

Engenharia de Software

**Aula: Introdução a
Engenharia de Software e
Conceitos Iniciais.**



Prof. Anderson Augusto Bosing

O que é **Software**?

Consiste em uma série de programas separados + arquivos de configuração + documentação do sistema + documentação do usuário + ,se for o caso, sites na WEB para os usuários fazerem download de informações recentes. Adaptado Sommerville (2014).

Pressman (2011) afirma que todo projeto de software nasce de uma necessidade de negócio, seja para adaptar algo já existente ou para criar um novo produto ou serviço.

O que é Engenharia de Software?

É uma disciplina da engenharia que se ocupa de todos os aspectos da produção de software, desde os estágios iniciais de especificação do sistema até a manutenção desse sistema, depois que ele entrou em operação.
Sommerville (2014).

Engenharia de Software é o estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais.
Pressman(2011)



O que é Engenharia de Software?

- Em suma, desenvolver software é uma atividade que não se confunde com escrever programas para computador.
- Desenvolvimento de software envolve procedimentos que exigem a abordagem simultânea e integra de aspectos técnicos e gerenciais.
- Se ocupa de todos os aspectos da produção do software, desde os estágios iniciais de especificação do sistema até a manutenção desse sistema.
- Trabalha com técnicas, teorias, métodos e ferramentas que auxiliam na produção do software.
- É uma área do conhecimento voltada para:
 - ✓ Especificação de software
 - ✓ Desenvolvimento de software
 - ✓ Validação de software
 - ✓ Evolução de software.

O que é Engenharia de Software?

Em resumo é uma disciplina de engenharia que se preocupa com todos os aspectos de produção de software.



Mas por que Engenharia?

Pois deve ser um processo sistêmico e regrado.



Prof. Anderson Augusto Bosing

Qual a diferença entre engenharia de software e ciência da computação?

Ciência da computação foca a teoria e os fundamentos; engenharia de software preocupa-se com o lado prático do desenvolvimento e entrega de softwares úteis.



Quais são os principais desafios da **engenharia de software**?

Lidar com o aumento de diversidade, demandas pela diminuição do tempo para entrega e desenvolvimento de software confiável.



Quais diferenças foram feitas pela Internet na engenharia de software?

A Internet tornou serviços de software disponíveis e possibilitou o desenvolvimento de sistemas altamente distribuídos baseados em serviços. O desenvolvimento de sistemas baseados em Web gerou importantes avanços nas linguagens de programação e reuso de software.

- ✓ Pagamentos por uso.
- ✓ Software as a Service(SaaS).
- ✓ Sistemas baseadas na web, em vez de sistemas locais.
- ✓ WebServices e API's possibilitam que funcionalidades sejam acessadas via internet.



Quais são os atributos de um bom software??

Um bom software deve prover a funcionalidade e o desempenho requeridos pelo usuário; além disso, deve ser confiável e fácil de manter e usar.



Quais são os atributos essenciais de um bom software?

Manutenibilidade

O software deve ser escrito de forma que possa evoluir para atender às necessidades dos clientes. Esse é um atributo crítico, porque a mudança de software é um requisito inevitável de um ambiente de negócio em mudança



Quais são os atributos essenciais de um bom software?

Confiança e proteção

A confiança do software inclui uma série de características como confiabilidade, proteção e segurança. Um software confiável não deve causar prejuízos físicos ou econômicos no caso de falha de sistema. Usuários maliciosos não devem ser capazes de acessar ou prejudicar o sistema.



Quais são os atributos essenciais de um bom software?

Eficiência

O software não deve desperdiçar os recursos do sistema, como memória e ciclos do processador. Portanto, eficiência inclui capacidade de resposta, tempo de processamento, uso de memória etc.



Quais são os atributos essenciais de um bom software?

Aceitabilidade

O software deve ser aceitável para o tipo de usuário para o qual foi projetado. Isso significa que deve ser compreensível, usável e compatível com outros sistemas usados por ele.



Engenheiro de Software

A função de um profissional como Engenheiro de Software:

- ✓ Atuar no desenvolvimento de aplicativos para dispositivos móveis, como smartphones, tablets, jogos e softwares.
- ✓ Atuar na área de Gestão que é o gerenciamento de negócios e projetos de empresas de computação e software.
- ✓ Atuar em projetos, desenvolvimento, implantação e evolução de softwares complexos, corretos, disponíveis, seguros e tolerantes a falhas e com usabilidade.
- ✓ Desenhar, especificar, programar e testar soluções que atendam às necessidades do mercado, da sociedade, das organizações e dos indivíduos, levando em conta os impactos sociais.

Engenheiro de Software

O profissional Engenheiro de Software produz um software que podemos dividir em:

Produtos genéricos - software de caixa ou de prateleira. A especificação do que o software deve fazer é de propriedade do desenvolvedor de software e as decisões sobre as mudanças de software são feitos pelo desenvolvedor.

Produto sob encomenda - quando um cliente solicita a uma empresa de software que desenvolva um software específico às necessidades da empresa.

A especificação do que o software deve fazer é propriedade do cliente para o software e eles tomam decisões sobre as mudanças necessárias no software.



Tipos de Aplicações de Software

Aplicações stand-alone.

Essas são as aplicações executadas em um computador local, como um PC. Elas contêm toda a funcionalidade necessária e não precisam estar conectadas a uma rede. Exemplos de tais aplicações são aplicativos de escritório em um PC, programas CAD, software de manipulação de fotos etc.

Aplicações interativas baseadas em transações

São aplicações executadas em um computador remoto e são acessadas pelos usuários a partir dos seus próprios PCs ou terminais. Essas incluem aplicações web tais como para e-commerce.



Tipos de Aplicações de Software

Sistemas de controle embutidos.

São sistemas de software de controle que controlam e gerenciam dispositivos de hardware. Numericamente, provavelmente existem mais sistemas embutidos do que qualquer outro tipo de sistema.

Sistemas de processamento de lotes

São sistemas corporativos projetados para processar dados em grandes lotes. Eles processam grande número de entradas individuais para criar as saídas correspondentes. Exemplos de sistemas de lotes incluem sistemas periódicos de cobrança, como sistemas de cobrança telefônica, e sistemas de pagamentos de salário.

Tipos de Aplicações de Software

Sistemas de entretenimento.

São sistemas cuja utilização principal é pessoal e cujo objetivo é entreter o usuário. A maioria desses sistemas é de jogos de diferentes tipos. A qualidade de interação com o usuário é a característica particular mais importante dos sistemas de entretenimento.

Sistemas para modelagem e simulação.

São sistemas que incluem vários objetos separados que interagem entre si, desenvolvidos por cientistas e engenheiros para modelar processos ou situações físicas. Esses sistemas geralmente fazem uso intensivo de recursos computacionais e requerem sistemas paralelos de alto desempenho para executar.

Tipos de Aplicações de Software

Sistemas de coleta de dados.

São sistemas que coletam dados de seu ambiente com um conjunto de sensores e enviam esses dados para outros sistemas para processamento. O software precisa interagir com sensores e frequentemente é instalado em um ambiente hostil, por exemplo, dentro de uma máquina ou em um lugar Remoto.

Sistemas de sistemas.

São sistemas compostos de uma série de outros sistemas de software. Alguns deles podem ser produtos genéricos de software, como um programa de planilha eletrônica. Outros sistemas do conjunto podem ser escritos especialmente para esse ambiente.



PONTOS IMPORTANTES

- ❑ A engenharia de software é uma disciplina de engenharia que se preocupa com todos os aspectos de produção de software.
- ❑ Atributos essenciais do produto de software são manutenibilidade, confiança, proteção, eficiência e aceitabilidade.
- ❑ Existem muitos tipos diferentes de sistemas e cada um requer ferramentas de engenharia de software e técnicas apropriadas para o seu desenvolvimento.

Engenharia de Software

**Aula: Conceitos de
Sistemas da Informação,
Mitos da Engenharia de
Software.**



Prof. Anderson Augusto Bosing

Mitos da Engenharia de Software

Mito do Gerenciamento

Mito 1. "Se a equipe dispõe de um manual repleto de padrões e procedimentos de desenvolvimento de software, então a equipe está apta a encaminhar bem o desenvolvimento. A equipe já não tem tudo o que ela precisa saber?"

Realidade: o manual é útil, contudo, muitas vezes, fica defasado rapidamente, não é tão completo quanto deveria, não é adaptável a todos os projetos ou, ainda, não busca melhorar o tempo de entrega com foco na qualidade, sendo assim, o manual apenas não é o suficiente.



Prof. Anderson Augusto Bosing

Mitos da Engenharia de Software

Mito do Gerenciamento

Mito 2. "Se o desenvolvimento do software estiver atrasado, basta aumentar a equipe para honrar o prazo de desenvolvimento".

Realidade: Existe um custo em tempo para que um desenvolvedor passe a render, e os desenvolvedores que já estão no projeto deverão parar para auxiliar os novos colaboradores. Certas coisas exigem tempo, porém, quando bem planejadas, a inclusão de novos desenvolvedores ajudará.



Prof. Anderson Augusto Bosing

Mitos da Engenharia de Software

Mito do Cliente

Mito 3. “Uma descrição breve e geral dos requisitos do software é o suficiente para iniciar o seu projeto... maiores detalhes podem ser definidos posteriormente”.

Realidade: quanto maior a definição dos requisitos, mais correto será o desenvolvimento e menor será o retrabalho.



Prof. Anderson Augusto Bosing

Mitos da Engenharia de Software

Mito do Desenvolvedor

Mito 4. “Após a edição do programa e a sua colocação em funcionamento, o trabalho está terminado.”

Realidade: O que ocorre na realidade é completamente diferente disto. Segundo dados obtidos que 50 a 70% do esforço de desenvolvimento de um software é despendido após a sua entrega ao cliente (manutenção).



Prof. Anderson Augusto Bosing

Mitos da Engenharia de Software

Mito do Desenvolvedor

Mito 5. “A equipe de desenvolvimento entende que, enquanto o software não estiver em uso, não é possível validar sua qualidade.”

Realidade: Inúmeros tipos de testes podem e devem ser feitos durante o desenvolvimento, visando encontrar a menor quantidade possível de erros durante a fase de uso.



Prof. Anderson Augusto Bosing

Mitos da Engenharia de Software

Mito do Desenvolvedor

Mito 6. “A engenharia de software obriga a equipe de desenvolvimento a entregar uma documentação volumosa e trabalhosa de ser criada.”

Realidade: A engenharia de software visa à entrega de um produto de qualidade, e não à entrega de uma documentação volumosa.



Prof. Anderson Augusto Bosing

O que é Sistema da Informação?

Sistema de Informação, sigla S.I., é um conjunto de componentes inter-relacionados (pessoas, hardware, software, redes de comunicações e recursos de dados) que coletam (ou recuperam), processam, armazenam e distribuem informações destinadas a apoiar a tomada de decisões, a coordenação e o controle de uma organização. (LAUDON; LAUDON, 2004, p. 7).



Conceitos de Informação

DADOS são fatos que ainda não foram trabalhados: Nome de Empregado e Registro de Identidade.

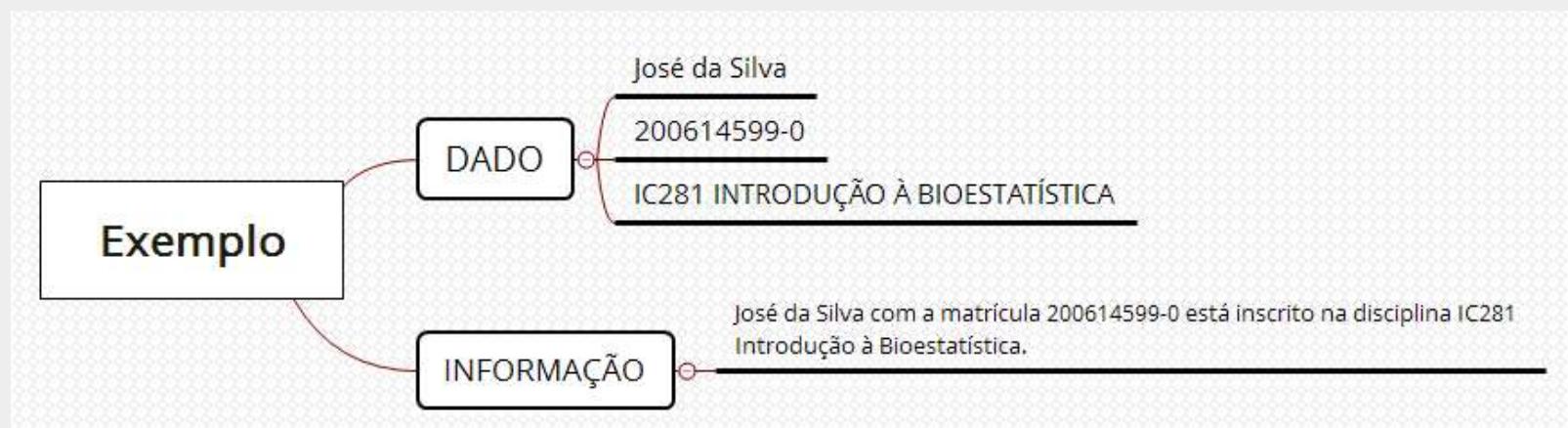


INFORMAÇÃO são dados organizados ou ordenados de forma significativa de modo que adquirem um valor adicional. Ex: Total de horas trabalhadas em uma semana, Estoque médio mensal, etc.

