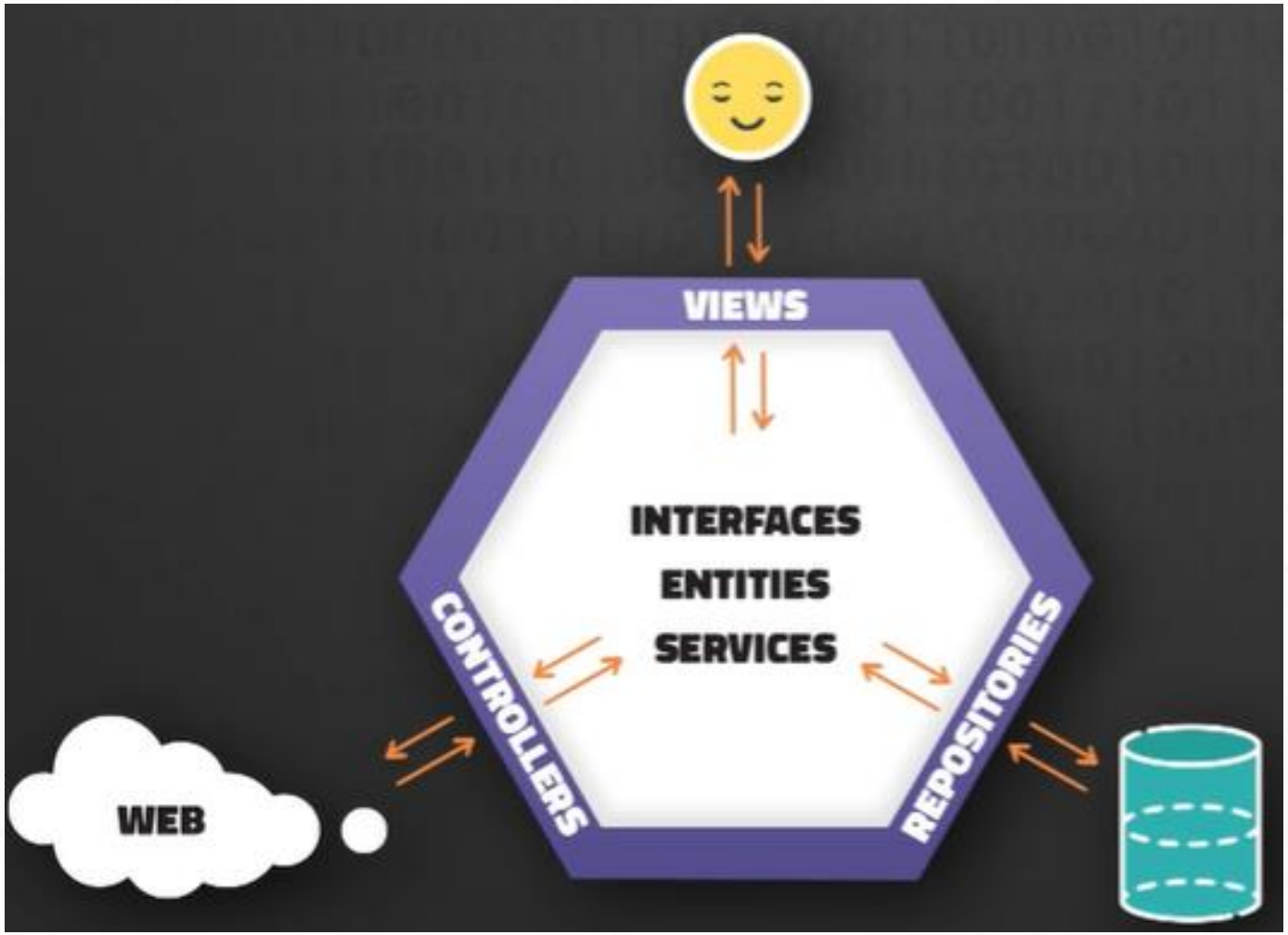
```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use App\Categoria;

class CategoriaController extends Controller
{
  public function index()
    {
      $categorias = Categoria::all();
      return view('categorias.list', ['categorias' => $categorias]);
    }
}
```

```blade
<!DOCTYPE html>
<html>
  <head>
    <title>Lista de Categorias</title>
  </head>
  <body>
    <h1>Categorias:</h1>
    <ul>
      @foreach ($categorias as $categoria)
        <li>{{ $categoria->nome }}</li>
      @endforeach
    </ul>
  </body>
</html>
```

