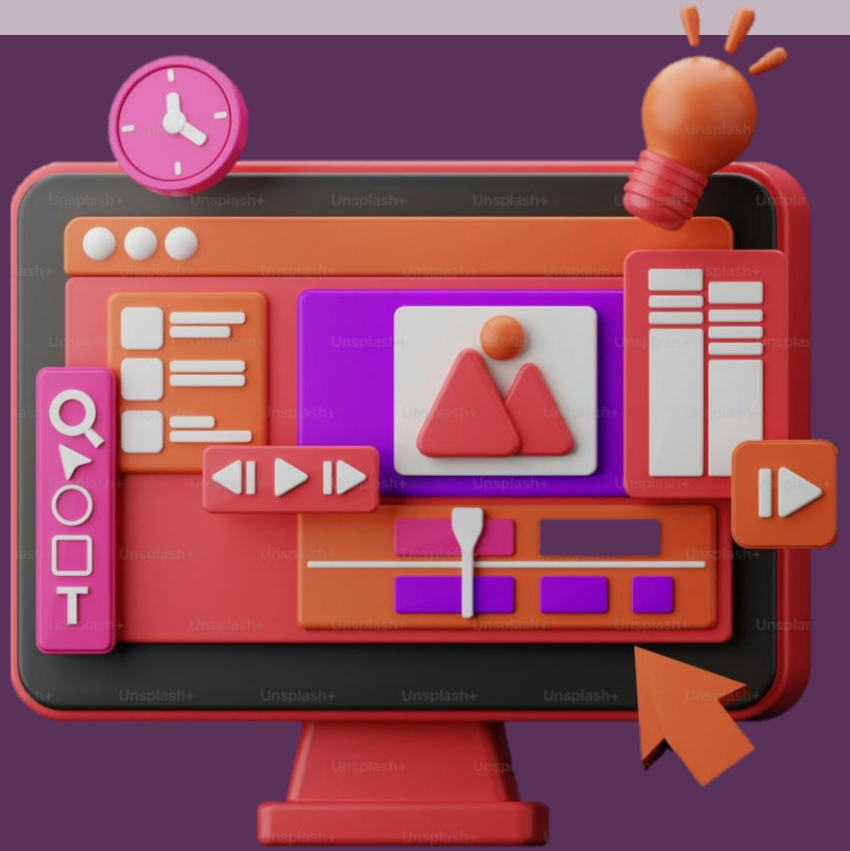
ENGENHARIA DE SOFTWARE

AULA 3 – PROCESSOS DE SOFTWARE



WALTER TRAVASSOS SARINHO

@WALTEROPROFESSOR

WALTER.TRAVASSOS@SECTRAS.EDU.BR

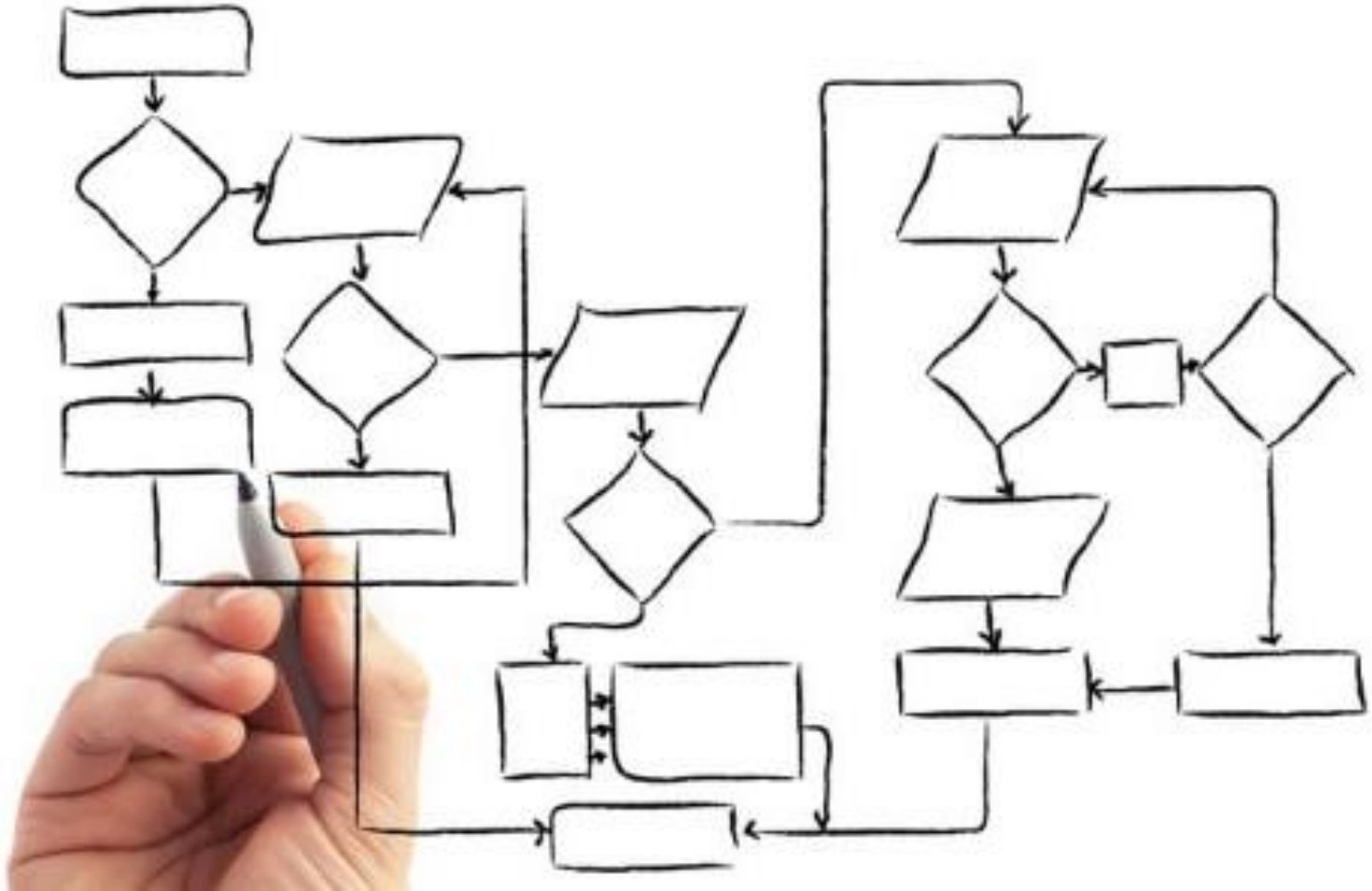
Lista de Exercícios 2

Aula de Hoje

- Os Processos de Software Tradicionais e Ágeis.
- Metodologias de Desenvolvimento Ágil.
- Scrum
- Kanban
- XP

O que é um Processo de Software?

É um conjunto de atividades relacionadas que levam à produção de um produto de software.



Atividades do Processo

- Os processos de software reais são **sequências intercaladas de atividades técnicas, colaborativas e gerenciais**, cujo objetivo global é **especificar projetar, implementar e testar um sistema de software**.
- Geralmente, os **processos são apoiados por ferramentas**. Isso significa que os desenvolvedores de software podem usar uma gama de ferramentas de software para ajudá-los, como: **sistemas de gerenciamento de requisitos, editores de modelo de projeto (design), editores de programa, ferramentas de teste automatizadas e depuradores**.

Etapas (mínimas) fundamentais para a engenharia de software (SOMMERVILLE, 2011)

1. **Especificação do Software** – a funcionalidade e as restrições.
2. **Projeto e Implementação de Software** – produção do software que atenda as especificações.
3. **Validação do Software** – atender demandas do cliente.
4. **Evolução do Software** – atender necessidades de mudanças.

As quatro atividades de processo básicas (SOMMERVILLE, 2018)

- São (i) a **especificação**, (ii) o **desenvolvimento**, (iii) a **validação** e a (iv) **evolução** — são organizadas de modo distinto em diferentes processos de desenvolvimento.
- No **modelo em cascata**, elas são organizadas em sequência; no desenvolvimento incremental, são intercaladas.
- O modo como essas atividades são executadas depende do tipo de software que está sendo desenvolvido, da experiência e da competência dos desenvolvedores e do tipo de empresa que o desenvolve.

Processos e Atividades

- Cada uma dessas 4 atividades fazem parte de todo processo de desenvolvimento de software.
- São **atividades complexas** que incluem **subatividades**.