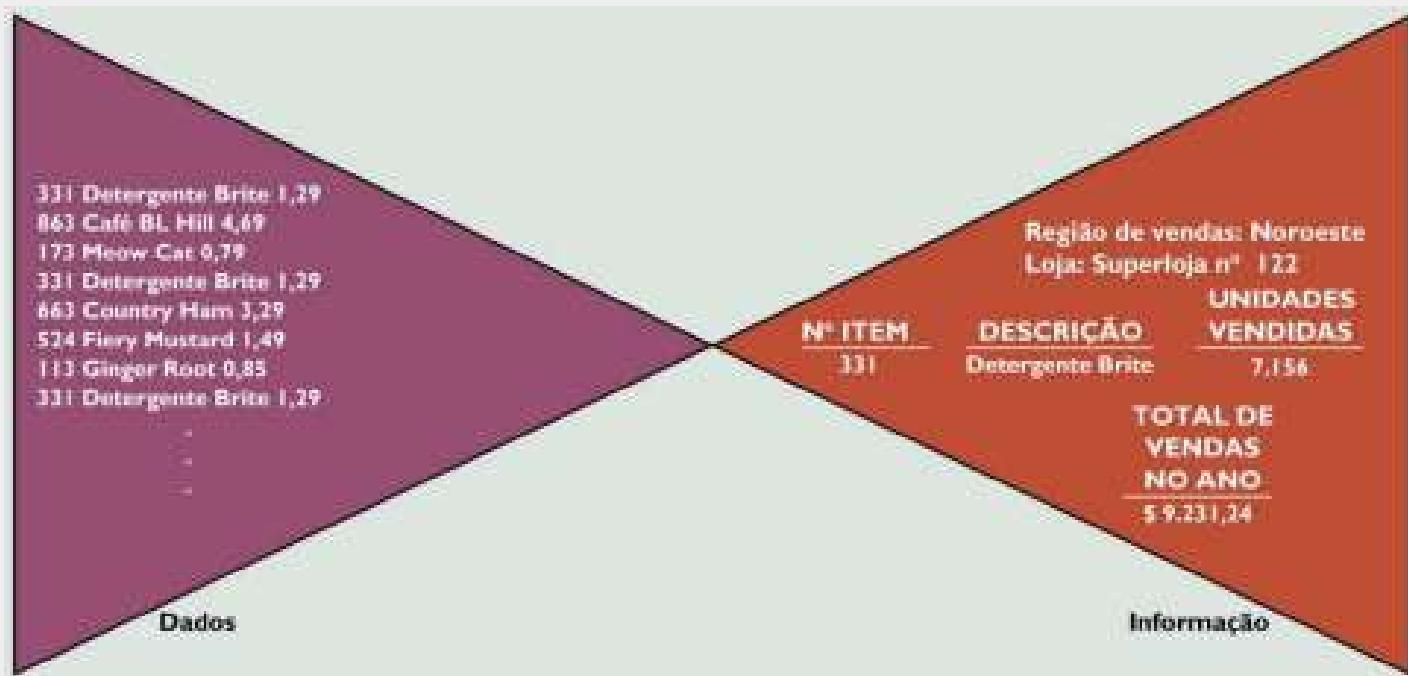
Conceitos de Informação

A transformação de dados em informação é tarefa principal dos sistemas de informação.

A transformação de dados em informação é um processo ou uma série de tarefas logicamente relacionadas, executadas para atingir um resultado definido



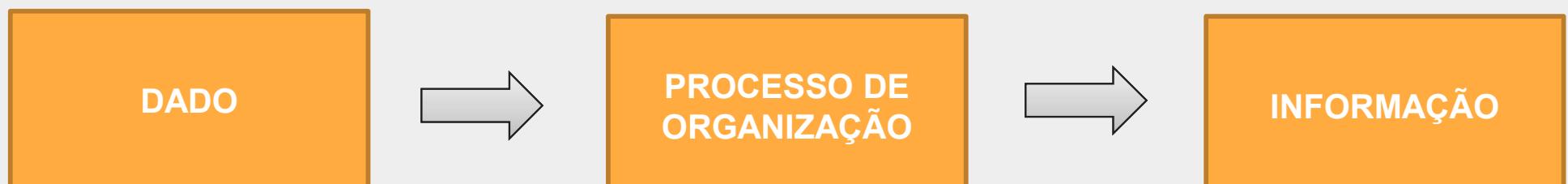
Conceitos de Informação



Conceitos de Informação

O processo de definição das relações entre dados requer conhecimento.

- Conhecimento são as regras, diretrizes, procedimentos usados para selecionar, organizar e manipular os dados, com a finalidade de torná-los úteis para uma tarefa específica.



Através da informação as pessoas adquirem conhecimento.

Conceitos de Informação

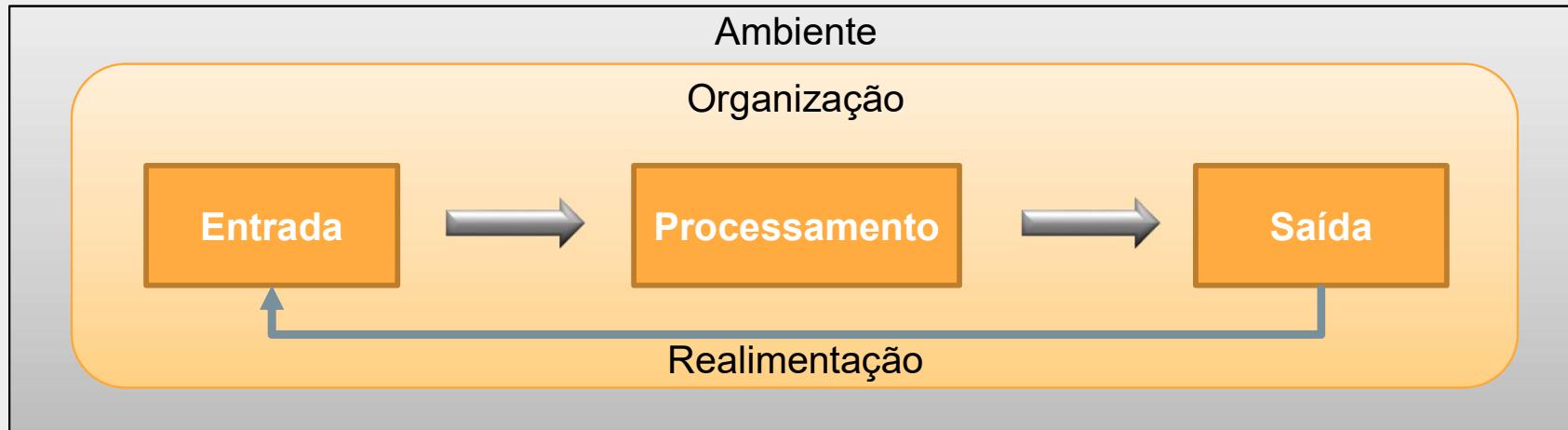


Entrada: envolve captação e reunião de elementos que entram no sistema para serem processados. Ex. matériaprima, energia, dados e esforço humano.

Processamento: envolve processos de transformação que converte insumo (entrada) em produto. Ex. controle de estoque, respiração humana ou cálculos matemáticos.

Saída: envolve a transferência de elementos produzidos por um processo de transformação até o seu destino final. Ex. produtos acabados, serviços, informações gerenciais.

Conceitos de Informação



Ambiente: é o conjunto de elementos que não pertencem ao sistema, mas qualquer alteração no sistema pode mudar ou alterar os seus elementos e qualquer alteração nos seus elementos pode mudar ou alterar o sistema.

Realimentação: pode ser considerado como a reintrodução de uma saída sob forma de informação. Ou ainda, é uma saída usada para fazer ajustes ou modificações nas atividades de entrada ou processamento.

Exemplo: Folha de Pagamento

Entrada:

- As horas trabalhadas pelos funcionários devem ser levantadas (**captadas**) através dos cartões de horas do funcionário.

Processamento:

- As horas trabalhadas de cada funcionário devem ser convertidas em pagamento líquido.
- Multiplicação das horas trabalhadas pela taxa de pagamento por hora do empregado, para se obter o pagamento bruto.
- Se o pagamento de horas semanais trabalhadas superar 40 horas, o pagamento de horas extras também pode ser determinado.
- Então as deduções são subtraídas do pagamento bruto para se obter o pagamento líquido.

Saída:

- cheques de pagamentos a empregados, relatórios para gerentes e informações fornecidas para acionistas, bancos, agências governamentais e outros grupos.



VAMOS PRATICAR ?

Desenvolvendo um SI para uma ONG

- Você está em contato com o coordenador de uma Organização Não Governamental chamada Parceiros Voluntários;
- A Idéia desta ONG é cadastrar profissionais voluntários (parceiros) e também instituições que necessitam dos serviços destes profissionais;
- Dessa forma, os profissionais cadastrados informariam sua especialidade, tempo e horário disponível, e as instituições a serem favorecidas pelos trabalhos voluntários, repassariam as informações necessárias para o encaminhamento de tais profissionais;
- O diretor da organização solicita o trabalho de sua equipe para desenvolver um Sistema de Informação para a ONG;
- Primeiramente descreva este sistema em termos de ENTRADAS – PROCESSAMENTOS e SAÍDAS, em seguida, identifique quais dados precisariam ser armazenados pelo sistema.



Prof. Anderson Augusto Bosing

Engenharia de Software

**Aula: Introdução a
Modelagem e Valor da
Informação no
Desenvolvimento de
Sistemas**



Prof. Anderson Augusto Bosing

Qual é o valor da informação para uma empresa?



Qual é o valor da informação para uma empresa?

Está surgindo um novo tipo de bem econômico: a **informação**.

A empresa que dispõe de mais informações sobre seu processo de negócio está em **vantagem**.

Em consequência, surgiu a necessidade de gerenciar informações de uma forma **adequada e eficiente**.

Qual é o valor da informação para uma empresa?

O **objetivo** principal e final da construção de um sistema de informações é a adição de valor à empresa.

Isso implica que a produtividade nos processos da empresa deve aumentar de modo significativo.

O sistema deve ser economicamente justificável.

Qual é o valor da informação para uma empresa?

- <https://adssettings.google.com/authenticated>



Qual é o valor da informação para uma empresa?

Uma característica é a complexidade de seu desenvolvimento, que aumenta à medida que o tamanho do sistema cresce.



Qual é o valor da informação para uma empresa?

Para a construção de sistemas de software mais complexos, é necessário um planejamento inicial. Equivalente ao projeto das plantas da engenharia civil também deve ser realizado. Essa necessidade leva ao conceito de modelo, tão importante no desenvolvimento de sistemas.

Gerenciamento de Complexidade

Pode haver diversos modelos de um mesmo sistema, cada qual descrevendo uma perspectiva.

Exemplo: Um Carro pode ter um modelo para representar sua parte elétrica, outro para a parte aerodinâmica etc.



Gerenciamento de Complexidade

Por meio de modelos podemos fazer estudos e prever comportamentos do sistema em desenvolvimento.

Modelos revelam as características essenciais de um sistema. Detalhes não relevantes e que só aumentariam a complexidade do problema podem ser ignorados.

Comunicação entre as Pessoas Envolvidas

O desenvolvimento de um sistema envolve a execução de uma grande quantidade de atividades.

Os modelos de um sistema servem para promover a difusão de informações relativas ao sistema entre os indivíduos envolvidos em sua construção.



Redução dos Custos no Desenvolvimento

Seres humanos estão sujeitos a cometer erros.

Certamente a correção desses erros é menos custosa quando detectada é realizada ainda nos modelos do sistema.

O que é melhor?

Corrigir um erro na projeto de um carro ou recolher todos os carros depois de fabricados.



Previsão do Comportamento Futuro do Sistema

O comportamento do sistema pode ser discutido mediante uma análise dos seus modelos.

Os modelos servem como um “laboratório”, em que diferentes soluções para um problema relacionado à construção do sistema podem ser experimentadas.



Engenharia de Software

**Aula: Processo e Ciclo de
Vida de software.**



Prof. Anderson Augusto Bosing

Processo de software

Sommerville (2014) afirma que um “processo de software é um conjunto de atividades relacionadas que levam à produção de um produto de software”.

Pressman (2011) complementa dizendo que “processo é uma metodologia para as atividades, ações e tarefas necessárias para desenvolver um software de alta qualidade”.

Processo de software

- É uma série de passos (um ROTEIRO).
- Para criar EM TEMPO um SOFTWARE de ALTA QUALIDADE, sem estourar o ORÇAMENTO.
- Como “escolher” um processo?
- ✓ As CARACTERÍSTICAS DA APLICAÇÃO (domínio do problema, tamanho, complexidade etc);
- ✓ A TECNOLOGIA a ser adotada na sua construção (paradigma de desenvolvimento, linguagem de programação, mecanismo de persistência etc), a organização;
- ✓ ONDE o produto será desenvolvido;
- ✓ O PERFIL DA EQUIPE de desenvolvimento.



Engenharia de Software - Ciclo de vida

- Independente da metodologia que será utilizada para desenvolvimento do software sempre existirá um ciclo de vida existente em todas as metodologias.
- Ciclo de vida pode ser definido como uma “estrutura contendo processos, atividades e tarefas envolvidas no desenvolvimento, operação e manutenção de um produto de software, abrangendo a vida do sistema desde a definição de seus requisitos até o término de seu uso”.

Engenharia de Software - Ciclo de vida

- Pesquisem e façam um resumo de cada uma das etapas do ciclo de vida de software:
 - Contemplem: Principais atividades realizadas, Profissionais de tecnologia envolvidos, Produtos ou artefatos gerados como resultado da etapa. Relação dessa etapa com as demais (como ela influencia ou depende das outras).
- ✓ Planejamento
 - ✓ Análise e Especificação de Requisitos
 - ✓ Projeto
 - ✓ Implementação
 - ✓ Testes
 - ✓ Entrega e Implantação
 - ✓ Operação
 - ✓ Manutenção



Ciclo de vida - Planejamento

- Fornece uma estrutura que possibilita ao gerente fazer estimativas iniciais de recursos, custos e prazos;
- O escopo do software é estabelecido;
- Um plano de projeto deve ser elaborado configurando o processo a ser utilizado;
- Esta atividade faz parte da **gerência de projeto**.

Ciclo de vida - Análise e Especificação de Requisitos

- O escopo do software é refinado;
- Nessa fase, a interação entre quem desenvolverá e o cliente é muito grande, contudo não se discute como será feito o software, mas sim o que ele deve fazer.
- Devem ser analisados o domínio do problema e o domínio da solução.
- São usadas as histórias de usuário (User Stories), que são pequenas histórias escritas para auxiliar na definição do requisito entre quem desenvolve e o cliente.
- Todo esse material é compilado e gera um relatório que servirá para os tomadores de decisão definirem, ou não, a continuidade e o desenvolvimento do software.



Ciclo de vida – Projeto

- Utiliza a fase anterior como insumo.
- Essa fase é voltada ao programador do software.
- Definição de como seria a interface que o usuário opera, quais cores seriam utilizadas na interface, como seria o fluxo de funcionamento de cada botão existente na interface, em qual banco de dados ficariam os dados gerados.
- O projeto, diferente da fase anterior, define como as coisas acontecerão.



Ciclo de vida – Implementação

- Essa fase deve traduzir o projeto em um software, utilizando ferramentas e linguagens adequadas.
- Dadas as inúmeras metodologias de desenvolvimento, cada programador pode desenvolver o código do sistema de uma forma diferente.