ă

ARQUITETURA EM CAMADAS




VIEWS
INTERFACES
ENTITIES
SERVICES
CONTROLLERS
REPOSITORIES
WEB

BIBLIOGRAFIA

Pressman, Roger S. Engenharia de software : uma abordagem profissional [recurso eletrônico] / Roger S. Pressman, Bruce R. Maxim ; [tradução: João Eduardo Nóbrega Tortello ; revisão técnica: Reginaldo Arakaki, Julio Arakaki, Renato Manzan de Andrade]. – 8. ed. – Porto Alegre : AMGH, 2016.

Morais, Izabelly Soares de. Engenharia de software [recurso eletrônico] / Izabelly Soares de Morais, Aline Zanin ; revisão técnica : Jeferson Faleiro Leon. – Porto Alegre : SAGAH, 2017.

PRESSMAN, Roger; MAXIM, Bruce. Engenharia de Software. Uma abordagem profissional. 8a. Ed. Bookman, 2016. https://integrada.minhabiblioteca.com.br/#/books/9788580555349/cfi/3!/4/2@100:0.00

SOMMERVILLE, Ian. Engenharia de Software. 9. ed. São Paulo: Pearson Prentice Hall, 2011. https://bv4.digitalpages.com.br/?term=engenharia%2520de%2520software&searchpage=1&filtro=todos&from=busca&page=_14&section=0#/legacy/276

LARMAN, Craig. Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e desenvolvimento iterativo. 3. ed Porto Alegre: Bookman, 2007. https://integrada.minhabiblioteca.com.br/#/books/9788577800476/cfi/0!/4/2@100:0.00

Aline Maciel Zenker... [et al.]. Arquitetura de sistemas [recurso eletrônico] / ; [revisão técnica: Jésus Henrique Segantini, Júlio Henrique Araújo Pereira Machado, Maria de Fátima Webber do Prado Lima].–Porto Alegre : SAGAH, 2019.

BIBLIOGRAFIA

Pressman, Roger S. Engenharia de software : uma abordagem profissional [recurso eletrônico] / Roger S. Pressman, Bruce R. Maxim ; [tradução: João Eduardo Nóbrega Tortello ; revisão técnica: Reginaldo Arakaki, Julio Arakaki, Renato Manzan de Andrade]. – 8. ed. – Porto Alegre : AMGH, 2016.

Morais, Izabelly Soares de. Engenharia de software [recurso eletrônico] / Izabelly Soares de Morais, Aline Zanin ; revisão técnica : Jeferson Faleiro Leon. – Porto Alegre : SAGAH, 2017.

PRESSMAN, Roger; MAXIM, Bruce. Engenharia de Software. Uma abordagem profissional. 8a. Ed. Bookman, 2016. https://integrada.minhabiblioteca.com.br/#/books/978580555349/cfi/3!/4/2@100:0.00

SOMMERVILLE, Ian. Engenharia de Software. 9. ed. São Paulo: Pearson Prentice Hall, 2011. https://bv4.digitalpages.com.br/?term=engenharia%2520de%2520software&searchpage=1&filtro=todos&from=busca&page=_14&section=0#/legacy/276

LARMAN, Craig. Utilizando UML e padrões: uma introdução à análise e ao projeto orientados a objetos e desenvolvimento iterativo. 3. ed Porto Alegre: Bookman, 2007. https://integrada.minhabiblioteca.com.br/#/books/978577800476/cfi/0!/4/2@100:0.00

Prikladnicki Rafael, Willi Renato, Milani Fabiano. Métodos ágeis para desenvolvimento de software / Organizadores, Rafael Prikladnicki, Renato Willi, Fabiano Milani. – Porto Alegre : Bookman, 2014.

IFSC. Ciclo de Vida Iterativo e Incremental. Wiki Instituto Federal de Santa Catarina, São José, out. 2006. Disponível em: <https://wiki.sj.ifsc.edu.br/wiki/index.php/Ciclo_de_Vida_Iterativo_e_Incremental>. Acesso em: 31 ago. 2017.

Layton, Mark C.; Ostermiller, Steven J. Gerenciamento Ágil de Projetos Para Leigos. Traduzido por Eveline Vieira Machado. Rio de Janeiro Alta Books, 2019.