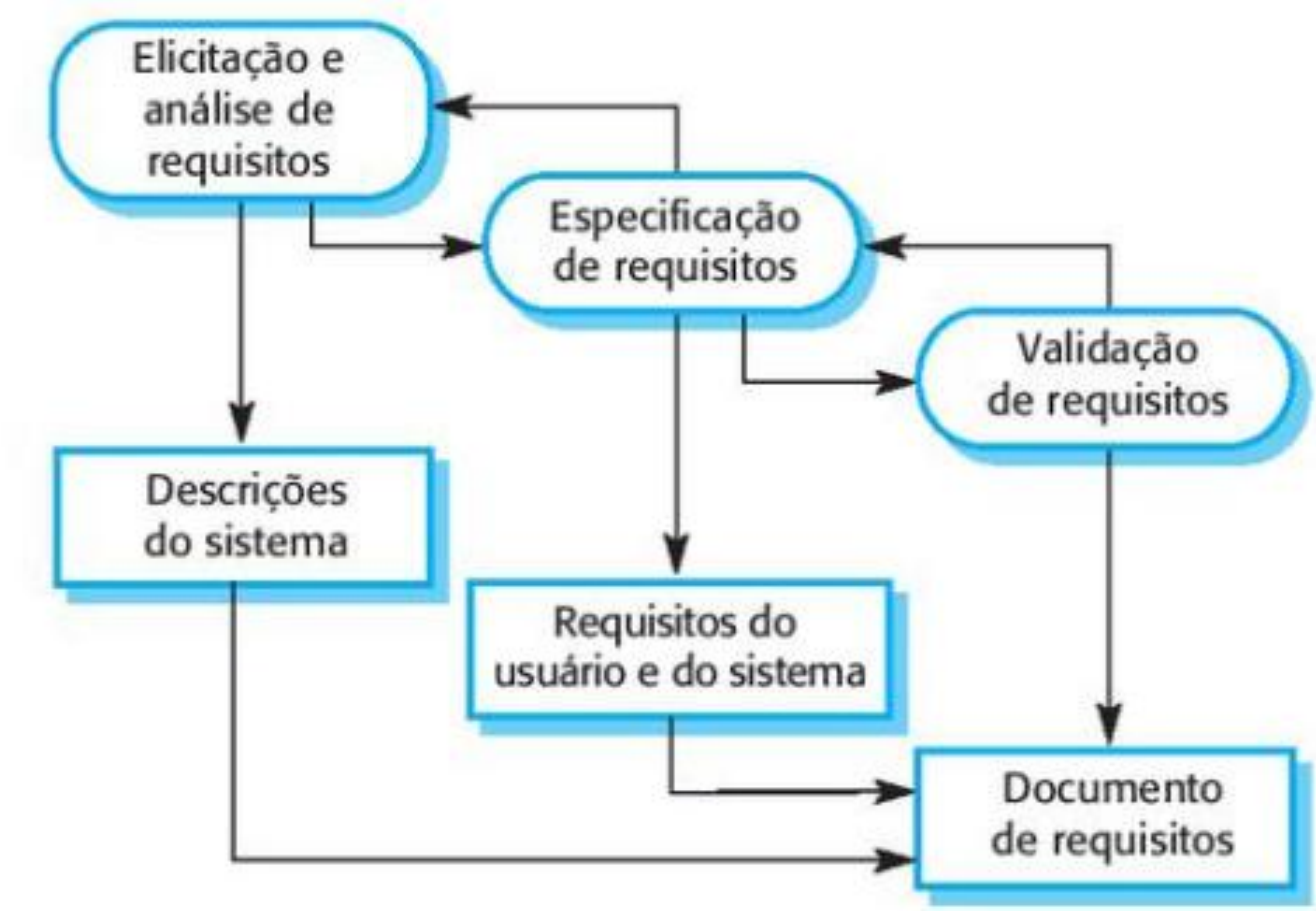
1. Especificação do Software

- **Especificação do software** ou **engenharia de requisitos** é o processo de compreender e definir quais serviços são necessários para o sistema e identificar as restrições sobre sua operação e desenvolvimento.
- A engenharia de requisitos é um estágio **particularmente crítico** do processo de software, já que os erros cometidos nessa etapa inevitavelmente geram problemas posteriores no projeto e na implementação do sistema.
- Antes de iniciar o processo de engenharia dos requisitos, uma empresa pode realizar um **estudo de viabilidade** ou de **marketing** para avaliar se há ou não uma demanda ou um mercado para o software e se ele é realista ou não em termos técnicos e financeiros.
- Os **estudos de viabilidade são de curto prazo**, relativamente baratos e orientam a decisão de ir adiante ou não com uma análise mais detalhada.

1. Especificação do Software

- O processo de engenharia de requisitos visa à **produção de um documento de requisitos** acordados que especifique um sistema que satisfaça os requisitos dos stakeholders.
- Os requisitos são apresentados normalmente em dois níveis de detalhe. Os **usuários finais** e os **clientes** precisam de uma **declaração de requisitos mais superficial** desse sistema; os **desenvolvedores**, de uma **especificação mais detalhada**.

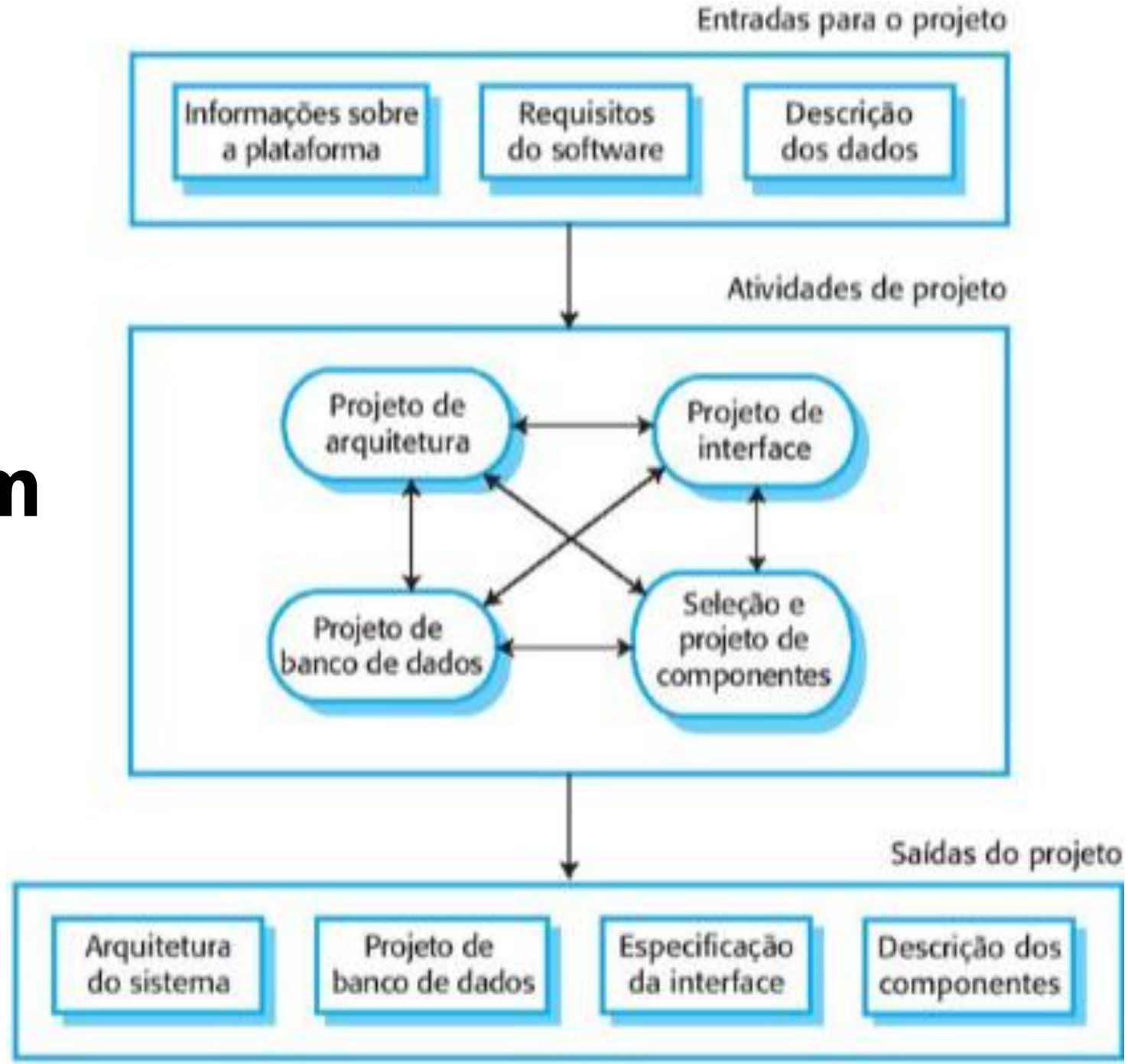
O processo da Engenharia de Requisitos (SOMMERVILLE, 2018)



2. Projeto e implementação do software

- O estágio de **implementação** no desenvolvimento de software é o **processo de elaborar um sistema executável para ser entregue ao cliente**.
- O projeto de software é uma **descrição da estrutura do software a ser implementado, dos modelos e estruturas de dados utilizados pelo sistema**, das **interfaces entre os componentes do sistema** e, às vezes, do **algoritmo utilizado**.
- Os projetistas acrescentam detalhes à medida que desenvolvem seu projeto, com revisões constantes para modificar os projetos iniciais.

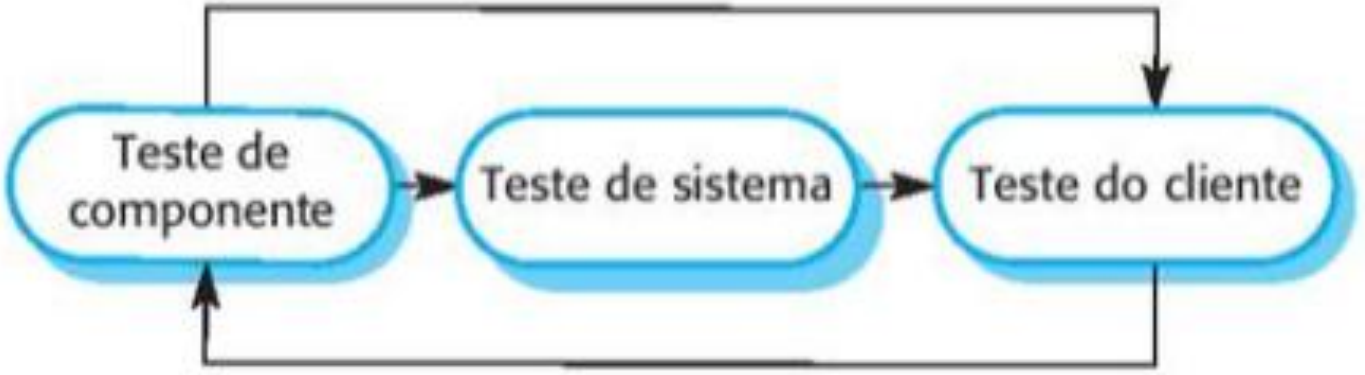
Modelo Geral de um Processo de Projeto



3. Validação do software

- A validação do software ou, em termos mais gerais, Verificação e Validação (V & V) destina-se a mostrar que **um sistema está em conformidade com sua especificação e que satisfaz as expectativas do cliente do sistema**.
- A **principal técnica de validação** é o **teste de programa**, no qual o sistema é executado usando dados de teste simulados.
- A validação também pode envolver processos de conferência como inspeções e revisões em cada estágio do processo de software, desde a definição dos requisitos do usuário até o desenvolvimento do programa. Entretanto, a **maior parte do tempo e esforço de V & V é consumida no teste do programa**.

Estágios do Teste



- A Figura mostra um processo de teste em três estágios, no qual os componentes do sistema são testados individualmente e, depois, o sistema integrado é testado.
- No software personalizado, o teste do cliente envolve testar o sistema com dados reais do cliente.
- Nos produtos vendidos como aplicações, o teste do cliente é feito por usuários selecionados, que experimentam e comentam o software, o que é conhecido como teste-beta.