Ciclo de vida – Testes

- **Teste de unidade:** cada componente é testado individualmente, sem conexão com outros componentes.
 - ✓ Exemplo: testar apenas o método que faz a soma dos valores das vendas, dados dois números, ele deve retornar a soma dos dois.
- **Teste de módulo:** um módulo é um agrupamento de pequenos componentes, que tem uma função específica.
 - ✓ Exemplo: testar a geração do relatório de vendas de um produto.
- **Teste de subsistemas:** são testados módulos integrados que controlam as interfaces do sistema.
 - ✓ Exemplo: testar as interfaces do sistema de compra e venda.

Ciclo de vida – Testes

- **Teste de sistema:** testa a integração dos subsistemas que formam o sistema principal.
 - ✓ Exemplo: realizar login, operações e geração de relatórios em um sistema.

- **Teste de aceitação:** testes realizados com dados reais fornecidos pelos clientes. Último teste antes de colocar o sistema em operação.
 - ✓ Exemplo: quando há muitos usuários na base e o login não acontece de forma instantânea.



Ciclo de vida – Entrega e Implantação

- O software deve ser instalado em ambiente produção.
- Envolve:
 - ✓ Treinamento de usuários;
 - ✓ Configuração do ambiente de produção;
 - ✓ Conversão bases de dados (se necessário).



Ciclo de vida – Operação

Após os testes, entrega e implantação, o software passa a ser utilizado de fato em um ambiente de produção.



Ciclo de vida – Manutenção

➤ Manutenção corretiva:

✓ Correção dos erros encontrados pelo cliente e que não foram detectados nas fases de testes anteriores.

➤ Manutenção adaptativa:

✓ Adaptação do software relacionado as mudanças do ambiente externo.

➤ Manutenção evolutiva:

✓ Mudanças não previstas nos requisitos originais, visando a melhorias de desempenho e novas funcionalidades.

➤ Manutenção preventiva:

✓ A iniciativa parte da equipe de desenvolvedores, visando evitar futuros problemas e melhorias em futuras manutenções.



Engenharia de Software - Ciclo de vida

➤ Em geral, um ciclo de vida envolve as etapas:

- ✓ Planejamento
- ✓ Análise e Especificação de Requisitos
- ✓ Projeto
- ✓ Implementação
- ✓ Testes
- ✓ Entrega e Implantação
- ✓ Operação
- ✓ Manutenção



Engenharia de Software

**Aula: Metodologias
Tradicionais de
Desenvolvimento de
Software.**



Prof. Anderson Augusto Bosing

O desenvolvimento de um software é uma tarefa **simples**?



O desenvolvimento de um software é uma tarefa **simples**?

O desenvolvimento de software é uma atividade complexa. Essa complexidade corresponde ao desenvolvimento dos seus diversos componentes: software, hardware, procedimentos etc.

Isso se reflete no alto número de projetos de software que não chegam ao fim, ou que extrapolam recursos de tempo e dinheiro alocados.

O desenvolvimento de um software é uma tarefa simples?

Chaos Report, um estudo clássico feito pelo Standish Group sobre projetos de desenvolvimento:

- Porcentagem de projetos que terminam dentro do prazo estimado: 10%.
- Porcentagem de projetos que são descontinuados antes de chegar ao fim: 25%.
- Porcentagem de projetos acima do custo esperado: 60%.
- Atraso médio nos projetos: um ano.

O desenvolvimento de um software é uma tarefa simples?

Lidar com a complexidade e minimizar os problemas envolvidos no desenvolvimento de software envolvem a definição de processos de desenvolvimento de software.

Um processo de desenvolvimento de software compreende todas as atividades necessárias para definir, desenvolver, testar e manter um produto de software.

Metodologias de Desenvolvimento de Software

- Conforme a necessidade de desenvolvimento de sistemas foi crescendo, as atividades realizadas pelos responsáveis no desenvolvimento se repetiam a cada novo software.
- A partir do estudo dessas atividades, foram surgindo metodologias e processos, visando padronizar e agilizar o tempo do projeto.



Metodologias de Desenvolvimento de Software

- 1. Modelo em Cascata ou Sequencial Linear;**
- 2. Modelo de Prototipação;**
- 3. Modelo RAD;**
- 4. Modelos Evolucionários:**
 - 4.1. Incremental;**
 - 4.2. Espiral;**
 - 4.3. Montagem de componente;**
 - 4.4. Desenvolvimento concorrente.**
 - 4.5. Modelo de Métodos Formais;**
 - 4.6. Técnicas de 4GT.**



Modelo em Cascata ou Sequencial Linear

- Baseado em projetos de engenharia clássicos ou ciclo da engenharia convencional, foi consolidado em 1970 por Royce;
- Organiza o processo em uma sequência linear de fases.
- É utilizado quando os requisitos são muito bem definidos.
- O modelo de ciclo de vida em cascata foi o primeiro modelo a ser conhecido em engenharia de software e está na base de muitos ciclos de vida utilizados hoje em dia. Este consiste basicamente num modelo linear em que cada passo deve ser completado antes que o próximo passo possa ser iniciado.
- O modelo em cascata é um exemplo de um processo dirigido a planos, isto é, as atividades devem ser programadas e planejadas antes de serem iniciadas.

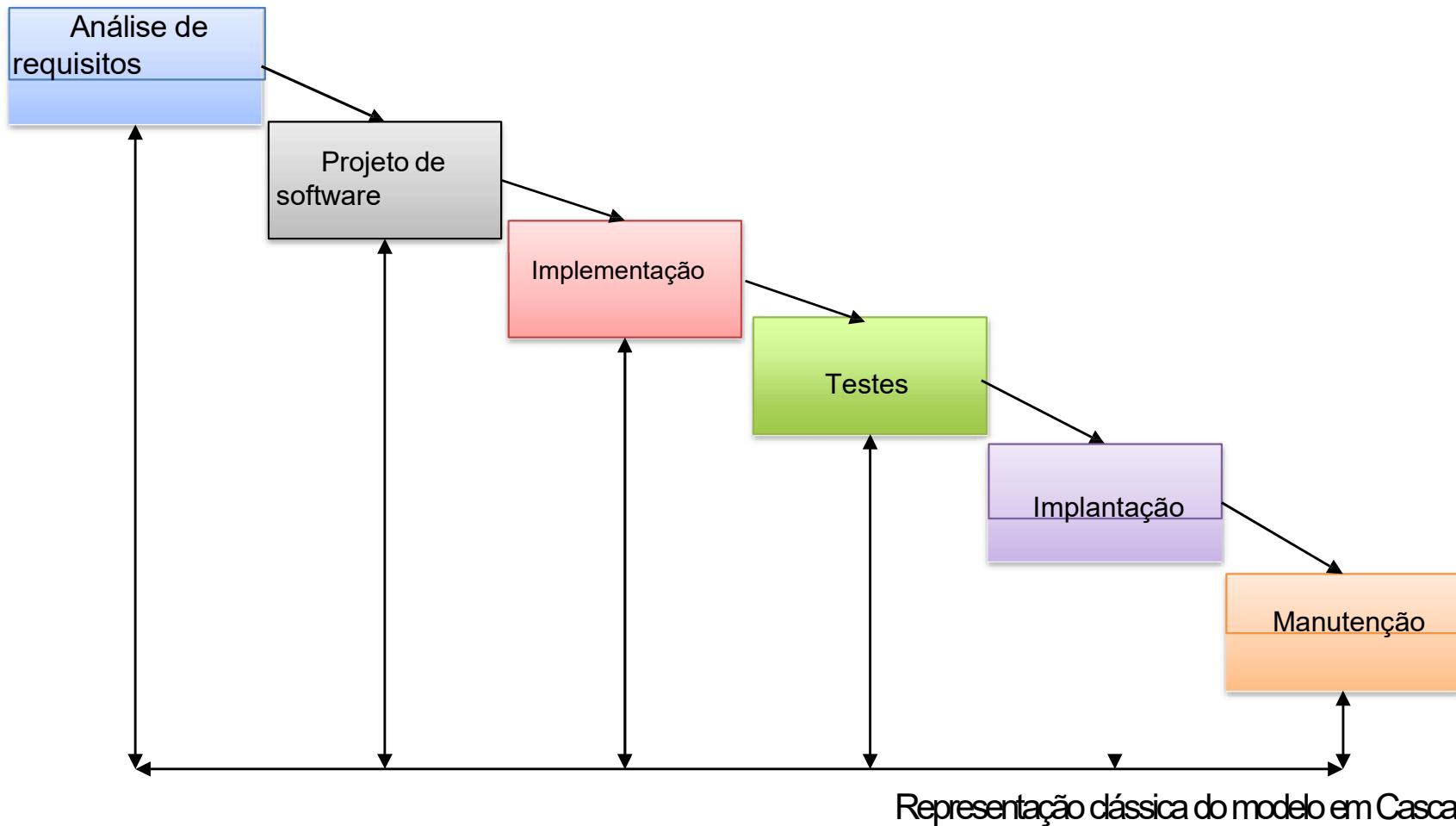
Modelo em Cascata ou Sequencial Linear

➤ Características:

- ✓ Ajuda os desenvolvedores a descrever o que precisam fazer (útil);
- ✓ Ajuda a explicar o processo de desenvolvimento aos clientes (simples e fácil);
- ✓ Os produtos intermediários são finalizados para começar próximo estágio e servem de insumo para o seu desenvolvimento;
- ✓ Seu enfoque está nos documentos e artefatos (requisitos, projetos, códigos).



Modelo em Cascata ou Sequencial Linear



Modelo em Cascata ou Sequencial Linear

- As funções, as restrições e os objetivos do sistema são estabelecidos por meio da consulta aos usuários do sistema.
- Deve-se compreender o domínio da informação, a função, desempenho e interfaces exigidos.
- Os requisitos (para o sistema e para o software) são documentados e revistos com o Cliente.

Analise de Requisitos

Modelo em Cascata ou Sequencial Linear

- Realiza todo o planejamento.
- São realizados os testes e as estimativas do desenvolvimento e criados os cronogramas para posterior acompanhamento do desenvolvimento do software.
- Define a arquitetura do sistema geral. É a tradução dos requisitos do software para um conjunto de representações que podem ser avaliadas quanto à qualidade, antes que a codificação se inicie.

Projeto de Software



Modelo em Cascata ou Sequencial Linear

- Escreve os códigos, as interfaces com os clientes, a arquitetura do software e a estrutura de dados, para atender a todas as regras de negócio levantadas na primeira fase.
- Recomenda-se realizar testes unitários nos módulos durante essa fase.

Implementação

Modelo em Cascata ou Sequencial Linear

- O teste se concentra nos aspectos lógicos internos do software, garantindo que todas as instruções tenham sido testadas;
- Se concentra também nos aspectos funcionais externos, para descobrir erros e garantir que a entrada definida produza resultados que concordem com os esperados.

Testes

Modelo em Cascata ou Sequencial Linear

➤ O sistema é instalado e colocado em operação.

Implantação

➤ Normalmente, é a fase mais longa do processo, pois tratará do suporte ao cliente, resolvendo os problemas quando o cliente der o feedback e implementando novos requisitos conforme necessidade.

Manutenção



Modelo em Cascata ou Sequencial Linear

- Apesar do modelo ser encadeado, passando de fase apenas com o término da fase anterior, na prática, acaba acontecendo uma sobreposição de algumas fases, ou seja, uma fase serve de insumo para a próxima fase e pode descobrir um problema na fase anterior, levando a fase anterior a ser modificada.
- Essa abordagem apresenta a vantagem de produzir uma documentação muito consistente em cada fase, porém é recomendada apenas quando os requisitos são pré-estabelecidos, bem conhecidos e com pouca probabilidade de mudança com o passar do tempo.



Modelo de prototipação

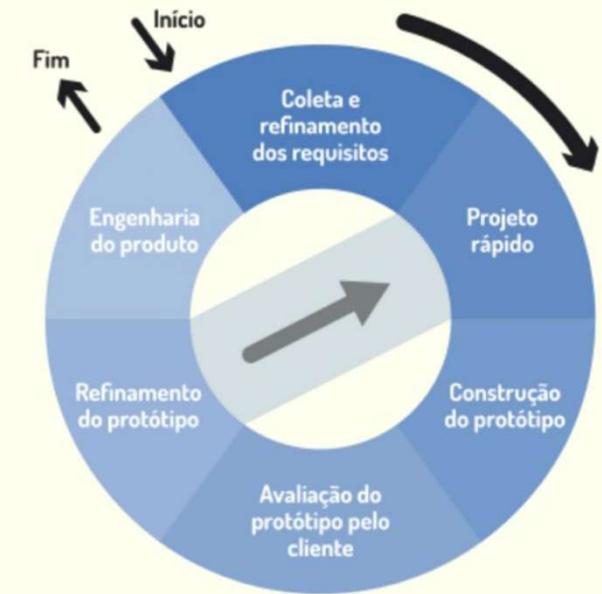
- Laudon (2014) afirma que prototipagem “consiste em montar um sistema experimental rapidamente e sem muitos gastos para submetê-lo à avaliação de usuários finais. O protótipo é uma versão funcional de um sistema de informação, ou de parte dele, mas deve ser considerado apenas um modelo preliminar.”
- De acordo com Pressman (2011), o protótipo serve como um mecanismo para identificação dos requisitos do software, pois, se ele é elaborado, o desenvolvedor tenta usar partes de programas existentes ou aplicar ferramentas que possibilitem programas executáveis serem gerados rapidamente.

Modelo de prototipação

Coleta e refinamento dos requisitos: ocorre a identificação dos requisitos junto ao cliente.

Nessa fase, o engenheiro mantém contato direto com o cliente apenas para captar as necessidades básicas de informação.

Projeto rápido: fase que define como será a iteração da prototipação e na qual acontece a criação de um projeto rápido, que contempla a parte do software que estará visível ao cliente. Abordagens de entrada e formatos de saída.

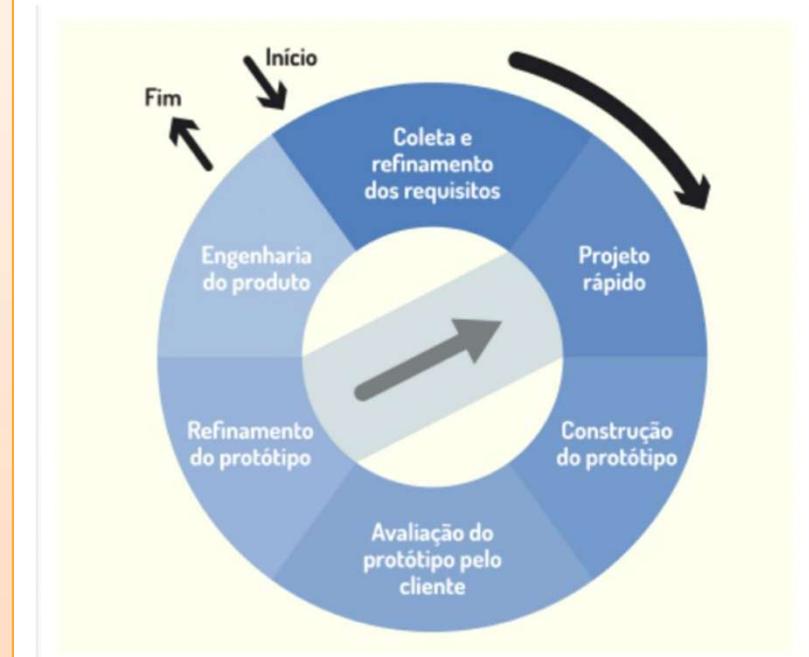


Modelo de prototipação

Construção Protótipo: implementação rápida do projeto.

Avaliação do protótipo pelo cliente: fase de utilização do protótipo desenvolvido na fase anterior pelo usuário final, a fim de validar o software, sugerir mudanças e aperfeiçoamentos.

Refinamento do Protótipo: cliente e desenvolvedor refinam os requisitos do software a ser desenvolvido. Ocorre neste ponto um processo de interação que pode conduzir à primeira atividade até que as necessidades do cliente sejam satisfeitas e o desenvolvedor compreenda o que precisa ser feito.



Modelo de prototipação

Engenharia de Produto:
Identificados os Requisitos, o protótipo deve ser descartado e a versão de produção deve ser construída considerando os critérios de qualidade.



Modelo de prototipação

➤ Vantagens:

- ✓ Todo o requisitos de sistema não tem que ser completamente determinado antecipadamente.
- ✓ A satisfação e interação do cliente no desenvolvimento do protótipo;
- ✓ Testes através do protótipo para atingir o objetivo final proposto.

➤ Desvantagens:

- ✓ O processo de prototipação pode dar ao usuário final a impressão que praticamente qualquer sugestão pode ser implementada, não importa qual estágio do processo de desenvolvimento se está.
- ✓ Além disso, para o usuário final não está claro o porquê da demora para entregar a aplicação final depois que uma versão demo do sistema foi exibida.
- ✓ O desenvolvedor frequentemente faz uma implementação comprometida (utilizando o que está disponível) com o objetivo de produzir rapidamente um protótipo. Depois de um tempo ele se familiariza com essas escolhas, e esquece que elas não são apropriadas para o produto final.

Desenvolvimento Rápido de Aplicação - RAD

- É o modelo sequencial linear mas que enfatiza um desenvolvimento extremamente rápido.
- A “alta velocidade” é conseguida através de uma abordagem de construção baseada em componentes.
- O desenvolvimento rápido de aplicação é adequado para construção de aplicações em curto espaço de tempo, utilizando o método incremental da prototipação, com o ciclo extremamente curto de desenvolvimento, que só é possível quando temos grande entendimento dos requisitos e o projeto é curto e simples.
- É um modelo de processo de desenvolvimento de software iterativo e incremental que enfatiza um ciclo de desenvolvimento extremamente curto (entre 60 e 90 dias).



Desenvolvimento Rápido de Aplicação - RAD

- A aplicação deve ser modularizada de forma que cada função deva ser completada em pelo menos 3 meses.
- Cada modulo pode ser alocado a uma equipe distinta e depois serem integradas para formar um todo.
- Criação e reutilização de componentes.
- No RAD, há o sequenciamento existente no modelo cascata e a ideia de apresentações parciais ao cliente para validação e continuação do desenvolvimento, vinda do modelo de prototipação, porém, nesse modelo, o que é apresentado ao cliente é, de fato, parte do software, e não apenas um protótipo.

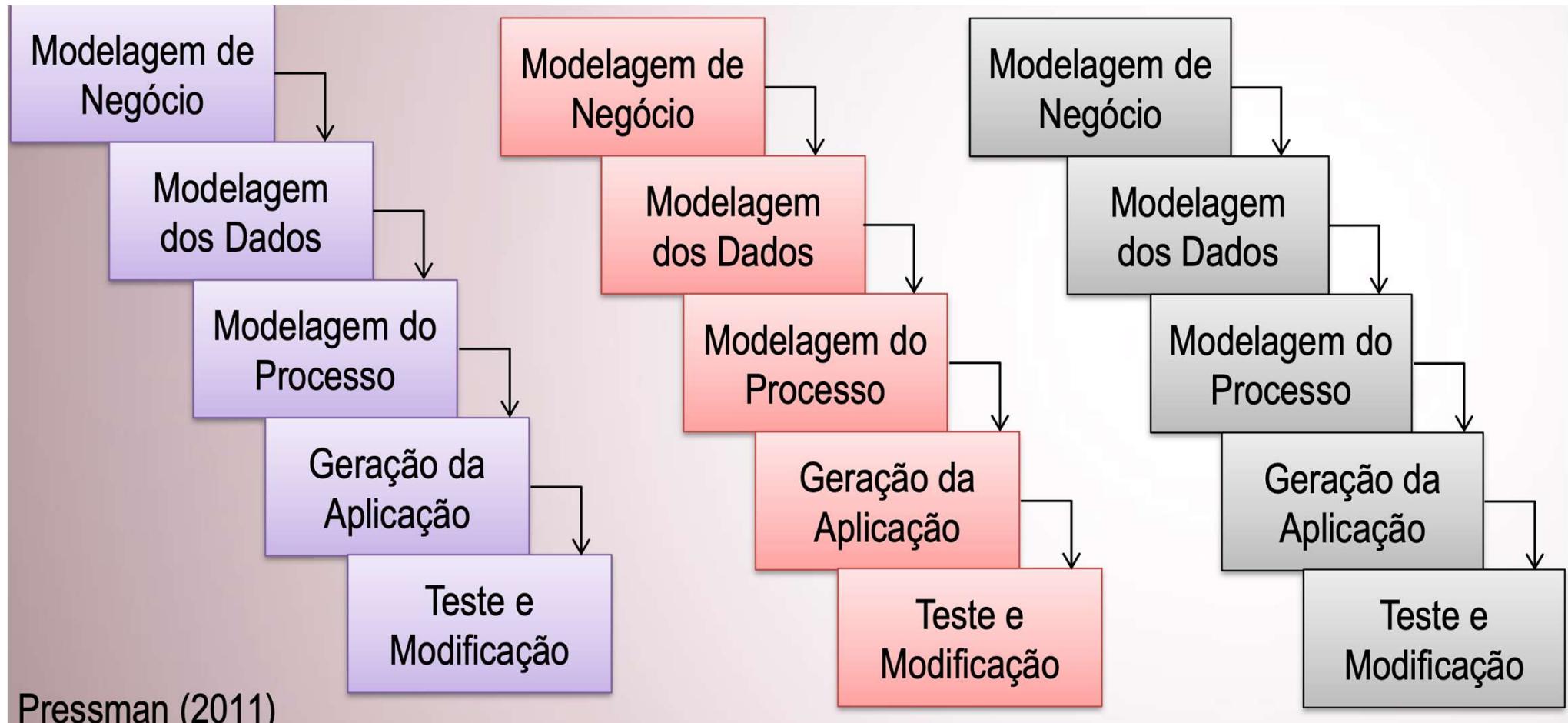


Desenvolvimento Rápido de Aplicação - RAD

- A aplicação deve ser modularizada de forma que cada função deva ser completada em pelo menos 3 meses.
- Cada modulo pode ser alocado a uma equipe distinta e depois serem integradas para formar um todo.
- Criação e reutilização de componentes.
- No RAD, há o sequenciamento existente no modelo cascata e a ideia de apresentações parciais ao cliente para validação e continuação do desenvolvimento, vinda do modelo de prototipação, porém, nesse modelo, o que é apresentado ao cliente é, de fato, parte do software, e não apenas um protótipo.



Desenvolvimento Rápido de Aplicação - RAD



Modelo Incremental

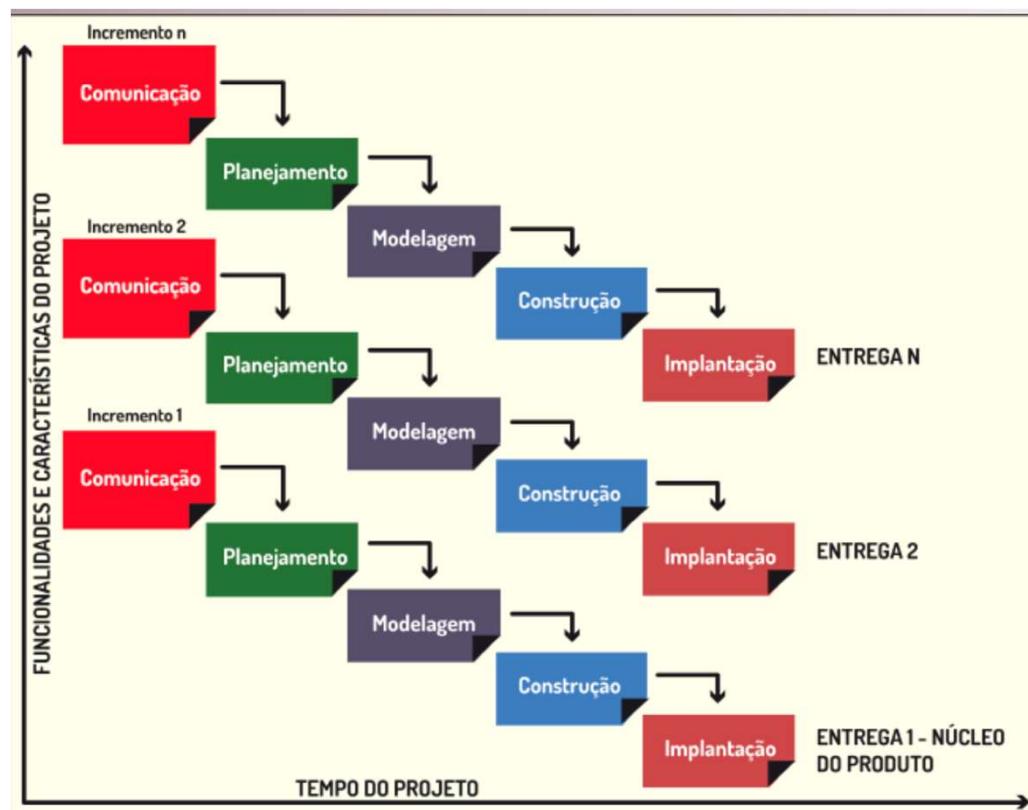
- Segundo Pressman (2011) “modelos evolucionários são iterativos. Apresentam características que possibilitam desenvolver versões cada vez mais completas do software”.
- Dessa forma, o primeiro modelo evolucionário que surgiu é o modelo incremental de desenvolvimento de software.
- Para Sommerville (2014) o sistema incremental tem o objetivo de reduzir o retrabalho custoso do modelo cascata, possibilitando ao cliente postergar decisões e requisitos conforme necessidade, mas mantendo o poder de gerenciamento que o modelo oferece.

Modelo Incremental

- Essa metodologia divide a fase de desenvolvimento do software em ciclos completos, passando por todas as fases existentes no modelo clássico, levantamento e análise de requisitos, projeto, implementação e testes.
- O modelo trabalha com 5 (cinco) fases distintas: comunicação, planejamento, modelagem, construção e implantação.



Modelo Incremental



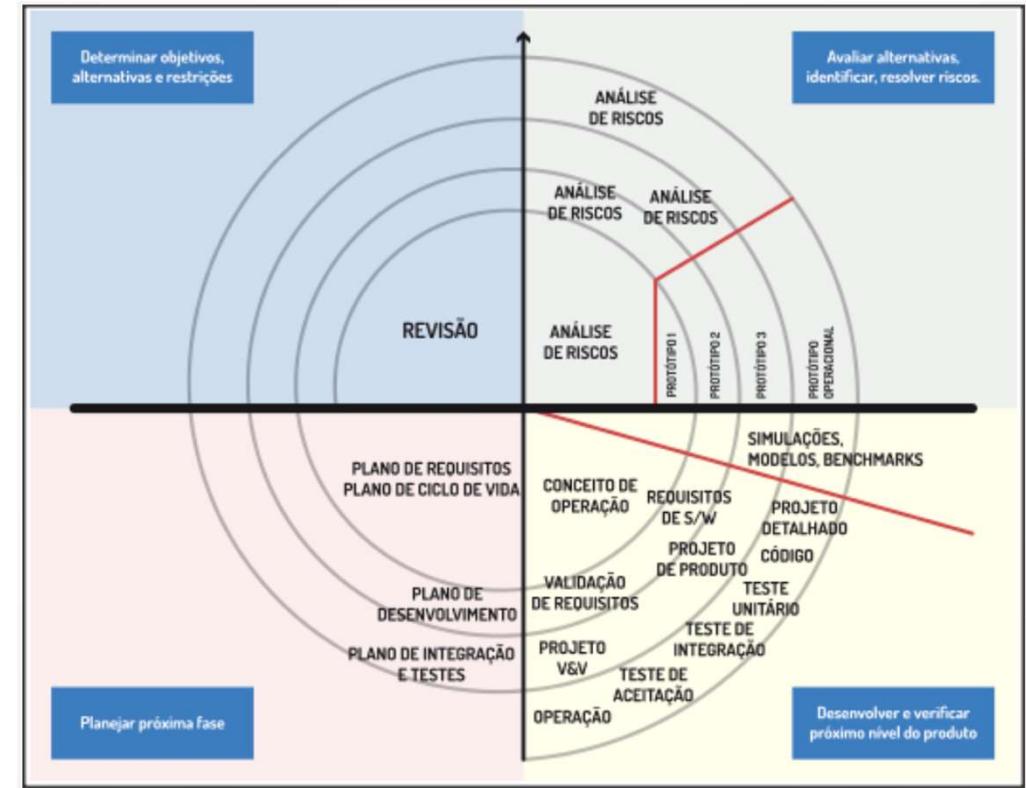
1. **Comunicação:** trata das informações do negócio, como são geradas, processadas e quem utilizará. Define os requisitos.
2. **Planejamento:** define os objetos, suas características e como se relacionarão entre si.
3. **Modelagem:** desenha o processo, visando atender à função do negócio, e define os procedimentos necessários para a manipulação dos objetos de dados definidos na etapa anterior.
4. **Construção:** ocorre a geração da aplicação, normalmente por meio de reutilização de componentes que serão integrados posteriormente.
5. **Implantação:** acontece alguns pequenos testes e a entrega de parte do produto.