4. Evolução do software

- A **flexibilidade do software** é uma das principais razões pelas quais, cada vez mais, ele é incorporado a sistemas grandes e complexos.
- Por exemplo, depois de tomada a decisão de produzir o hardware, é muito caro fazer alterações em seu projeto (design). Entretanto, alterações no software podem ser feitas a qualquer momento, durante ou depois do desenvolvimento do sistema.
- **Mesmo as grandes mudanças no software ainda são muito mais baratas do que mudanças equivalentes no hardware do sistema.**

2000

Peso 15,8kg
Altura 38,1cm
Largura 38,1cm
Profundidade 43,5cm
Preço US$ 1499
CPU 500MHz
RAM 128MB
Resolução 1024x768
Armazenamento 13 a 30GB
Modelo **iMac DV SE**

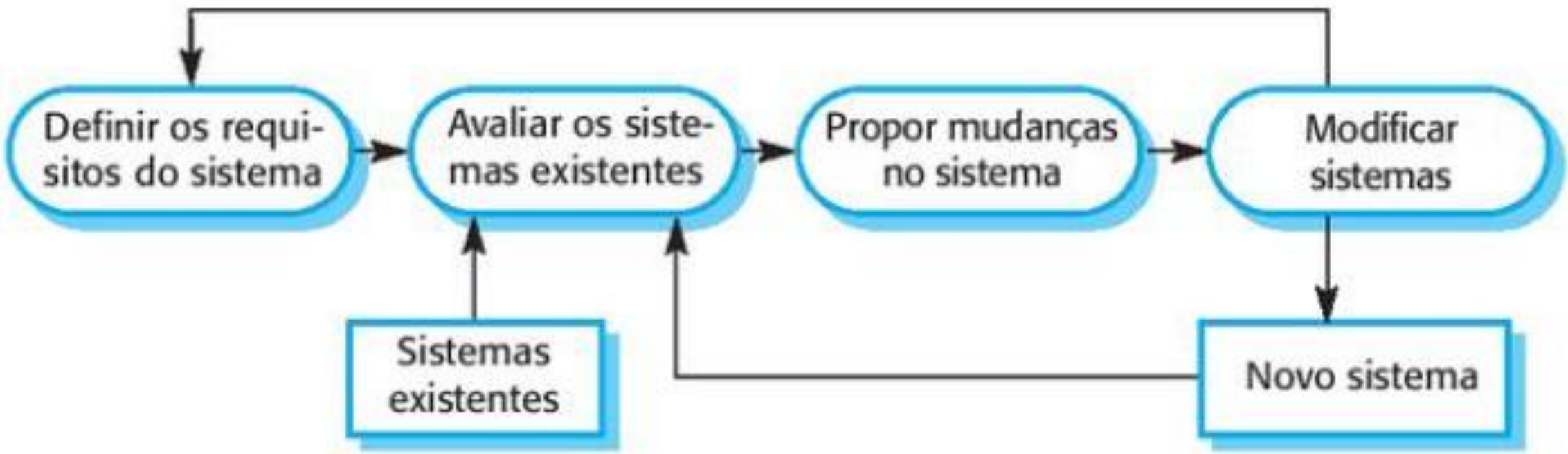
2010

Peso 137g
Altura 11,5cm
Largura 5,8cm
Profundidade 0,9cm
Preço US$ 699
CPU 1GHz
RAM 512MB
Resolução 960x640
Armazenamento 32GB
Modelo **iPhone 4**

Evolução do software

- Historicamente, sempre houve uma divisão entre o **processo de desenvolvimento** e o **processo de evolução** (manutenção) do software, no entanto, essa distinção é cada vez mais irrelevante.
- É mais realista encarar **a engenharia de software como um processo evolutivo**, no qual o software é alterado continuamente ao longo de sua vida útil em resposta à mudança dos requisitos e das necessidades do cliente.

Evolução do Sistema de Software

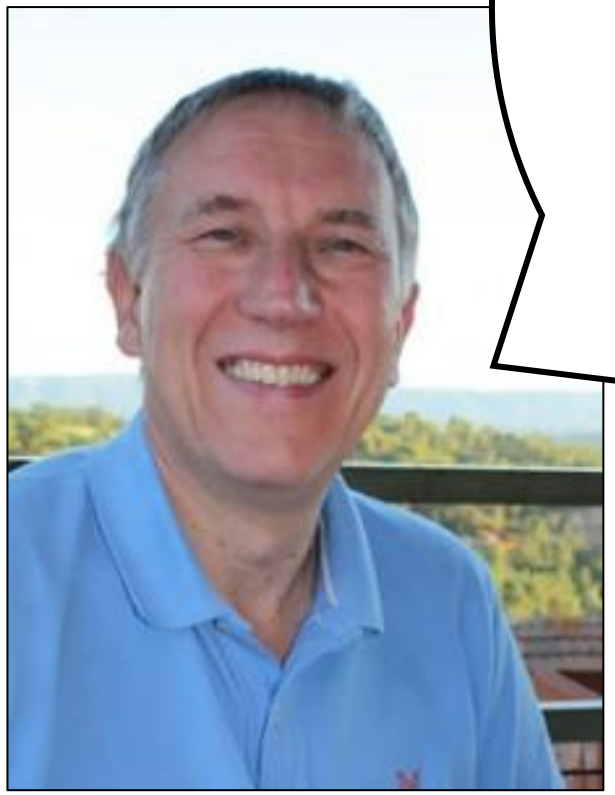


Aplicações de Negócio

As aplicações de negócios não são, necessariamente, desenvolvidas seguindo as 4 etapas que apresentamos.

Softwares novos, em geral, são desenvolvidos por meio de:

1. **Extensão** e **modificação** de **sistemas existentes**.
2. **Configuração** e **integração** de **componentes já existentes do sistema**.



NÃO EXISTE UM PROCESSO IDEAL, A MAIORIA DAS ORGANIZAÇÕES DESENVOLVE OS PRÓPRIOS PROCESSOS DE DESENVOLVIMENTO DE SOFTWARE!

(SOMMERVILLE, 2011, P. 19).

Será que existe uma forma de produzir software mais rapidamente?

Metodologias de Desenvolvimento Ágil

Introdução

A melhor maneira de obter o **melhor software** é por meio de um cuidadoso planejamento de projeto, garantia da qualidade formalizada, uso de métodos de análise e projeto apoiado por ferramenta CASE*.

- V ou F?
- Verdadeiro... Para a década de 1980!

*CASE: *Computer-Aided Software Engineering* - é uma classificação que abrange todas ferramentas baseadas em computadores que auxiliam atividades de engenharia de software, desde análise de requisitos e modelagem até programação e testes.

Metodologia ágil
x
não tão ágil

Desenvolvimento Ágil de Software

Ian Sommerville – Engenharia de Software - Capítulo 3

- Uma **metodologia** é um conjunto estruturado de práticas (por exemplo: Material de Treinamento, Programas de educação formais, Planilhas e Diagramas) que pode **ser repetível** durante o processo de produção de software.
- As **Metodologias de Engenharia de Software** abrangem muitas disciplinas, incluindo o Gerenciamento de Projetos em suas diversas fases: análise, projeto, codificação, teste, e também, a garantia da qualidade.

Ágil x Não Ágil

- Os processos de desenvolvimento de software que são **baseados em especificações completas** de:
  i. Requisitos
  ii. Projeto
  iii. Construção
  iv. Teste de sistema
- não estão voltados para o **desenvolvimento rápido de software**.

No desenvolvimento baseado em especificações completas...

O que acontece quando os requisitos mudam?

Resposta: o Projeto do sistema precisa ser retrabalhado, novamente documentado, implementado e re-testado.


QUAIS CARACTERÍSTICAS
DO **MERCADO DE TRABALHO**
INFLUENCIARAM O
SURGIMENTO DAS
METODOLOGIAS ÁGEIS?

Demandas

1. **Necessidade** de suprir novas **demandas** rapidamente.
2. **Tempo** - desenvolvimento e **entregas rápidas** são requisitos críticos.
3. Os **requisitos** propostos **mudam** (inevitavelmente).
4. Pode ser que, somente após a **entrega do sistema** e este ser **experimentado pelos usuários** é que os **reais requisitos se tornem claros.**

Como Funciona o Desenvolvimento Rápido de Software?

- Geralmente são processos iterativos em que as etapas de (i) especificação, (ii) projeto, (iii) desenvolvimento e (iv) teste são **intercaladas**.
- O software não é disponibilizado integralmente, mas sim, a partir de uma série de **incrementos**.
- Cada incremento apresenta uma nova **funcionalidade** do sistema.


ASSIM, PODEMOS RESUMIR 3 CARACTERÍSTICAS FUNDAMENTAIS DO DESENVOLVIMENTO ÁGIL DE SOFTWARE!

As 3 Características Fundamentais do Desenvolvimento Ágil de Software

1. Os processos de especificação, projeto e implementação são concorrentes.

Não existe especificação detalhada do sistema e a documentação do projeto é minimizada (ou gerada automaticamente).

O documento de requisitos define apenas as características mais importantes do sistema.

As 3 Características Fundamentais do Desenvolvimento Ágil de Software

2. O sistema é desenvolvido por meio de uma série de incrementos.

Usuários finais e todos os *stakeholders* participam da especificação e avaliação de cada incremento.

Novos requisitos e funcionalidades podem ser propostas para implementação no próximo incremento do sistema.

As 3 Características Fundamentais do Desenvolvimento Ágil de Software

3. As interfaces com o usuário do sistema são geralmente desenvolvidas usando-se um sistema de desenvolvimento interativo que permite que o projeto de interface seja criado rapidamente por desenho e inserção de ícones na interface.

Ou seja, desenvolver de forma ágil requer uma ferramenta **RAD**!

O DESENVOLVIMENTO INCREMENTAL PERMITE A PRODUÇÃO E ENTREGA DE SOFTWARE EM INCREMENTOS (EM VEZ DE UM ÚNICO PACOTE).

QUAIS AS DUAS PRINCIPAIS VANTAGENS DESSA ABORDAGEM?

ESSA É FÁCIL...

1 - ENTREGA ACELERADA DOS SERVIÇOS AO CLIENTE.

2 - ENGAJAMENTO DO USUÁRIO COM O SISTEMA.


ATÉ AGORA,
SÓ VI COISA BOA...
CADÊ OS PROBLEMAS
DESSE TIPO DE
DESENVOLVIMENTO?!?!

Principais Problemas Relacionados ao Desenvolvimento Iterativo e Incremental

1. **Problemas de Gerenciamento** – requisitos mudam rapidamente. Vale a pena documentar? É preciso aprender a utilizar novas tecnologias.
2. **Problemas de Contrato** – Como fechar um valor para o software se os requisitos estão sempre mudando a cada incremento?
3. **Problemas de Validação** – Intercalar o desenvolvimento nas etapas de especificação e desenvolvimento retarda a etapa de testes.
4. **Problemas de Manutenção** – Alterações contínuas corrompem a estrutura do software. Novos desenvolvedores que trabalham nesses projetos encontram dificuldades para compreender o software.

ESSES SÃO OS PRINCIPAIS CONCEITOS ACERCA DO DESENVOLVIMENTO ÁGIL...

A PARTIR DE AGORA VAMOS COLOCAR A MÃO NA MASSA NAS MÉTODOS ÁGEIS!

OS MÉTODOS ÁGEIS

Intervalo: 15 minutos

Kanban, Scrum e XP

5 Características Comuns dos Métodos Ágeis

Princípio	Descrição
Envolvimento do Cliente	Clientes devem ser profundamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos requisitos do sistema e avaliar as iterações do sistema.
Entrega incremental	O software é desenvolvido em incrementos e o cliente especifica os requisitos a serem incluídos em cada incremento.
Pessoas, não processo	As habilidades da equipe de desenvolvimento devem ser reconhecidas e exploradas. Os membros da equipe devem desenvolver suas próprias maneiras de trabalhar sem processos prescritivos.
Aceite mudanças	Tenha em mente que os requisitos do sistema vão mudar, por isso projete o sistema para acomodar essas mudanças.
Mantenha a simplicidade	Concentre-se na simplicidade do software que está sendo desenvolvido e no processo de desenvolvimento. Sempre que possível, trabalhe ativamente para eliminar a complexidade do sistema.