Modelo espiral

- Pressman (2011) define que esse é um modelo de processo de software evolucionário que acopla a natureza **iterativa da prototipação** com os **aspectos sistemáticos e controlados do modelo cascata**. Fornece potencial para o rápido desenvolvimento de versões cada vez mais completas de software.
- Cada loop na espiral representa uma fase do processo de software.
- As atividades acontecem em forma de espiral, no sentido horário, começando do centro, no ponto indicado como início. O software será construído em versões evolutivas, sendo que, em cada iteração da espiral, temos a entrega de um protótipo ou de uma versão mais completa do sistema.

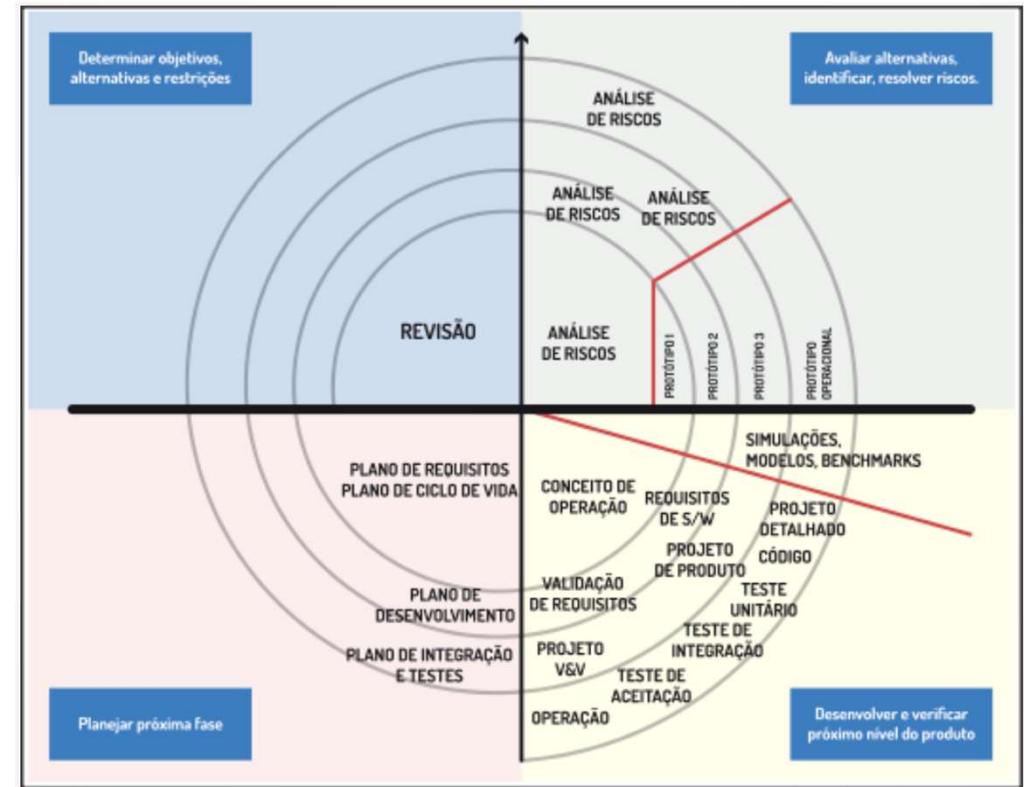
Modelo espiral

- 1. Planejamento e Requisitos** - nessa fase, é feito o levantamento de requisitos junto ao cliente, estabelecendo as restrições e as diretrizes do software, bem como os riscos. É criado um plano de gerenciamento, com cronogramas e estimativas de custos.
- 2. Análise de Riscos** - nesse ponto, é realizado uma análise para cada um dos riscos levantados e estudado maneiras de reduzi-los.
- 3. Desenvolvimento e Validação** – o protótipo é desenvolvido e validado.
- 4. Entrega e Avaliação de Continuidade** - nessa fase, ocorre uma entrega para o cliente e uma avaliação, para decidir se o projeto continuará para o próximo loop da espiral. Caso seja decidida a continuidade, o custo, o cronograma e o planejamento da quantidade de iterações planejadas anteriormente serão refinados de acordo com os feedbacks recebidos do usuário.



Modelo espiral

- Sommerville (2014) “A importante distinção entre o modelo espiral e outros modelos de processo de software é a explícita consideração dos **riscos** no modelo em espiral”.
- Segundo o **PMBOK**: riscos são efeitos negativos nos objetivos do projeto, como escopo, qualidade, custo e prazo.
- Exemplo:
Durante a primeira espiral, é utilizado prototipação, para tirar dúvidas e realizar alinhamento com o cliente, visando à redução dos riscos, e, nos próximos loops da espiral, é utilizado o modelo cascata.



Montagem de Componente

- Faz reuso de software visando o ganho em agilidade e confiabilidade, utilizando tecnologias de orientação a objetos.
- Sendo que agilidade desse modelo se deve ao fato do **reuso dos componentes**, obtido por meio de adaptações e de combinações.
- *Pressman (2011)* define que “a engenharia de software baseada em componentes identifica, constrói, cataloga e dissemina um conjunto de componentes de software em determinado domínio de aplicação. Esses componentes são então qualificados, adaptados e integrados para uso em um novo sistema..



Montagem de Componente

➤ Exemplo:

Um software para controle de loja de materiais de construção.
Para ele, é criado o componente de login e o componente de
cadastro de usuários.

○ Um software para controle de uma loja de moveis.

Pode ser reutilizado os dois componentes criados para o software de controle da loja de materiais de
construção para compor o software da loja de moveis, visto que já estão em uso, funcionando e testados.



Técnicas de 4a Geração

- As técnicas de quarta geração englobam um **conjunto de ferramentas de softwares** que visa permitir ao desenvolvedor especificar as características do software em alto nível.
- A técnica de quarta geração tem o objetivo de que os requisitos descritos pelo cliente sejam capazes de ser introduzidos em uma ferramenta que automaticamente geraria a estratégia do projeto, as codificações e realizaria os testes.

Engenharia de Software

**Aula: Metodologias Ágeis
de Desenvolvimento de
software.**



Prof. Anderson Augusto Bosing

Por que agil ? O que é agilidade?



O que é agilidade ?

Em fevereiro de 2001, dezessete representantes de diversas práticas e metodologias de desenvolvimento se reuniram em uma estação de esqui, em Utah nos EUA para discutir métodos mais leves de desenvolvimento do que o tradicional desenvolvimento orientado a documentos.

Auto denominados de “The Agile Alliance” criaram o Manifesto for Agile Software Development ou simplesmente Manifesto Ágil para definir a abordagem hoje conhecida como desenvolvimento ágil.

“Estamos descobrindo maneiras melhores de desenvolver software fazendo nós mesmos e ajudando outros a fazê- lo.



O que é agilidade ?

- Agilidade:
Rapidez, desembaraço;
Qualidade de quem é veloz;
- Capacidade de responder rapidamente a mudanças:
Mudanças de tecnologias, de equipe, de requisitos...
- Entregar valor ao cliente quando se lida com
imprevisibilidade e dinamismo dos projetos;

Metodologias ágeis

- “Linha de pensamento” revolucionária:
 - Precisamos parar de tentar evitar mudanças;
- Metodologias ágeis:
 - “Filosofia” onde muitas “metodologias” se encaixam;
 - Definem um conjunto de atitudes e não um processo prescritivo.

Metodologias ágeis

Reação às metodologias tradicionais;

- “Manifesto ágil” (2001):<https://agilemanifesto.org/>
 - Movimento iniciado por programadores experientes e consultores em desenvolvimento de software;
 - Questiona e se opõe a uma série de mitos/práticas adotadas em abordagens tradicionais de Engenharia de Software e Gerência de Projetos.



Metodologias ágeis

Princípios do Manifesto Ágil:

- Indivíduos e interações são mais importantes que processos e ferramentas;
- Software funcionando é mais importante que documentação completa (abrangente);
- Colaboração do cliente é mais importante que negociação de contratos;
- Adaptação às mudanças é mais importante que seguir um plano;



Metodologias ágeis

Segundo Pressman:

- A engenharia de software ágil combina uma filosofia e um conjunto de diretrizes de desenvolvimento;
- A filosofia encoraja a satisfação do cliente e a entrega incremental de software logo no início;
- Equipes de projetos pequenas e altamente motivadas;
- Métodos informais;
- Produtos de trabalho de engenharia de software mínimos e simplicidade global de desenvolvimento;
- As diretrizes de desenvolvimento enfatizam a entrega em contraposição à análise e ao projeto (apesar destas atividades não serem desencorajadas);
- Comunicação ativa e contínua entre desenvolvedores e clientes.



Prof. Anderson Augusto Bosing

Princípios da agilidade

1. A mais alta prioridade é a satisfação do cliente, por meio da liberação mais rápida e contínua de software de valor;
2. Receba bem as mudanças de requisitos, mesmo em estágios tardios do desenvolvimento. Processos ágeis devem admitir mudanças que trazem vantagens competitivas para o cliente;
3. Libere software freqüentemente (em intervalos de 2 semanas até meses), dando preferência para uma escala de tempo mais curta;
4. Mantenha pessoas ligadas ao negócio (clientes) e desenvolvedores trabalhando juntos a maior parte do tempo do projeto;



Princípios da agilidade

5. Construa projetos com indivíduos motivados, dê a eles o ambiente e suporte que precisam e confie neles para ter o trabalho realizado;
6. O método mais eficiente e efetivo para repassar informação entre uma equipe de desenvolvimento é pela comunicação face-a-face;
7. Software funcionando é a principal medida de progresso de um projeto de software;
8. Processos ágeis promovem desenvolvimento sustentado. Assim, patrocinadores, desenvolvedores e usuários devem ser capazes de manter conversação pacífica indefinidamente;



Princípios da agilidade

9. A atenção contínua para a excelência técnica e um bom projeto (design) aprimoram a agilidade;
10. Simplicidade - a arte de maximizar a quantidade de trabalho não feito – é essencial, devendo ser assumida em todos os aspectos do projeto;
11. As melhores arquiteturas, requisitos e projetos emergem de equipes auto-organizadas;
12. Em intervalos regulares, as equipes devem refletir sobre como se tornarem mais efetivas, e então refinarem e ajustarem seu comportamento de acordo.



Características gerais

- Procuram minimizar riscos desenvolvendo software em pequenos espaços de tempo (iterações);
- Cada iteração é como um pequeno projeto:
 - Planejamento, requisitos, projeto, codificação, testes...
 - Propõem o desenvolvimento de software de forma mais rápida, com um grande número de ciclos, mas com qualidade;
- Objetivo de cada iteração:
 - Produzir componentes de software (incrementos);
 - Arquitetura vai sendo desenhada a partir da refatoração dos componentes;



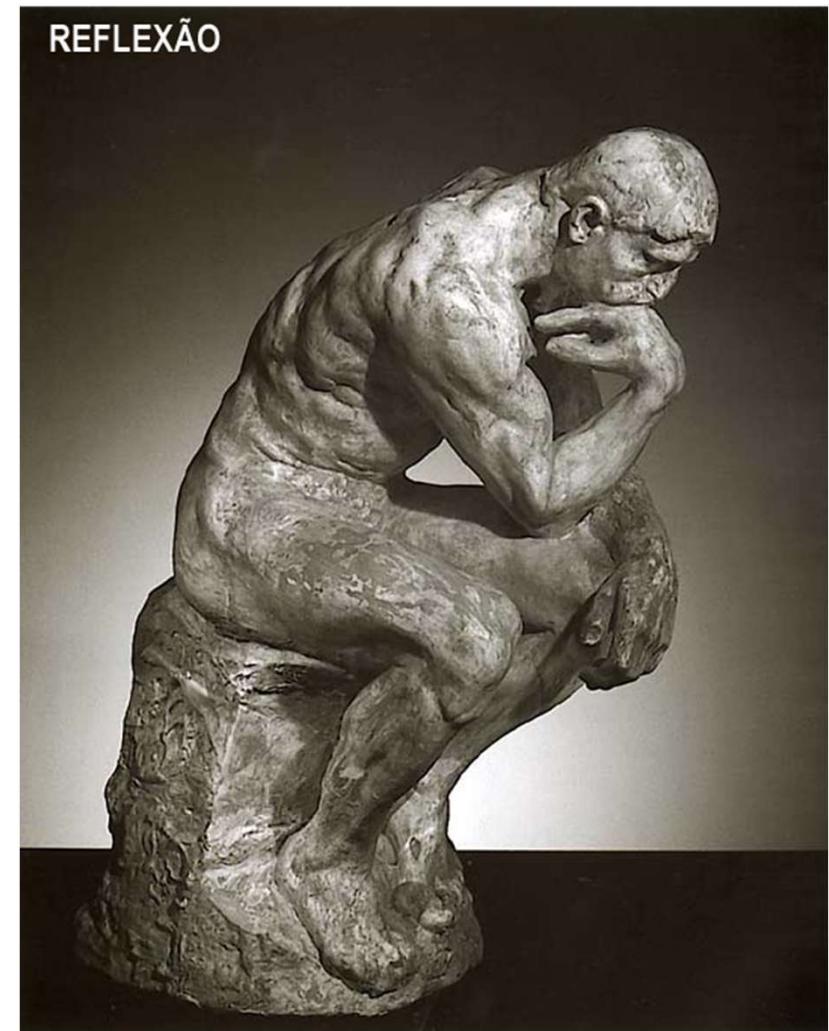
Características gerais

- Um catalizador efetivo para o feedback do cliente é um protótipo executável ou parte de um sistema operacional (“incrementos de software”);
- “Incrementos de software” devem ser entregues em curtos períodos de tempo de modo que a adaptação acerte o passo com as modificações (imprevisibilidade);
- Essa abordagem iterativa habilita o cliente a avaliar o incremento de software regularmente, fornecer feedback necessário à equipe de software e influenciar as adaptações do processo feitas para acomodar o feedback.

Características gerais

- Encorajamento de atitudes reflexivas e contínuo aprendizado;
- Lições aprendidas: as lições aprendidas de qualquer atividade de solução de problema (inclusive aquelas que resolvem o problema errado) podem ser benéficas para a equipe mais adiante no projeto ou em outros projetos;

REFLEXÃO



Características gerais

- Fatores Humanos: Segundo Cockburn e Highsmith, “o desenvolvimento ágil enfoca os talentos e habilidades dos indivíduos moldando o processo a pessoas e equipe específicas”;
- Auto-organização no contexto de desenvolvimento ágil:
 1. A equipe ágil organiza-se para o trabalho a ser feito;
 2. A equipe organiza o processo para melhor acomodar seu ambiente local;
 3. A equipe organiza o cronograma de trabalho para conseguir melhor entrega do incremento de software;
- Uma equipe auto-organizada está no controle do trabalho que realiza. A equipe estabelece os seus próprios compromissos e define os planos para cumpri-los;

Características gerais

Segundo Ken Schwaber:

“A equipe seleciona quanto trabalho acredita que pode realizar dentro da iteração e a equipe se compromete com o trabalho. Nada desmotiva tanto uma equipe quanto alguém de fora assumir compromissos por ela. Nada motiva tanto uma equipe quanto a aceitação da responsabilidade de cumprir seus compromissos que ela própria estabeleceu.”



Conclusões

Pró-agilidade x Pró-engenharia tradicional segundo HighSmith:

- “Os metodologistas tradicionais são um punhado de bitolados que preferem produzir documentação perfeita a um sistema funcionando que satisfaça às necessidades do negócios.”;
- “Os metodologistas levianos, quer dizer, ‘ágéis’, são um punhado de gloriosos hackers que terão uma grande surpresa quando tiverem de ampliar seus brinquedos para chegar a um software que abranja toda a empresa.”;



Conclusões

- Não fique limitado a uma “arma” ou técnica em particular;
- Metodologias diferentes são necessárias para diferentes tipos de projetos
 - Fatores a serem considerados:
 - Número de Pessoas envolvidas no Projeto;
 - Criticidade do Sistema;
 - Prioridades do Projeto;
- Construa a sua “caixa de ferramentas”;

Conclusões

- Em cada modelo ágil (XP, SCRUM, Crystal, FDD) há um conjunto de “idéias” (“tarefas de trabalho”) que representam um afastamento significativo da engenharia de software convencional;
- Segundo Pressman, muitos conceitos ágeis são simples adaptações de bons conceitos de engenharia de software;
- Conclusão segundo Pressman: “há muito a ser ganho considerando o melhor de ambas as escolas, e quase nada a ser ganho denegrindo qualquer uma dessas abordagens.”

Engenharia de Software

Aula: Agilidade e Scrum.



Prof. Anderson Augusto Bosing

SCRUM

Em fevereiro de 2001, dezessete representantes de diversas práticas e metodologias de desenvolvimento se reuniram em uma estação de esqui, em Utah nos EUA para discutir métodos mais leves de desenvolvimento do que o tradicional desenvolvimento orientado a documentos.

Auto denominados de “The Agile Alliance” criaram o Manifesto for Agile Software Development ou simplesmente Manifesto Ágil para definir a abordagem hoje conhecida como desenvolvimento ágil.

“Estamos descobrindo maneiras melhores de desenvolver software fazendo nós mesmos e ajudando outros a fazê- lo.



SCRUM

MANIFESTO FOR AGILE SOFTWARE DEVELOPMENT

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

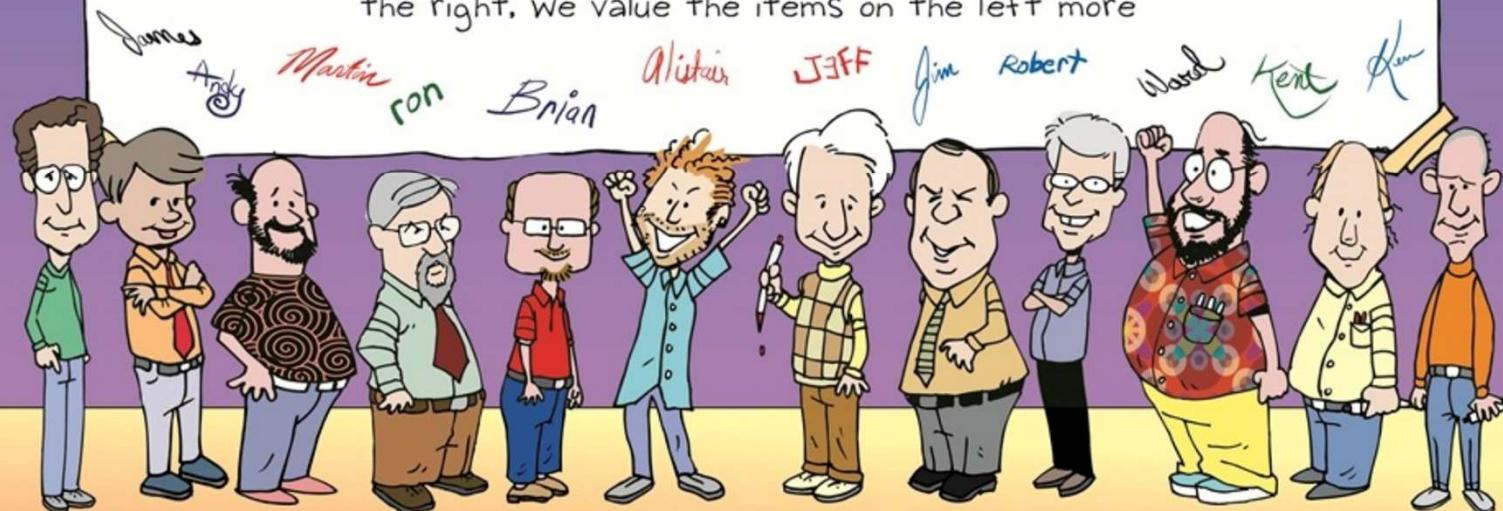
INDIVIDUALS AND INTERACTIONS — OVER PROCESSES AND TOOLS

WORKING SOFTWARE — OVER COMPREHENSIVE DOCUMENTATION

CUSTOMER COLLABORATION — OVER CONTRACT NEGOTIATION

RESPONDING TO CHANGE — OVER FOLLOWING A PLAN

That is, while there is value in the items on the right, we value the items on the left more



ThoughtWorks®

© 2011, ThoughtWorks, Inc. All rights reserved. All trademarks and logos are property of their respective owners.

ThoughtWorks
studios



Prof. Anderson Augusto Bosing

SCRUM

O *Scrum* (o nome é derivado de uma atividade que ocorre durante um jogo de *rugby*) é um modelo ágil de processo que foi desenvolvido por Jeff Sutherland e por sua equipe no início da década de 90.



CRIADOS DO SCRUM



SCRUM

O Scrum é um framework de processo ágil utilizado para gerenciar e controlar o desenvolvimento de um produto de software através de práticas iterativas e incrementais.

É composto por um conjunto de boas práticas de gestão que admite ajustes rápidos, acompanhamento e visibilidade constantes e planos realísticos;



SCRUM

- Uma alternativa de utilizar métodos ágeis na gerência de projetos;
- Pode ser aplicável a qualquer tipo de projeto;
- É simples:
 - “Processo, artefatos e regras são poucos e fáceis de entender”;
 - “A simplicidade pode ser decepcionante aos acostumados com metodologias clássicas”.



SCRUM

- Não é um método prescritivo:
 - Não define previamente o que deve ser feito em cada situação;
 - Projetos complexos não permitem prever todos os eventos;
- Aplica o senso comum:
 - Combinação de experiência, treinamento, confiança e inteligência de toda a equipe;
 - Senso comum em vez do senso de uma única pessoa é uma das razões do sucesso do *Scrum*;

SCRUM

- Os princípios *Scrum* são consistentes com o manifesto ágil:
 - Pequenas equipes de trabalho são organizadas de modo a maximizar a comunicação, minimizar a supervisão e maximizar o compartilhamento de conhecimento tácito informal.
 - O processo precisa ser adaptável tanto a modificações técnicas quanto de negócios para garantir que o melhor produto possível seja produzido.
 - O processo produz freqüentes incrementos de software que podem ser inspecionados, ajustados, testados, documentados e expandidos.
 - O trabalho de desenvolvimento e o pessoal que realiza é dividido em partições claras, de baixo acoplamento, ou em pacotes.
 - Testes e documentação constantes são realizados à medida que o produto é construído.



PAPÉIS DO SCRUM

Todas as responsabilidades de gerenciamento são divididas entre três papéis:

- Product Owner;
- Scrum Master;
- Equipe/Desenvolvedores;

Para o bom funcionamento do *Scrum* as pessoas responsáveis pelo projeto devem ter autoridade para fazer o que for necessário pelo seu sucesso;

Pessoas não responsáveis não podem interferir no projeto:

- Tal fato gera aumento de produtividade;
- Evita situações constrangedoras para os envolvidos;

Cada um conhece sua participação frente ao projeto e trabalha em conjunto para conseguir alcançar o objetivo definido.



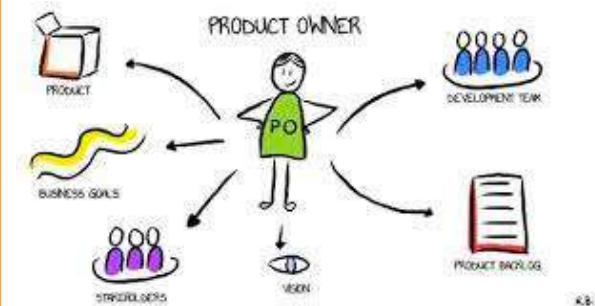
PAPÉIS – Product Owner (PO)

O Product Owner é responsável por maximizar o valor do produto resultante do trabalho do Scrum Team. A forma como isso é feito pode variar amplamente entre organizações, Scrum Teams e indivíduos.

O Product Owner também é responsável pelo gerenciamento eficaz do Product Backlog , que inclui:

- Desenvolver e comunicar explicitamente a meta do produto;
- Criar e comunicar claramente os itens do Product Backlog;
- Ordenar os itens do Product Backlog; e,
- Garantir que o Product Backlog seja transparente, visível e comprehensível.

O Product Owner pode fazer o trabalho acima ou pode delegar a responsabilidade a outros. Independentemente disso, o Product Owner ainda é o responsável.



PAPÉIS – Product Owner (PO)

Para que os Product Owners tenham sucesso, toda a organização deve respeitar suas decisões.

Essas decisões são visíveis no conteúdo e na ordem do Product Backlog e por meio do incremento inspecionável na revisão da sprint.

O Product Owner é uma pessoa, não um comitê. O Product Owner pode representar as necessidades de muitos stakeholders no Product Backlog. Aqueles que desejam alterar o Product Backlog podem fazê-lo tentando convencer o Product Owner



PAPÉIS – Scrum Master (SM)

O Scrum Master é responsável por estabelecer o Scrum.

Eles fazem isso ajudando todos a entender a teoria e a prática do Scrum, tanto no Scrum Team quanto na organização.

O Scrum Master é responsável pela eficácia do Scrum Team. Eles fazem isso permitindo que o Scrum Team melhore suas práticas, dentro do framework Scrum.

Scrum Masters são verdadeiros líderes que servem ao Scrum Team e à organização como um todo.

O Scrum Master serve ao Scrum Team de várias maneiras, incluindo:

- Treinar os membros do time em autogerenciamento e cross-funcionalidade;
- Ajudar o Scrum Team a se concentrar na criação de incrementos de alto valor que atendem à Definição de Pronto;

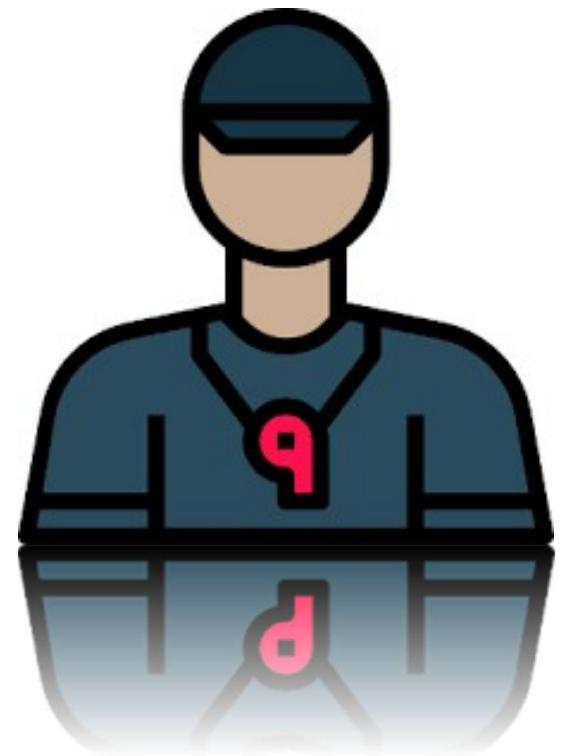


PAPÉIS – Scrum Master (SM)

- Provocando a remoção de impedimentos ao progresso do Scrum Team;
- Garantir que todos os eventos Scrum ocorram e sejam positivos, produtivos e mantidos dentro do Timebox.

O Scrum Master serve o Product Owner de várias maneiras, incluindo:

- Ajudar a encontrar técnicas para a definição eficaz de meta do Produto e gerenciamento do Product Backlog;
- Ajudar o Scrum Team a entender a necessidade de itens do Product Backlog claros e concisos;
- Ajudar a estabelecer o planejamento empírico do produto para um ambiente complexo; e,
- Facilitar a colaboração dos stakeholder, conforme solicitado ou necessário.



PAPÉIS – Scrum Master (SM)

O Scrum Master serve a organização de várias maneiras, incluindo:

- Liderar, treinar e orientar a organização na adoção do Scrum;
- Planejar e aconselhar implementações de Scrum dentro da organização;
- Ajudar os funcionários e os stakeholders a compreender e aplicar uma abordagem empírica para trabalhos complexos; e,
- Remover barreiras entre stakeholders e Scrum Teams.