Parece óbvio, mas lembrem-se dos 7 mandamentos da TAM em 1997

São frequentemente associados ao contexto da cultura interna da empresa naquela época, inclusive no período do trágico acidente do voo TAM 3054 em Congonhas.

1. Nada substitui o lucro
2. Em busca do ótimo não se faz o bom
3. Mais importante que o cliente é a segurança
4. A maneira mais fácil de ganhar dinheiro é parar de perder
5. Pense muito antes de agir
6. A humildade é fundamental
7. Quem não tem inteligência para criar tem que ter coragem para copiar

6 Limitações dos Métodos Ágeis

Existem situações onde a utilização dos métodos ágeis é bem difícil:

1. Necessita de um **cliente disposto** a se envolver no processo.
2. **Personalidade inadequada** para trabalhar em equipe.
3. **Priorizar mudanças** se torna difícil com uma grande quantidade de *stakeholders*.

6 Limitações dos Métodos Ágeis

4. Manter a **simplicidade requer tempo** (e nem sempre o cronograma permite...).
5. **Não é adequado a todo tipo de desenvolvimento.**
6. **Não é adequado para o desenvolvimento de sistemas críticos** (implicações de segurança).

É adequado para pequenas (e médias) empresas, como também para produtos que serão utilizados em computadores pessoais.

Manifesto Ágil

https://agilemanifesto.org/iso/ptbr/manifesto.html

Estamos descobrindo maneiras melhores de desenvolver software, fazendo-o nós mesmos e ajudando outros a fazerem o mesmo. Através deste trabalho, passamos a valorizar:

Indivíduos e interações mais que processos e ferramentas
Software em funcionamento mais que documentação abrangente
Colaboração com o cliente mais que negociação de contratos
Responder a mudanças mais que seguir um plano

Ou seja, mesmo havendo valor nos itens à direita, valorizamos mais os itens à esquerda.

Kent Beck
Mike Beedle
Arie van Bennekum
Alistair Cockburn
Ward Cunningham
Martin Fowler

James Grenning
Jim Highsmith
Andrew Hunt
Ron Jeffries
Jon Kern
Brian Marick

Robert C. Martin
Steve Mellor
Ken Schwaber
Jeff Sutherland
Dave Thomas

Também conhecido como Uncle Bob (Robert Cecil Martin)

- É uma grande personalidade da comunidade de **desenvolvimento de software, métodos ágeis** e ***software craftsmanship***, atuando na área desde 1970.
- Atualmente é consultor internacional e autor de vários livros abordando o tema.
- Tio Bob foi um dos 17 signatários originais do ***Agile Manifesto*** em 2001.

12 Princípios do Manifesto Ágil

https://agilemanifesto.org/iso/ptbr/principles.html

Nós seguimos estes princípios:

Nossa maior prioridade é satisfazer o cliente através da entrega contínua e adiantada de software com valor agregado.

Mudanças nos requisitos são bem-vindas, mesmo tardiamente no desenvolvimento. Processos ágeis tiram vantagem das mudanças visando vantagem competitiva para o cliente.

Entregar frequentemente software funcionando, de poucas semanas a poucos meses, com preferência à menor escala de tempo.

Pessoas de negócio e desenvolvedores devem trabalhar diariamente em conjunto por todo o projeto.

Construa projetos em torno de indivíduos motivados. Dê a eles o ambiente e o suporte necessário e confie neles para fazer o trabalho.

O método mais eficiente e eficaz de transmitir informações para e entre uma equipe de desenvolvimento é através de conversa face a face.

Software funcionando é a medida primária de progresso.

Os processos ágeis promovem desenvolvimento sustentável. Os patrocinadores, desenvolvedores e usuários devem ser capazes de manter um ritmo constante indefinidamente.

Contínua atenção à excelência técnica e bom design aumenta a agilidade.

Simplicidade--a arte de maximizar a quantidade de trabalho não realizado--é essencial.

As melhores arquiteturas, requisitos e designs emergem de equipes auto-organizáveis.

Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz e então refina e ajusta seu comportamento de acordo.

Software Craftsmanship
Software Artesanal

- Traduzindo literalmente, Software Craftsmanship significa algo em como “Software Artesanal”.
- O movimento gira em torno de sermos “melhores artesões” de software.
- Apesar do manifesto do Software Craftsmanship só ter sido criado em 2009, suas raízes vem desde 1999, com a publicação do livro *Pragmatic Programmer*, que trata sobre como ser um programador melhor, em diversos sentidos da palavra.

Definição de Software Craftsmanship

- É um **estilo de vida** em que os desenvolvedores escolhem ser **responsáveis por suas próprias carreiras** e por **melhorar seu ofício, aprendendo** constantemente novas ferramentas e técnicas.
- O Software Craftsmanship tem tudo a ver com colocar **responsabilidade, profissionalismo, pragmatismo** e **orgulho** no desenvolvimento de software.

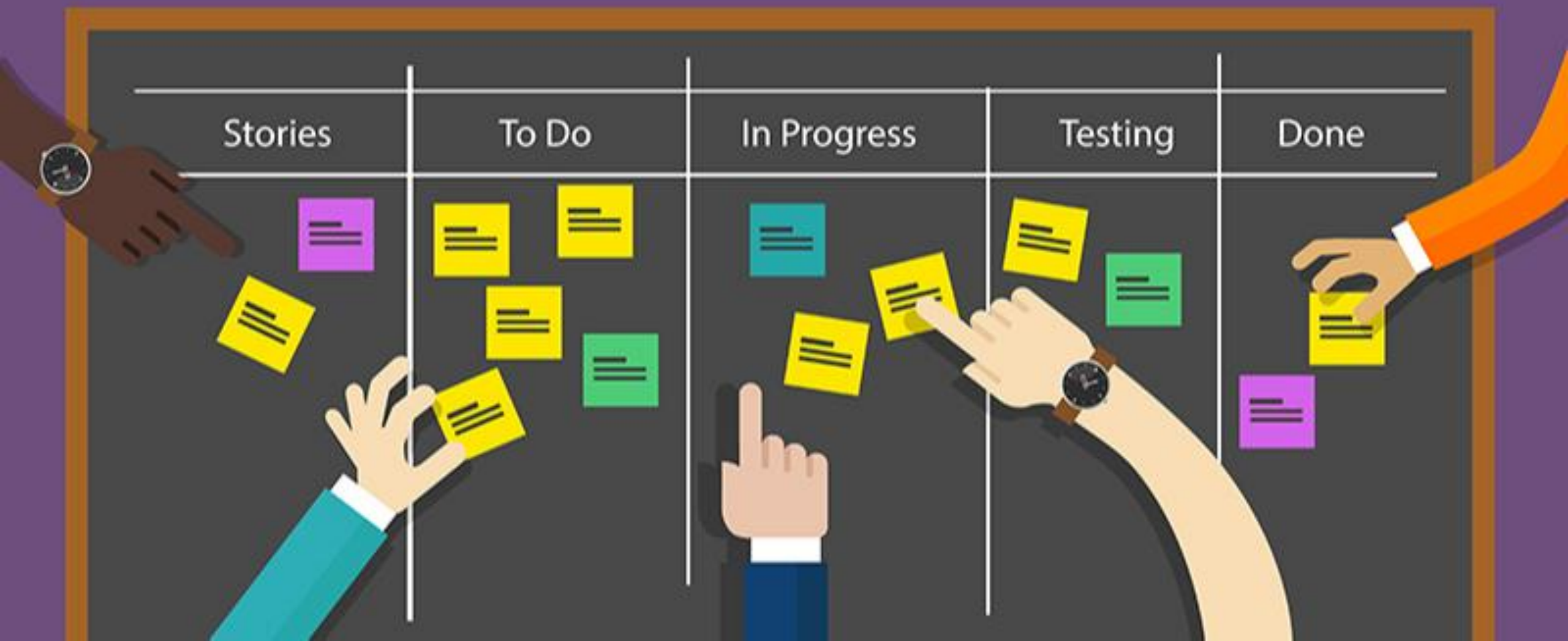
Segundo ele mesmo pontuou, o movimento do software craftsmanship seria uma “extensão” do manifesto ágil, obviamente sem anular os pontos que foram citados no manifesto ágil.

Exemplos de Métodos Ágeis

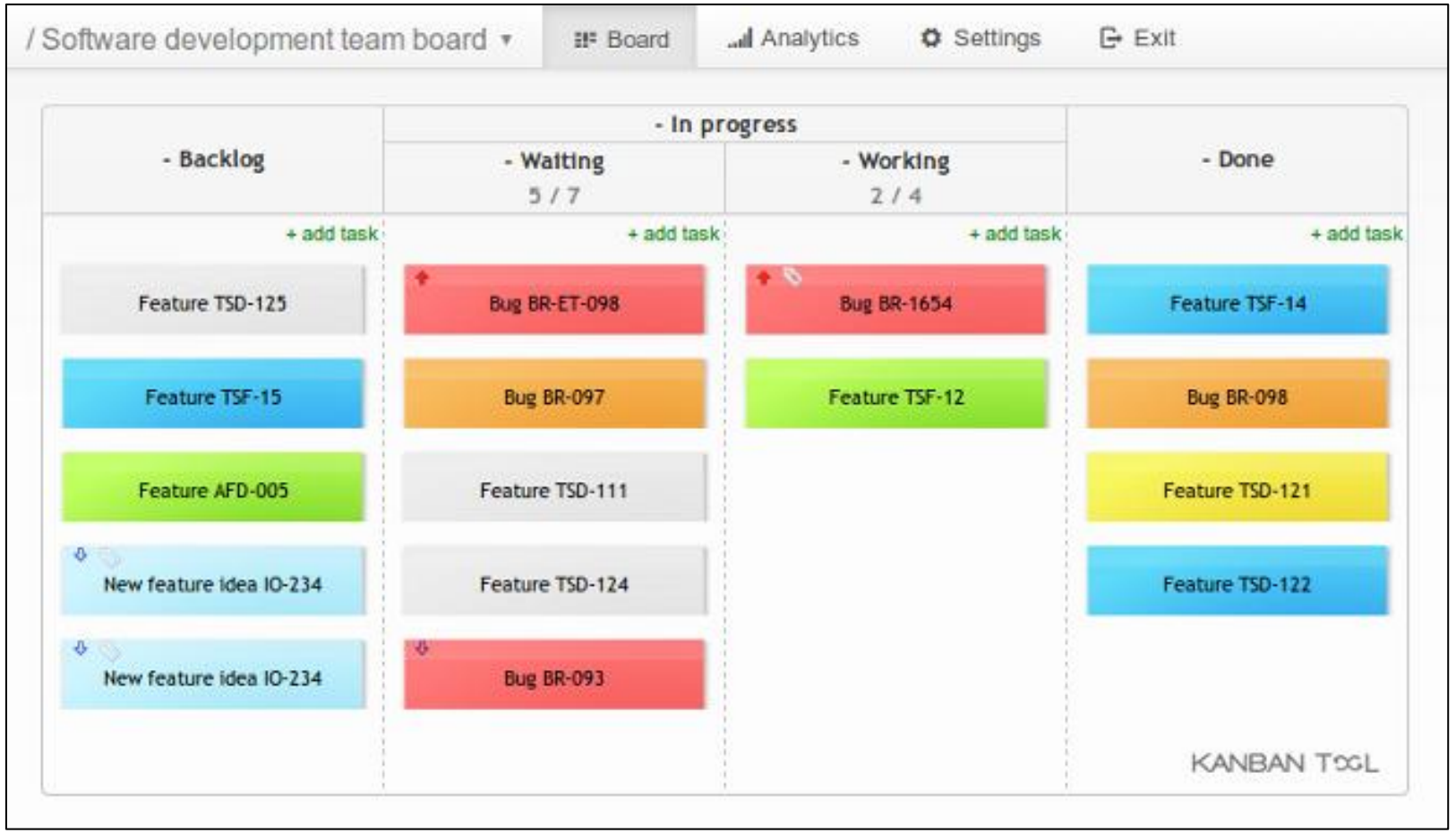
- **Extreme Programming – XP** (BECK, 1999; BECK 2000)
- **Scrum** (SCHWABER; BEEDLE, 2001)
- Feature Driven Development - FDD (PALMER; FELSING, 2002)
- Test Driven Development – TDD (BECK, 2003)
- Crystal (COCKBURN, 2001)
- Lean (WOMACK, JONES e ROSS, 1990)