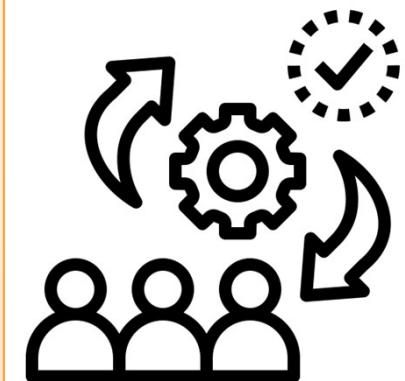
PAPÉIS – Equipe/Desenvolvedores

Developers são as pessoas do Scrum Team que estão comprometidas em criar qualquer aspecto de um Incremento utilizável a cada Sprint.

As habilidades específicas necessárias pelos Developers geralmente são amplas e variam de acordo com o domínio de trabalho.

No entanto, os Developers são sempre responsáveis por:

- Criar um plano para a Sprint, o Sprint Backlog;
- Introduzir gradualmente qualidade aderindo a uma Definição de Pronto;
- Adaptar seu plano a cada dia em direção à meta da Sprint;
- Responsabilizar-se mutuamente como profissionais.



PAPÉIS – Equipe/Desenvolvedores

Developers são as pessoas do Scrum Team que estão comprometidas em criar qualquer aspecto de um Incremento utilizável a cada Sprint.

As habilidades específicas necessárias pelos Developers geralmente são amplas e variam de acordo com o domínio de trabalho.

No entanto, os Developers são sempre responsáveis por:

- Criar um plano para a Sprint, o Sprint Backlog;
- Introduzir gradualmente qualidade aderindo a uma Definição de Pronto;
- Adaptar seu plano a cada dia em direção à meta da Sprint;
- Responsabilizar-se mutuamente como profissionais.



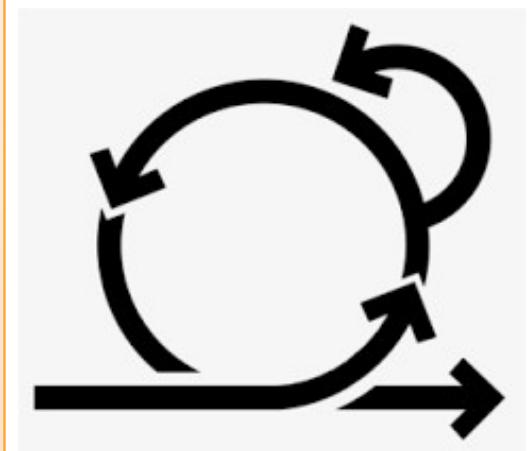
Eventos do Scrum

Cada evento no Scrum é uma oportunidade formal para inspecionar e adaptar os artefatos do Scrum. Esses eventos são projetados especificamente para permitir a transparência necessária.

A falha em operar quaisquer eventos conforme prescrito resulta em oportunidades perdidas de inspeção e adaptação.

Os eventos são usados no Scrum para criar regularidade e minimizar a necessidade de reuniões não definidas no Scrum.

O ideal é que todos os eventos sejam realizados no mesmo horário e local para reduzir a complexidade.



Sprint

Sprints são o coração do Scrum, onde ideias são transformadas em valor.

São eventos de duração fixa de um mês ou menos para criar consistência.

Uma nova Sprint começa imediatamente após a conclusão da Sprint anterior.

Todo o trabalho necessário para atingir a meta do Produto, incluindo Sprint Planning, Daily Scrums, Sprint Review e Sprint Retrospective, acontece dentro de Sprints.

Uma Sprint pode ser cancelada se a Meta da Sprint se tornar obsoleta. Apenas o Product Owner tem autoridade para cancelar a Sprint.



Sprint Planning

A Sprint Planning inicia a Sprint ao definir o trabalho a ser realizado na Sprint.

Este plano resultante é criado pelo trabalho colaborativo de todo o Scrum Team.

O Product Owner garante que os participantes estejam preparados para discutir os itens mais importantes do Product Backlog e como eles são mapeados para a Meta do Produto.

A Sprint Planning aborda os seguintes tópicos:

Por que esta Sprint é valiosa?

O que pode ser feito nesta Sprint?

Como o trabalho escolhido será realizado?

A Sprint Planning tem um Timebox definido com duração máxima de oito horas para uma Sprint de um mês. Para Sprints mais curtas, o evento geralmente é mais curto.



Daily Scrum

O propósito da Daily Scrum é inspecionar o progresso em direção a Meta da Sprint e adaptar o Sprint Backlog conforme necessário, ajustando o próximo trabalho planejado.

A Daily Scrum é um evento de 15 minutos para os Developers do Scrum Team.

Para reduzir a complexidade, é realizado no mesmo horário e local, todos os dias úteis da Sprint.

Se o Product Owner ou o Scrum Master estão trabalhando ativamente nos itens do Sprint Backlog, eles participam como Developers.

Os Developers podem selecionar qualquer estrutura e técnicas que quiserem, desde que seu Daily Scrum se concentre no progresso em direção a Meta da Sprint e produza um plano de ação para o próximo dia de trabalho.

Isso cria foco e melhora o autogerenciamento, melhoram as comunicações, identificam os impedimentos, promovem a rápida tomada de decisões e consequentemente, eliminam a necessidade de outras reuniões.



Sprint Review

O propósito da Sprint Review é inspecionar o resultado da Sprint e determinar as adaptações futuras.

O Scrum Team apresenta os resultados de seu trabalho para os principais stakeholders e o progresso em direção a Meta do Produto é discutido.

Durante o evento, o Scrum Team e os stakeholders revisam o que foi realizado na Sprint e o que mudou em seu ambiente.

O Product Backlog também pode ser ajustado para atender a novas oportunidades.

A Sprint Review é uma sessão de trabalho e o Scrum Team deve evitar limitá-la a uma apresentação.

A Sprint Review é o penúltimo evento da Sprint e tem um Timebox com prazo máximo de quatro horas para uma Sprint de um mês. Para Sprints mais curtas, o evento geralmente é mais curto.



Sprint Retrospective

O propósito da Sprint Retrospective é planejar maneiras de aumentar a qualidade e a eficácia.

O Scrum Team inspeciona como foi a última Sprint em relação a indivíduos, interações, processos, ferramentas e sua Definição de Pronto.

Os elementos inspecionados geralmente variam com o domínio de trabalho.

As suposições que os desviaram são identificadas e suas origens exploradas. O Scrum Team discute o que deu certo durante a Sprint, quais problemas encontraram e como esses problemas foram (ou não) resolvidos.

O Scrum Team identifica as mudanças mais úteis para melhorar sua eficácia. As melhorias mais impactantes são endereçadas o mais rápido possível.

Essas podem até ser adicionadas ao Sprint Backlog para a próxima Sprint. A Sprint Retrospective conclui a Sprint.

É limitada pelo Timebox de no máximo três horas para uma Sprint de um mês. Para Sprints mais curtas, o evento geralmente é mais curto.



Artefatos do Scrum

Os artefatos do Scrum representam trabalho ou valor.

Eles são projetados para maximizar a transparência das principais informações.

Assim, todos os que os inspecionam têm a mesma base para adaptação. Cada artefato contém um compromisso para garantir que ele forneça informações que aumentem a transparência e o foco contra o qual o progresso pode ser medido:

- Para o Product Backlog, é a Meta do produto.

- Para o Sprint Backlog, é a Meta da Sprint.
- Para o incremento, é a Definição de Pronto.

Esses compromissos existem para reforçar o empirismo e os valores Scrum para o Scrum Team, e seus stakeholders.



Product Backlog

O Product Backlog é uma lista ordenada e emergente do que é necessário para melhorar o produto. É a única fonte de trabalho realizado pelo Scrum Team.

Os itens do Product Backlog que podem ser realizados pelo Scrum Team em uma Sprint são considerados preparados para seleção no evento Sprint Planning.

Eles geralmente adquirem esse grau de transparência após as atividades de refinamento.

O Product Backlog refinement é o ato de quebrar e incluir definição adicional aos itens do Product Backlog para ter itens menores e mais precisos.

Esta é uma atividade contínua para adicionar detalhes, como descrição, ordem e tamanho.

Os atributos geralmente variam de acordo com o domínio de trabalho. Os Developers que farão o trabalho são responsáveis pelo dimensionamento. O Product Owner pode influenciar os Developers, ajudando-os a entender e selecionar trade-offs (trocas de itens).



Sprint Backlog

O Sprint Backlog é composto pela Meta da Sprint (por que), o conjunto de itens do Product Backlog selecionados para a Sprint (o que), bem como um plano de ação para entregar o Incremento (como).

O Sprint Backlog é um plano feito por e para os Developers. É uma imagem altamente visível, em tempo real do trabalho que os Developers planejam realizar durante a Sprint para atingir a Meta da Sprint.

Consequentemente, o Sprint Backlog é atualizado ao longo da Sprint conforme mais é aprendido. Deve ter detalhes suficientes para que eles possam inspecionar seu progresso na Daily Scrum.



Incremento

Um incremento é um trampolim concreto em direção a Meta do produto. Cada incremento é adicionado a todos os incrementos anteriores e completamente verificado, garantindo que todos os incrementos funcionem juntos.

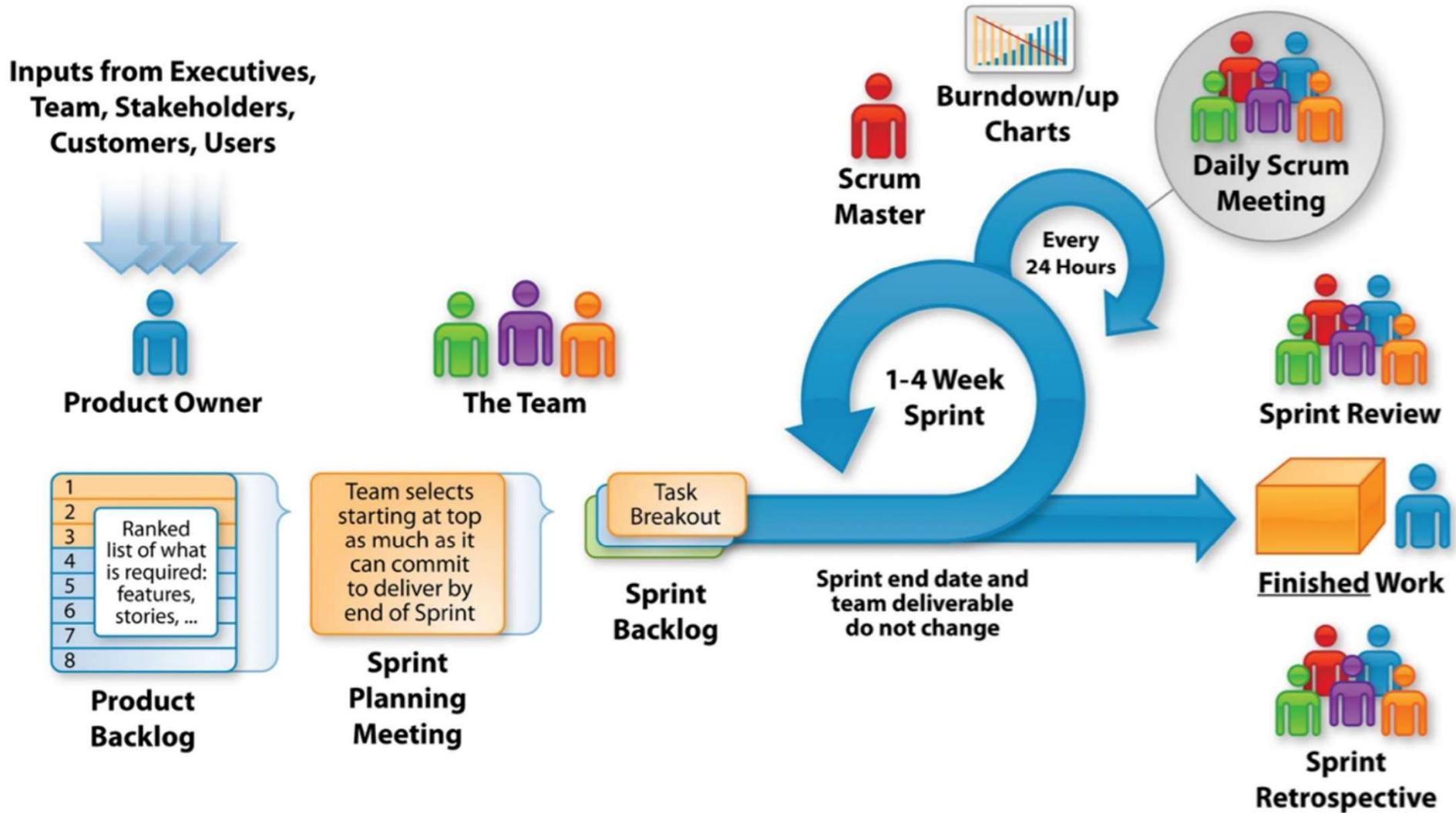
A fim de fornecer valor, o incremento deve ser utilizável.

Vários incrementos podem ser criados em uma Sprint. A soma dos incrementos é apresentada na Sprint Review.