- Entre (tantos) outros!

Kanban



Kanban



Se baseia em princípios ágeis, como **adaptabilidade, colaboração** e **entrega contínua de valor**, mas adota uma estrutura visual e uma filosofia de melhoria contínua **inspirada no sistema de produção da Toyota**.

Como Funciona na Prática?

1. **Configuração do Quadro Kanban:**

- O quadro Kanban é configurado com colunas que representam as etapas do processo.
- Exemplos comuns de colunas são "**A Fazer**", "**Em Progresso**", "**Em Revisão**", "**Concluído**".
- As colunas podem ser ajustadas conforme as necessidades do projeto.
- Cada cartão no quadro representa uma tarefa ou item de trabalho, com detalhes sobre o que precisa ser feito.

Como Funciona na Prática?

2. Movimentação dos Cartões:

- À medida que o trabalho avança, os cartões são movidos de uma coluna para outra no quadro.
- Se a coluna "**Em Progresso**" tiver um limite de 3 tarefas e já houver 3 cartões nela, ninguém pode começar uma nova tarefa até que uma das tarefas seja concluída e movida para a próxima coluna.

Como Funciona na Prática?

3. Revisão e Adaptação:

- A equipe revisa regularmente o quadro para **identificar bloqueios ou tarefas que estão demorando mais do que o esperado**.
- Discussões sobre como melhorar o processo, ajustar os limites de **WIP*** (Work in Progress) ou reorganizar as colunas são comuns para melhorar a eficiência e a entrega de valor.

*WIP - Quantidade de trabalho que pode estar em andamento em cada etapa do processo

SCRUM

SCRUM

O Scrum é uma metodologia ágil usada para o desenvolvimento de software, baseada em ciclos curtos e iterativos chamados **sprints** (geralmente de 1 a 4 semanas).

Pilares do SCRUM

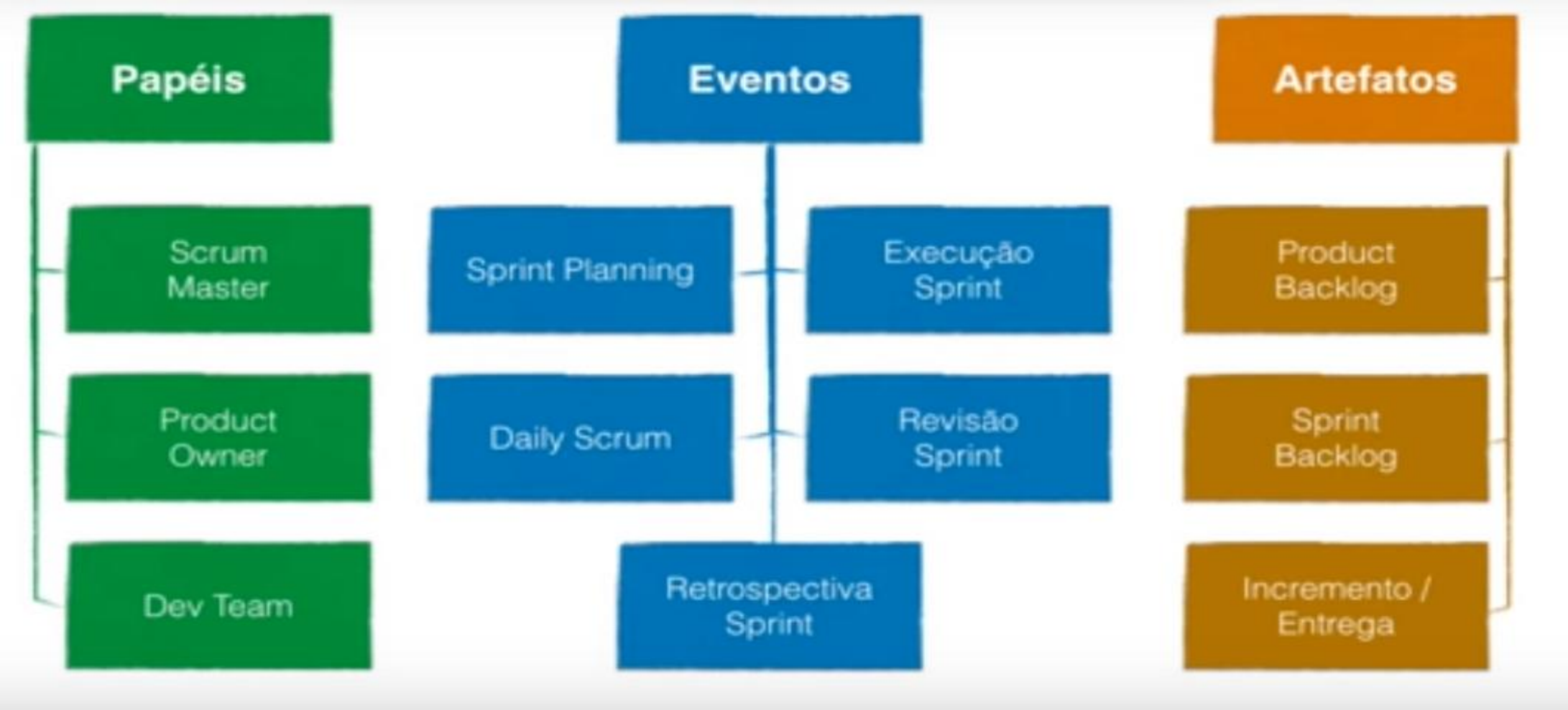
O SCRUM se sustenta em 3 pilares fundamentais:

1 – **Transparência**: dos processos, dos requisitos de entrega e do status.

2 – **Inspeção**: constante de tudo o que está sendo feito.

3 – **Adaptação**: tanto do processo, quanto do produto frente a mudanças.

Os principais elementos do SCRUM



Como Funciona?

Papéis Principais:

1. **Product Owner**: Define o que deve ser desenvolvido, prioriza as tarefas e mantém o backlog (lista de funcionalidades a serem desenvolvidas).
2. **Scrum Master**: Facilita o processo, remove impedimentos e garante que a equipe siga as práticas Scrum.
3. **Equipe de Desenvolvimento (Dev Team)**: Autogerenciada, responsável por entregar o software no final de cada sprint.

Como Funciona?

Processos (Eventos):

- **Planejamento do Sprint**: A equipe escolhe itens do backlog para desenvolver no sprint e define metas claras.
- **Daily Scrum**: Reuniões diárias de 15 minutos para sincronizar o trabalho e ajustar o plano do sprint.
- **Desenvolvimento**: A equipe trabalha para completar as tarefas selecionadas durante o sprint.
- **Revisão do Sprint**: Ao final do sprint, a equipe demonstra o que foi desenvolvido e recebe feedback.
- **Retrospectiva do Sprint**: A equipe discute o que funcionou bem e o que pode ser melhorado para os próximos sprints.

Como Funciona?

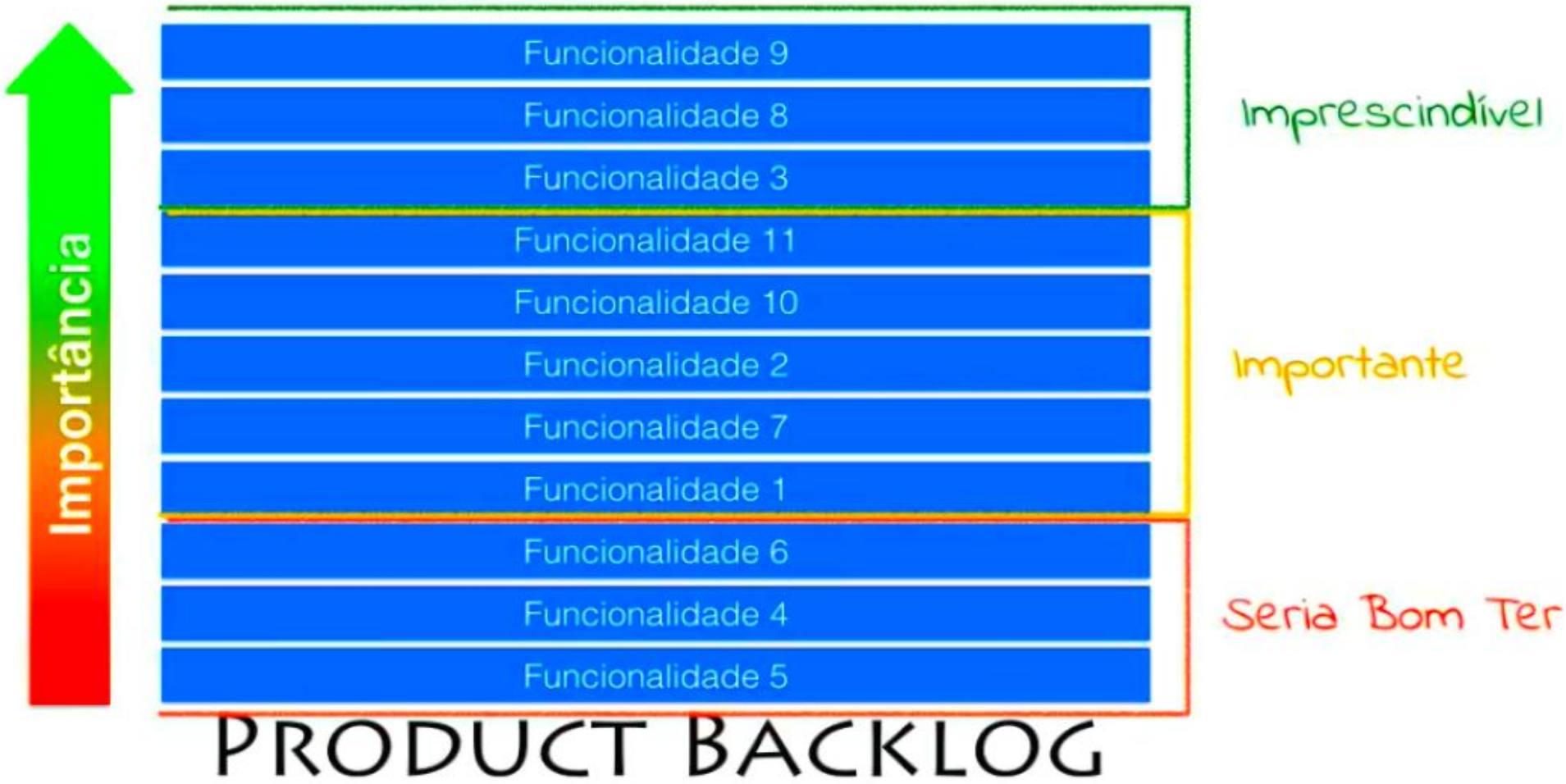
Entrega Contínua:

- Ao fim de cada sprint, o objetivo é ter uma versão funcional do software, que pode ser entregue ou melhorada no próximo ciclo.

Artefatos do SCRUM

- Tudo começa com a visão do produto. Deve-se desmembrar esta visão num conjunto de funcionalidades para criar o **Product Backlog**.
- As funcionalidades devem ser dispostas por ordem de prioridade (necessidade do cliente). Quem define essa ordem é o **Product Owner** com auxílio do **SCRUM Master**. A prioridade deve ser entendida como aquilo que mais agrega valor ao negócio.
- Podem ser definidas como, por exemplo: **imprescindível**, **importante** e "**seria bom ter**". No topo deve ficar as funcionalidades com maior importância.

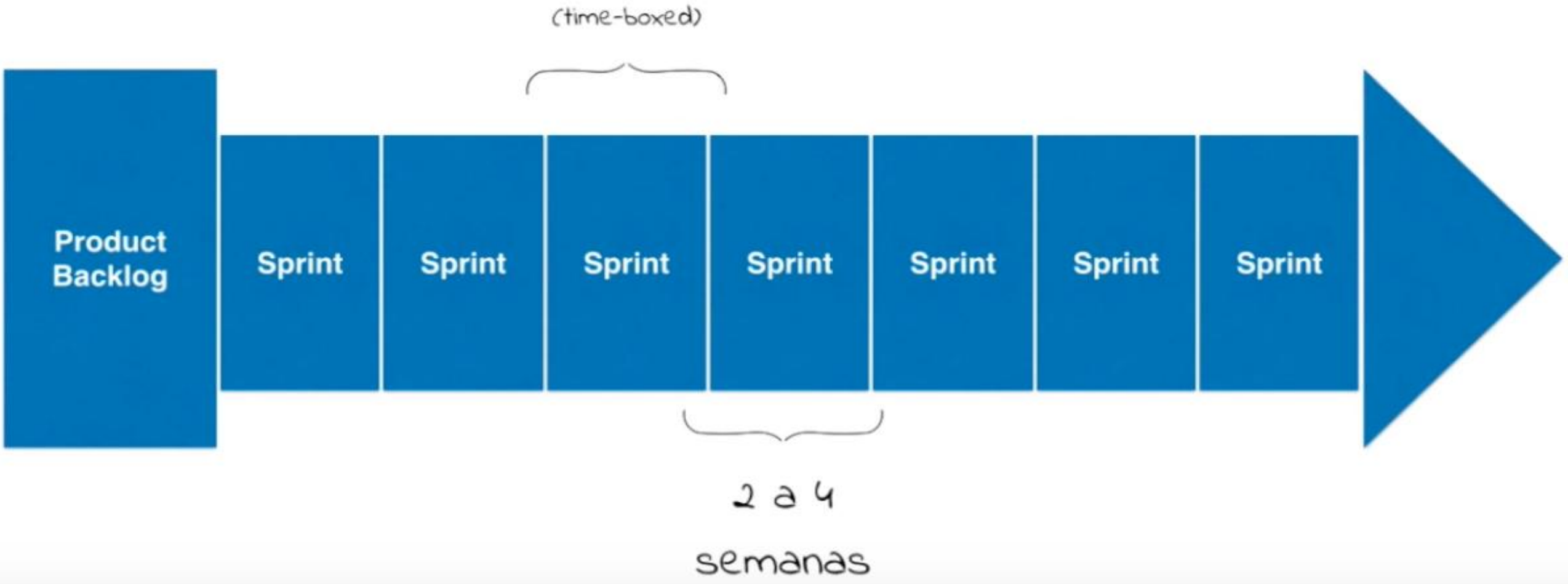
Exemplo de Product Backlog



Product Backlog e Sprints

- Não existe um software específico para gerar o **Product Backlog**. Pode ser utilizado o Excel, por exemplo. O importante é ter a lista de requisitos priorizada.
- O projeto é planejado em Sprints. Cada Sprint representa um período de tempo onde um dado conjunto de funcionalidades do Product Backlog será implementado e entregue – **incremento / entrega** (artefato).
- Para planejar cada Sprint deve-se considerar uma regra do SCRUM: **time-boxed**, o tempo fixo. O ideal é que cada Sprint tenha uma duração de tempo fixa. Normalmente, os Sprints possuem duração de 2 a 4 semanas.

Faltam 7 Sprints para concluir o projeto



Sprint Planning

- Antes de cada Sprint começar, é realizada uma reunião de planejamento chamada de **Sprint Planning** (evento). Com base na capacidade e velocidade da equipe SCRUM é definida quantas funcionalidades podem ser implementadas no tempo de um Sprint.
- Ao final de cada **Execução Sprint** (evento), espera-se que um incremento seja entregue (artefato) contendo uma parte funcional do sistema.
- No decorrer das entregas, o Product Owner pode perceber necessidades de mudanças. Elas devem ser inseridas também no Product Backlog de acordo com sua devida prioridade. Este processo é repetido até implementar todas as funcionalidades do Product Backlog.

Daily SCRUM

Todo dia é realizada uma reunião rápida de 15 minutos com todos os membros da equipe – o **Daily SCRUM** (evento). Três perguntas devem ser respondidas por todos os membros da equipe durante a reunião:

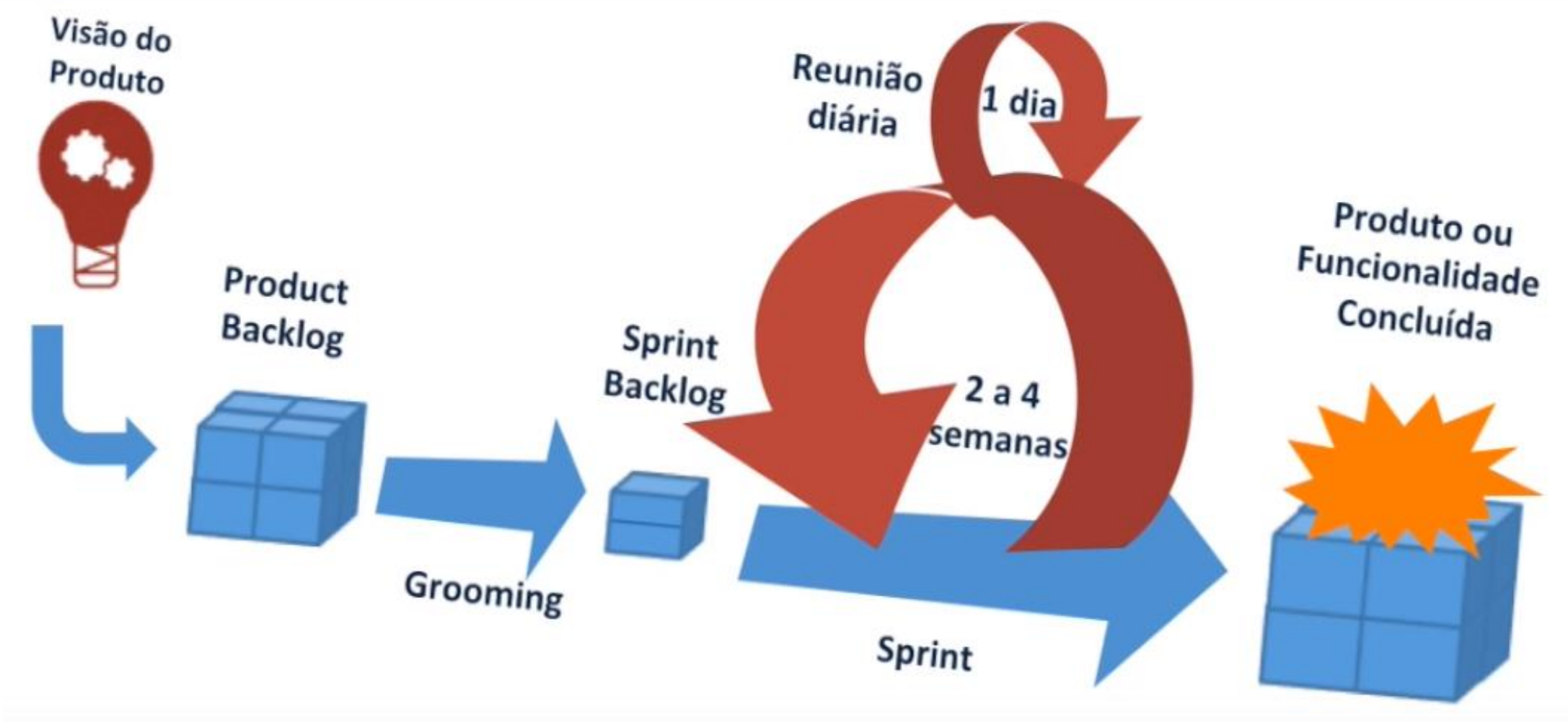
- **1 – O que fez ontem?** - Que ajudou o time a atingir a meta do Sprint;
- **2 – O que vai fazer hoje?** - Para ajudar a alcançar a meta do Sprint;
- **3 – Existe algum impedimento?** – Que não permita a mim ou ao time atingir a meta do Sprint.

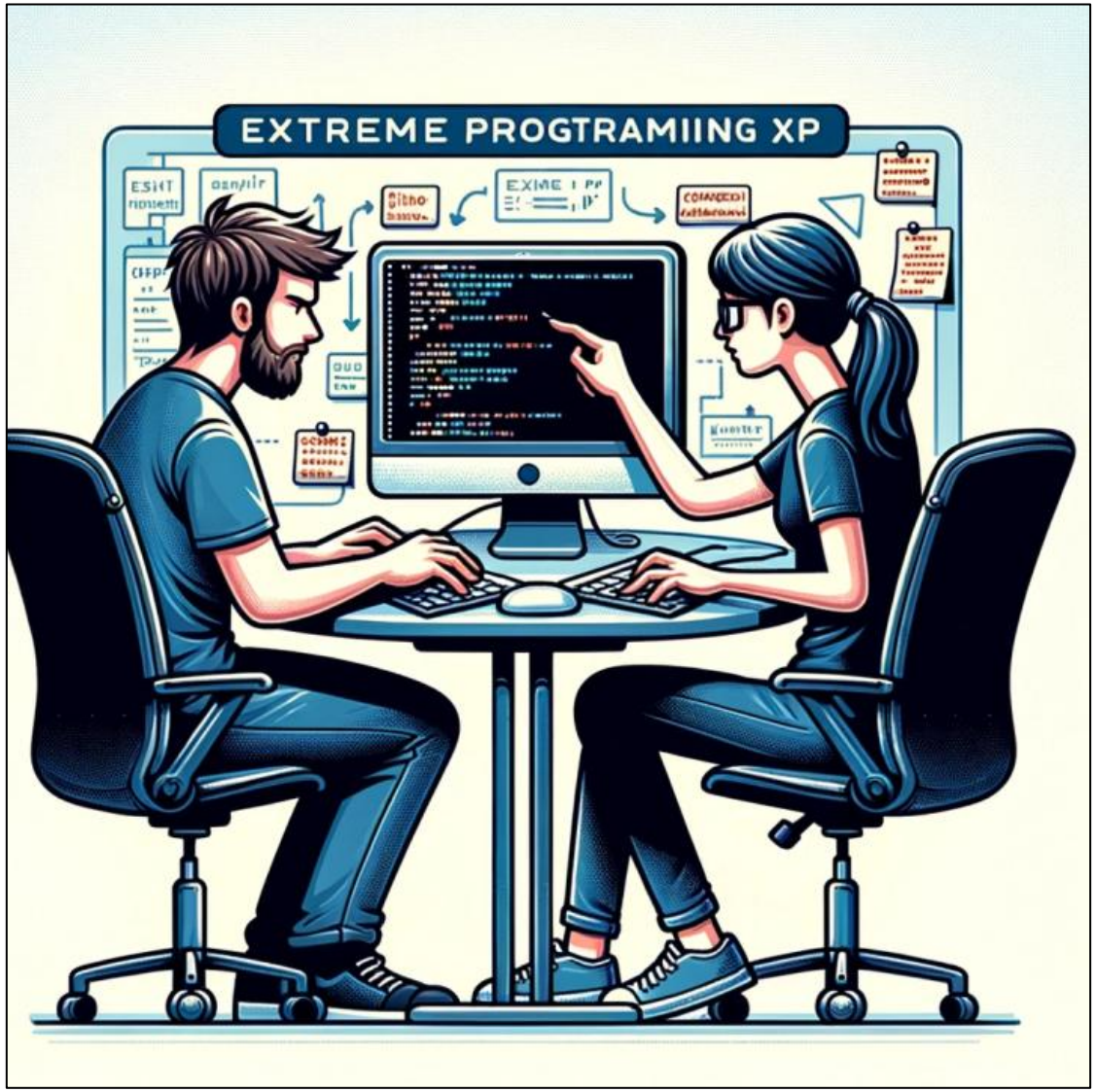
Ao responder estas 3 questões todos conseguem visualizar de uma maneira geral como está progredindo o trabalho do Sprint.

Sprint Review

- Ao final de cada Sprint, são executadas mais duas atividades. O Sprint Review (evento) que tem por objetivo **verificar se o que está sendo feito está de acordo com o esperado**. É a apresentação do que foi feito no Sprint. É aqui quer surgem as mudanças (e atualização do Product Backlog).
- Enquanto o **Sprint Review** tem como foco verificar as necessidades de mudanças do produto, a Retrospectiva Sprint (evento) tem como objetivo **verificar as necessidades de mudança do processo**. Neste momento é que são exaltados os pontos positivos e apresentados os pontos negativos durante o processo, ou seja, é identificado o que se deve melhorar e o que se deve parar de fazer.

Mais um gráfico...





Extreme Programming – XP

Características da Extreme Programming (XP)

- Caracterizada pelo desenvolvimento iterativo e o envolvimento do cliente em níveis "**extremos**".
- Requisitos são expressos como **cenários** (chamados de história do usuário) que são implementados como uma série de tarefas.
- Programadores trabalham em **pares** e desenvolvem testes para cada tarefa antes da escrita do código.
- Todos os **testes** devem ser **executados** com sucesso quando um novo código é integrado ao sistema.

Como Funciona o Extreme Programming (XP)?

1. Desenvolvimento incremental apoiado por pequenos e frequentes releases do sistema e por uma abordagem de **descrição de requisitos baseado nas histórias ou cenários do cliente** (base para planejamento do processo).
2. Envolvimento do cliente é apoiado pelo **engajamento** em tempo integral na equipe de desenvolvimento. O cliente é responsável pela definição dos testes de aceitação do sistema.
3. As pessoas (não os processos) são apoiadas ao dividirem a programação com seus **pares**. Não envolve excessivas horas de trabalho seguidas.

Como Funciona o Extreme Programming (XP)?

4. As mudanças são apoiadas por: (i) *releases* regulares, (ii) desenvolvimento **test-first** (o teste é escrito antes do código – deve-se gerar um módulo executável) e (iii) integração contínua.
5. A manutenção da simplicidade é apoiada pelo **refactoring** constante (integração contínua) para aprimorar a qualidade do código e o uso de projetos simples que **não** antecipem mudanças futuras no sistema.

Princípio Fundamental da Engenharia de Software

"Devemos projetar pensando em mudanças futuras!"

Disse a Engenharia de Software.

- **XP descarta esse princípio** ao manter cada *release* focada unicamente nos requisitos de cada tarefa.
- O software passa por constante *refactoring* (integração contínua). A equipe de programação procura por possíveis melhorias no software, implementando-as imediatamente.

Cartão de Histórias do Cliente

Baixando e Imprimindo um artigo

Primeiro, você seleciona o artigo que deseja em uma lista. Depois você precisa informar ao sistema como quer pagar pelo artigo – isso pode ser feito por meio de uma assinatura, de uma conta empresarial ou por cartão de crédito.
Após, você obtém do sistema um formulário de direitos autorais para preenchimento. Após enviar o formulário, o artigo desejado é baixado para seu computador.
Em seguida, você escolher uma impressora para imprimir uma cópia do artigo.
Você informa ao sistema que a impressão foi bem sucedida.
Se o artigo for somente para impressão, você não poderá manter uma versão em PDF, de modo que o artigo será excluído automaticamente do seu computador.

Cartões de Tarefas para Baixar Documentos

Tarefa 1 – Implementar o fluxo de trabalho principal

Tarefa 2 – Implementar catálogo e seleção de artigos

Tarefa 3 – Implementar arrecadação de pagamentos

O pagamento pode ser feito de 3 maneiras diferentes. O usuário seleciona por meio de qual maneira deseja pagar. Se o usuário tiver uma assinatura de biblioteca, ele poderá inserir a chave de assinante que deve ser verificada pelo sistema. Ele também pode inserir um número de conta da empresa. Caso este seja válido, um débito do custo do artigo será enviado para esta conta. Finalmente, ele poderá inserir um número de cartão de crédito de 16 dígitos e data de expiração para validação e, caso sejam válidos, um débito será enviado para a conta do cartão de crédito.

Teste em XP

- Processo de teste é informal.
- Cada tarefa representa uma característica discreta do sistema. Dessa forma, um teste unitário* pode ser projetado para validar esta tarefa.

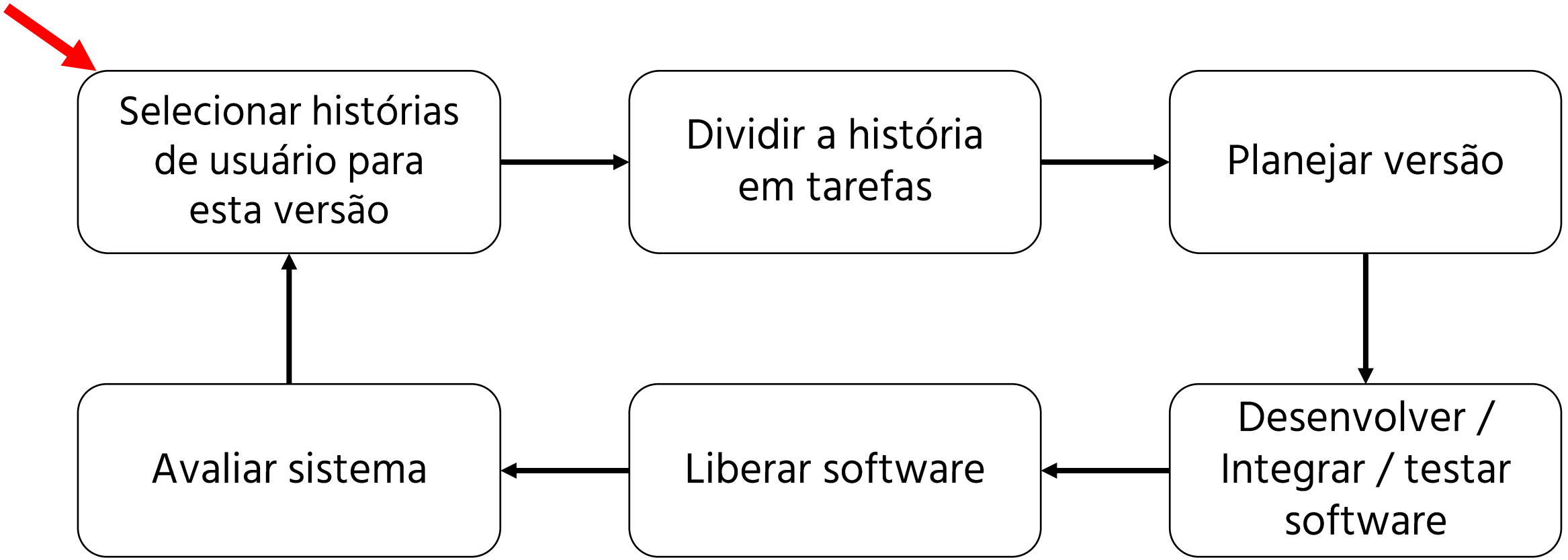
*Teste de unidade é toda a aplicação de teste nas assinaturas de entradas e saídas de um sistema, consiste em validar dados válidos e inválidos via I/O sendo aplicado por desenvolvedores ou analistas de teste.

Descrição de caso de teste para validação de cartão de crédito

Baixando e Imprimindo um artigo

Entrada:
Uma string que representa o número do cartão de crédito e dois inteiros que representam o mês e o ano de expiração do cartão.
Testes:
Verificar se todos os bytes na string são dígitos.
Verificar se o mês varia entre 1 e 12 e o ano é posterior ou igual ao ano atual.
Usando os 4 primeiros dígitos do número do cartão de crédito, verificar se o emissor do cartão é valido consultando a tabela de emissores de cartões.
Verificar a validade do cartão de crédito enviando o número do cartão e a data de expiração para o emissor de cartões.
Saída:
OK ou mensagem de erro indicando que o cartão é inválido.

Ciclo de um Release (incremento) em Extreme Programming



Pergunta
Concurso Professor IFPB – Informática – perfil 2

48. Extreme Programming (XP) é uma metodologia voltada para equipes pequenas de desenvolvedores que precisam implementar software em projetos propensos à mudanças frequentes. Sobre essa metodologia, é correto afirmar que ela recomenda:

A) Realizar reuniões "*stand up*" todos os dias.
B) Escrever testes de integração antes de implementar funcionalidades.
C) Evitar fazer integração de código frequentemente.
D) Realizar reuniões de retrospectiva ao final de cada iteração.
E) Não adotar padrões de formatação de código-fonte.

Resposta: A

Stand up meeting

https://www.desenvolvimentoagil.com.br/xp/praticas/reuniao_pe

- Stand up meeting é uma breve reunião realizada diariamente, normalmente de manhã, pela equipe de desenvolvimento com o objetivo de compartilhar informações sobre o projeto e priorizar suas atividades
- Trata-se de um diálogo entre todos os membros da equipe, se possível envolvendo também a presença do cliente.
- Em qualquer diálogo presencial, existem pelo menos duas coisas que são compartilhadas de maneira bastante rica: **informações** e **emoções**.

Stand up meeting

https://www.desenvolvimentoagil.com.br/xp/praticas/reuniao_pe

- O diálogo diário, realizado no stand up meeting, permite que cada desenvolvedor **descreva brevemente o que fez no dia anterior**, eventuais problemas que detectou, soluções interessantes que foram criadas etc.
- **Não é necessário entrar em detalhes**. A ideia é que as pessoas saibam o que está acontecendo e quem fez o que. Se um desenvolvedor se interessar bastante por um assunto a respeito do qual outra pessoa trabalhou no dia anterior, ele pode, após o **stand up meeting**, se reunir com essa pessoa e discutir a solução com maior nível de detalhe.
- Pode até resolver fazer par com aquela pessoa durante o dia de trabalho que está começando.

E a produtividade?

"Dois programadores juntos
programam a metade de
dois programadores individuais".

Verdadeiro ou Falso?

Produtividade...

E a produtividade?

- Os estudos de desenvolvimento com programação em pares mostram **o contrário** (WILLIAMS et al., 2000).
- A produtividade de desenvolvimento com a programação em pares **parece ser comparável** a duas pessoas que trabalham independentemente.
- Os pares discutem o software antes do desenvolvimento, assim, existem **poucos** inícios problemáticos e **menos retrabalho**.
- Ao final, existe uma quantidade considerável de **erros que são evitados** diminuindo assim, o retrabalho durante todo o processo de desenvolvimento.

Vantagens da Programação em Pares

Os pares são criados dinamicamente – constantemente estão mudando, assim, elencam-se algumas vantagens dessa abordagem:

1. Programação **sem egoísmo** (WEINBERG, 1971) – o código é de "propriedade" da equipe.
2. Processo **informal** de revisão.
3. Apoio ao **refactoring** – o commit na versão completa do sistema.

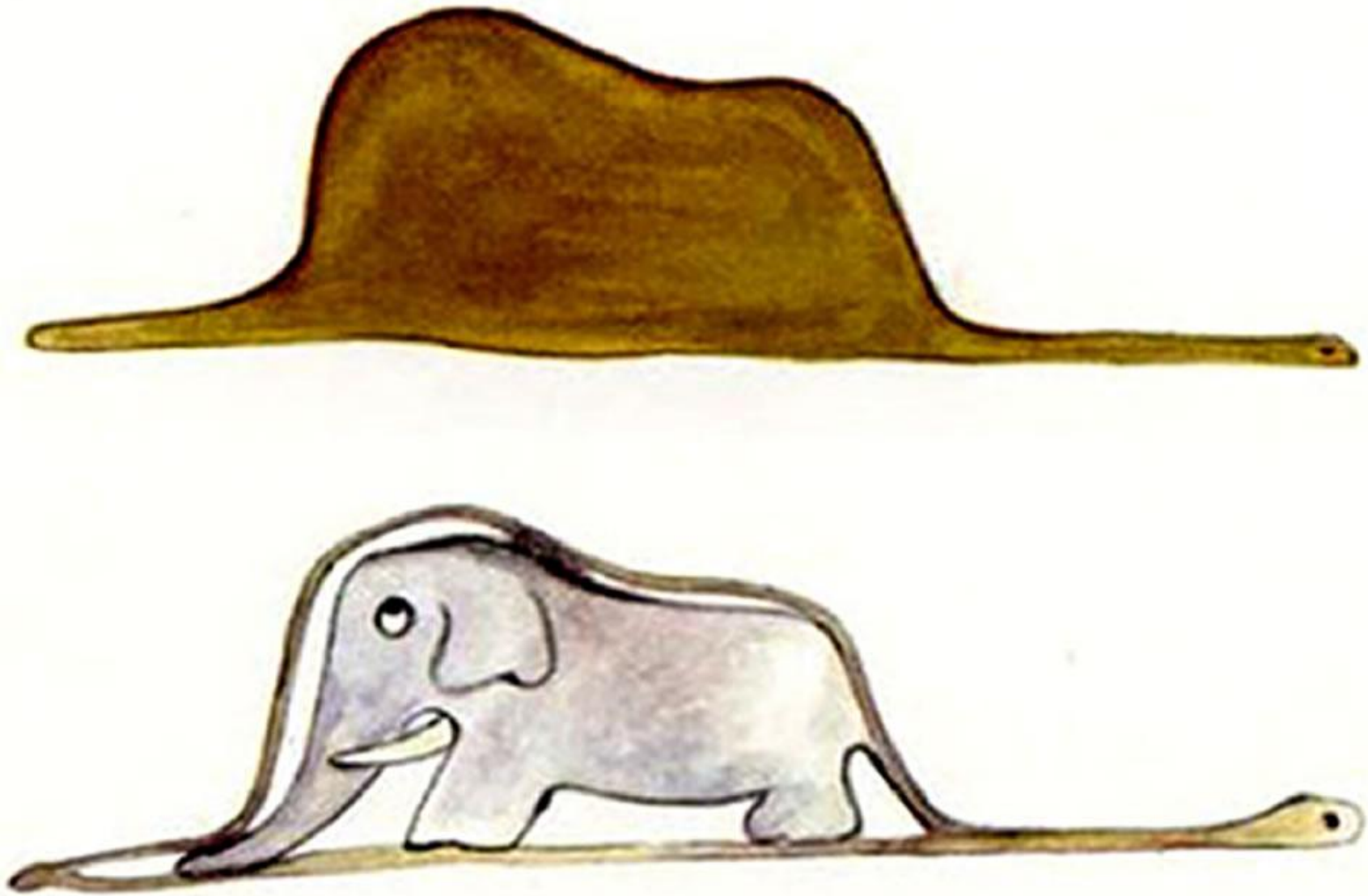
LIÇÕES
PARA
ENGENHARIA DE SOFTWARE DO
PEQUENO PRÍNCIPE

Página 10

"Mostrei minha obra-prima para as pessoas adultas e perguntei se meu desenho lhes dava medo“.

Elas responderam: “medo de um chapéu? Por quê?

Meu desenho não representava um chapéu. Representava uma jiboia digerindo um elefante. Então desenhei o interior da jiboia, para que as pessoas adultas também conseguissem entender. Elas sempre precisam de explicações.

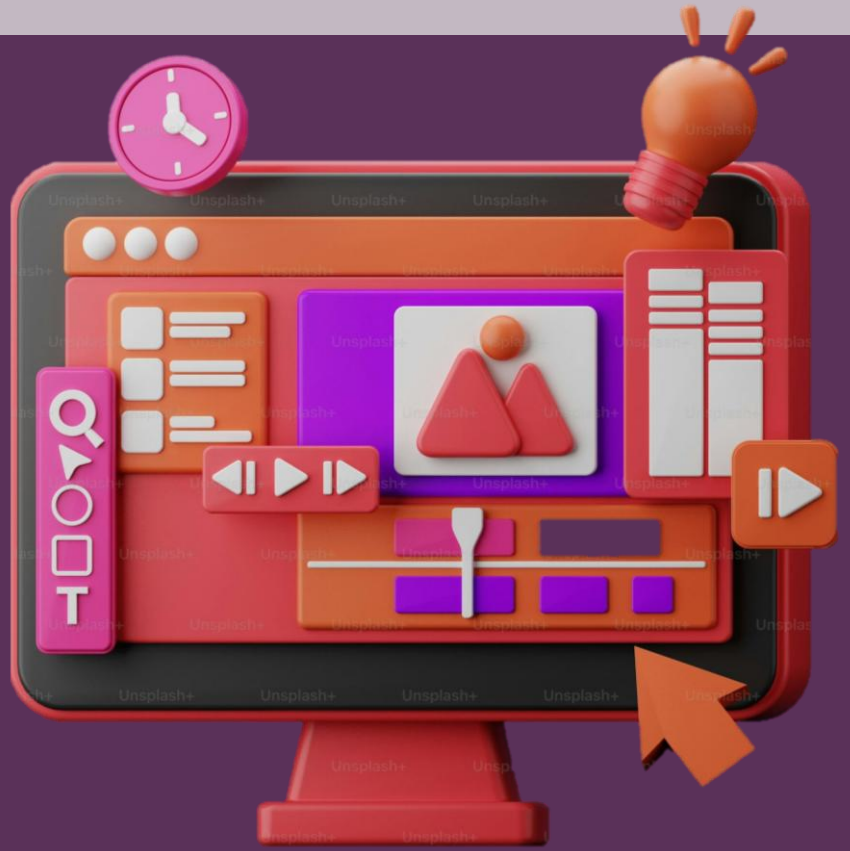


Referências

- PRESSMAN, R. **Engenharia de Software**. 7. ed. 2011 – Capítulo 5.
- SOMMERVILLE, I. **Engenharia de Software**. 9. ed. 2011 – Capítulo 4
- VAZQUEZ, C. E.; SIMÕES, G. S. **Engenharia de Requisitos**: software orientado a negócios. 2016.

ENGENHARIA DE SOFTWARE

AULA 3 – PROCESSOS DE SOFTWARE



WALTER TRAVASSOS SARINHO

@WALTEROPROFESSOR

WALTER.TRAVASSOS@SECTRAS.EDU.BR