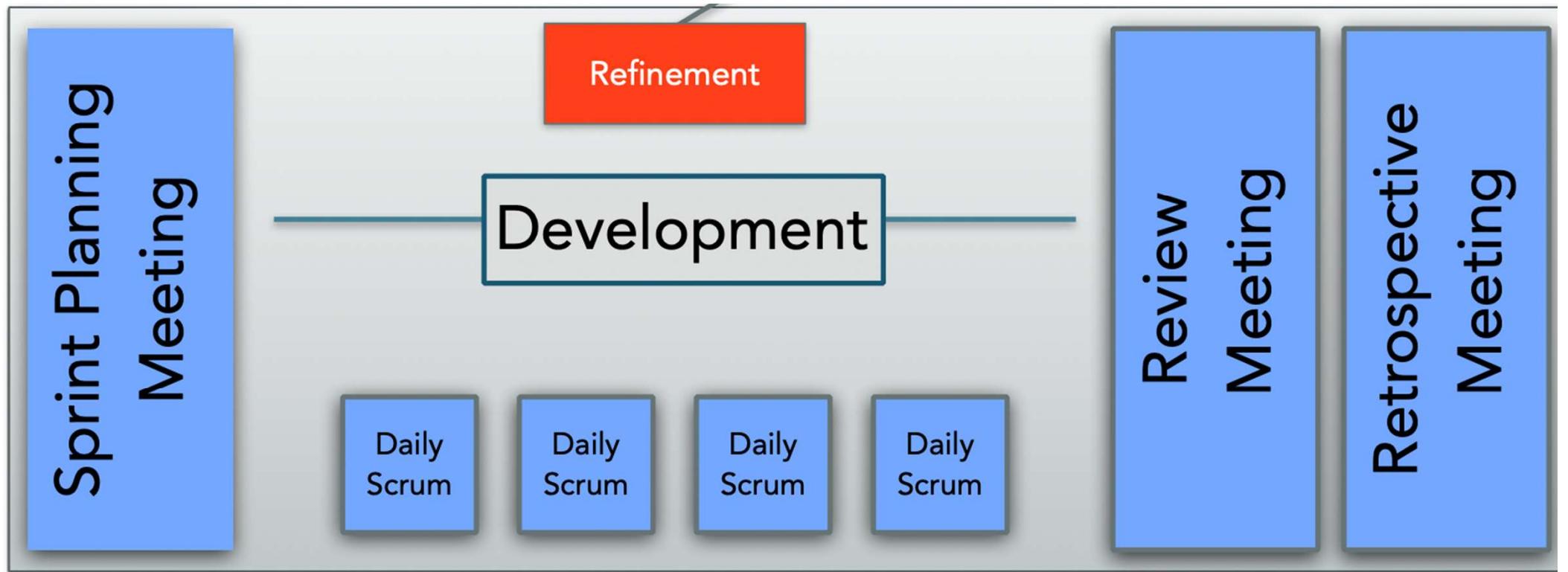
No entanto, um incremento pode ser entregue aos stakeholders antes do final da Sprint. A Sprint Review nunca deve ser considerada um marco para liberar valor. O trabalho não pode ser considerado parte de um incremento a menos que atenda a Definição de Pronto.



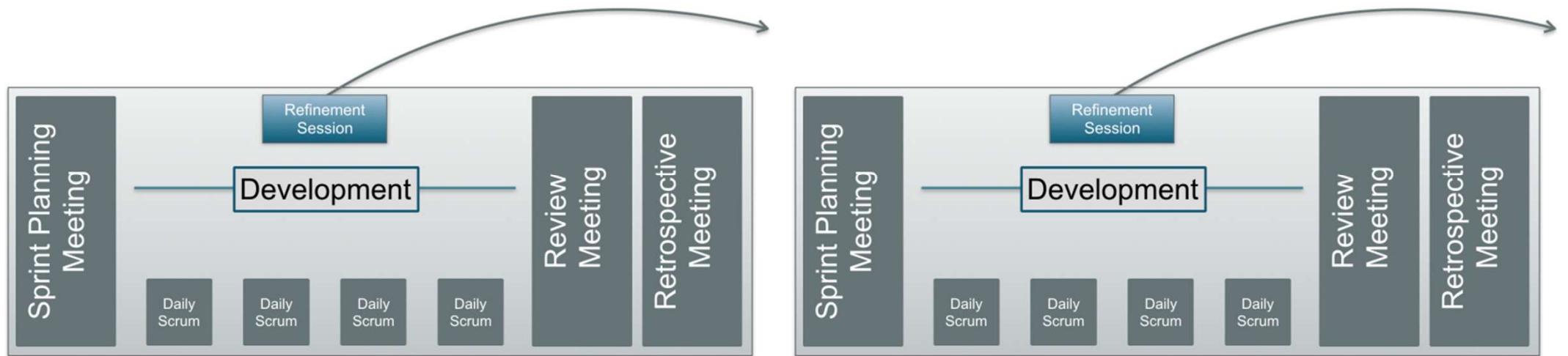
Visão Geral das Fases



Visão Geral das Fases



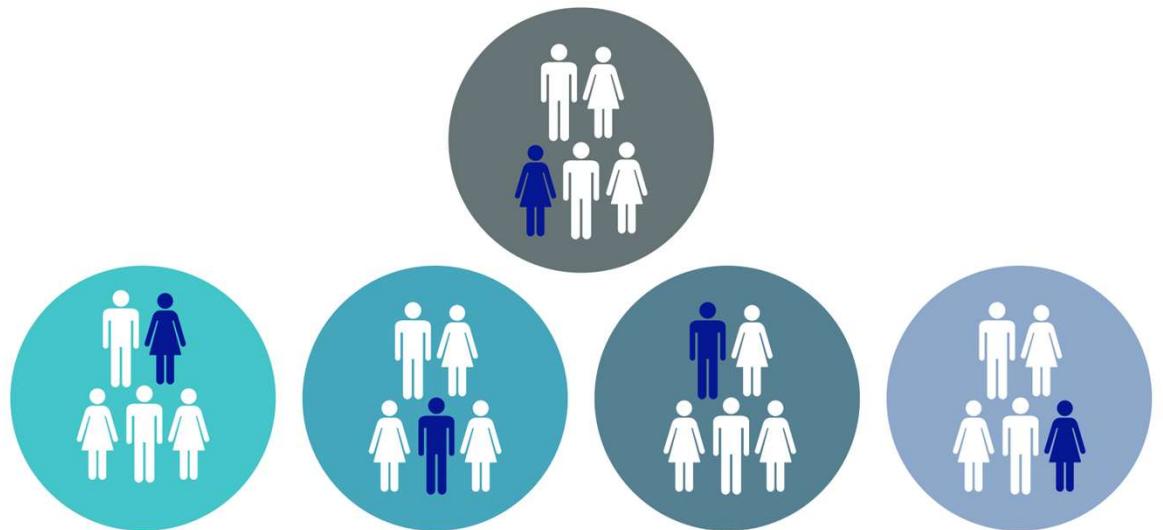
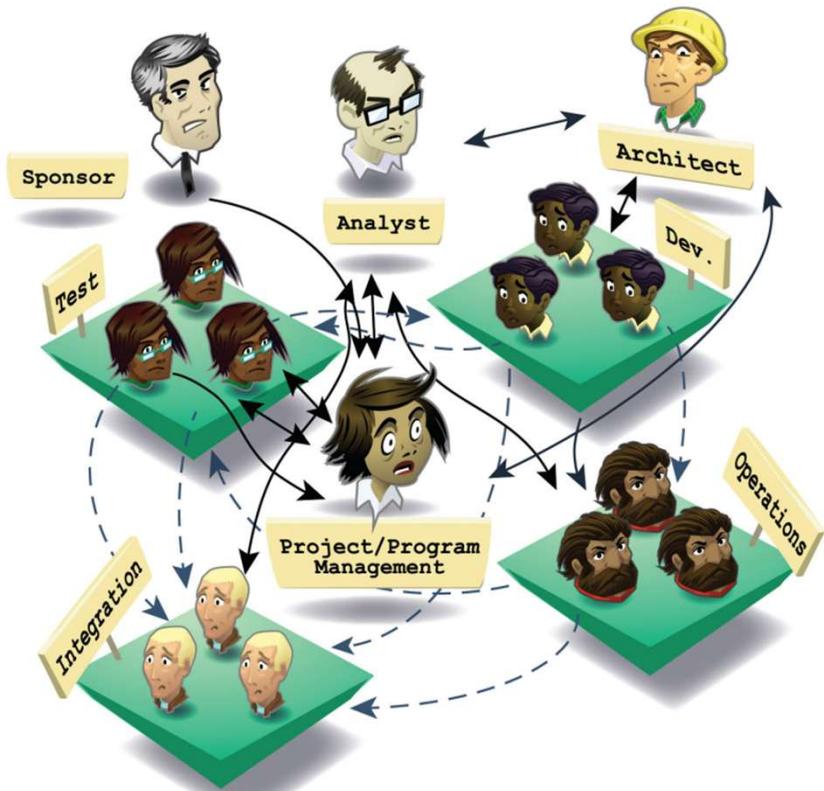
Visão Geral das Fases



Boas Práticas



Times Ponta a Ponta



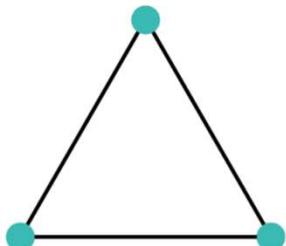
Times Ponta a Ponta

Modelo Tradicional		Modelo Ágil
<i>tendências:</i>		<i>tendências:</i>
times em silos	↔	times ponta-a-ponta
menos autonomia	↔	mais autonomia
maior custo de coordenação	↔	menor custo de coordenação
mais dependência	↔	menos dependência
maior leadtime	↔	menor leadtime
metas individuais	↔	metas compartilhadas
menor velocidade na tomada de decisão	↔	maior velocidade na tomada de decisão
baixa visão estratégica das equipes	↔	alta visão estratégica das equipes

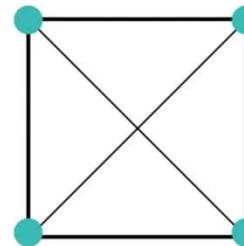
Times Auto Gerenciáveis.



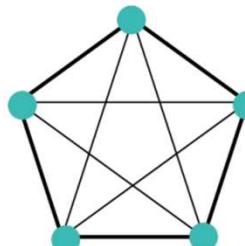
Times Menores



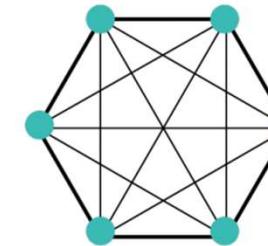
3 pessoas, 3 linhas



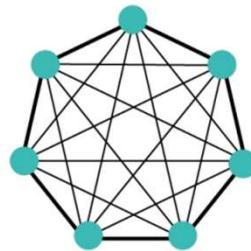
4 pessoas, 6 linhas



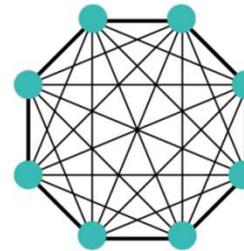
5 pessoas, 10 linhas



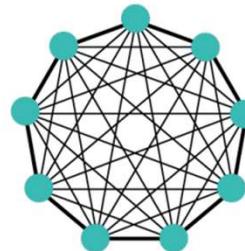
6 pessoas, 15 linhas



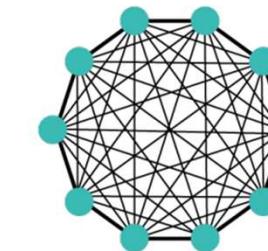
7 pessoas, 21 linhas



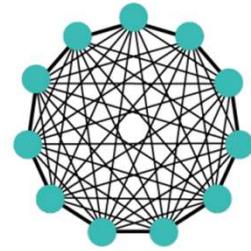
8 pessoas, 28 linhas



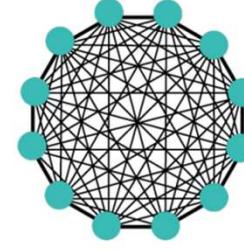
9 pessoas, 36 linhas



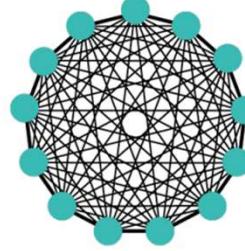
10 pessoas, 45 linhas



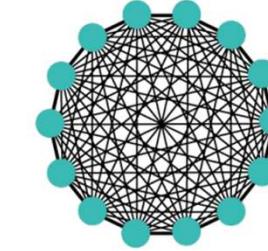
11 pessoas, 55 linhas



12 pessoas, 66 linhas



13 pessoas, 78 linhas

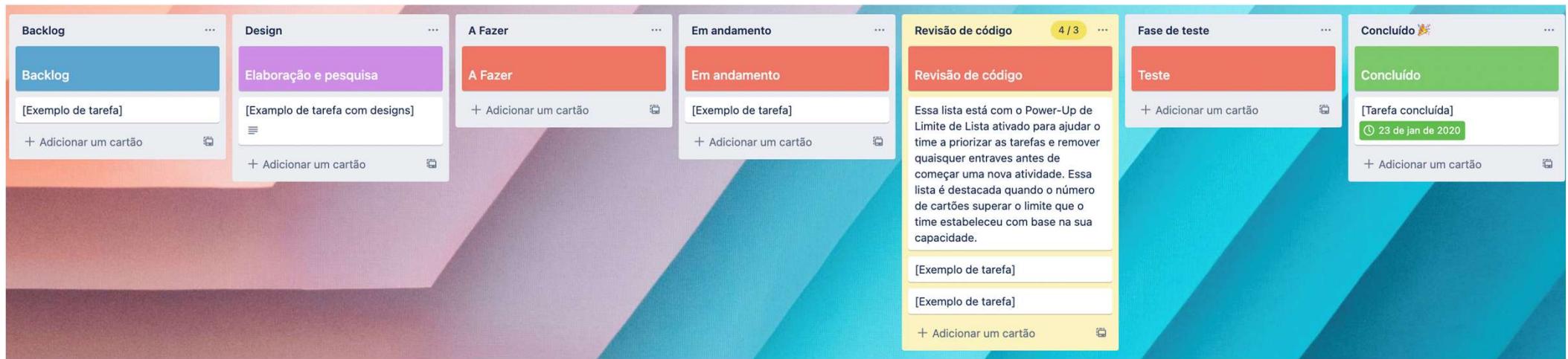


14 pessoas, 91 linhas

Gestão Visual



Gestão Visual



Prof. Anderson Augusto Bosing

Gestão Visual



Prof. Anderson Augusto Bosing

Aprendizagem Baseada em Equipe(Valor = 3.0 Entrega: 30/09/2024)

Temos diversas metodologias ágeis de desenvolvimento:
XP(eXtreme Programming).
FDD(Feature Driven Development).
Kanban.
Crystal.
MSF(Microsoft Solutions Framework).

Dividam-se em grupos de 4 pessoas.

Cada grupo deverá realizar uma pesquisa sobre as metodologias ágeis acima elencando componentes humanos envolvidos, fases do processo, contextualização geral, vantagens e desvantagens.

Cada grupo deverá gerar um página html contendo a pesquisa.
A página web deve ser estilizada conforme as decisões do grupo.

Valor = 3.0 pontos.



Análise e Projetos de Sistemas

Aula: Introdução a Requisitos de Software , Engenharia de Requisitos, Requisitos de Usuários e Sistema.



Prof. Anderson Augusto Bosing

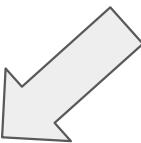
Como entender e atender as necessidades de um cliente?

O que são Requisitos?

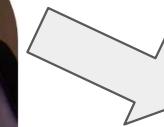
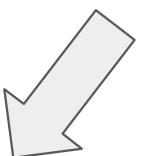
Caso você fosse comprar um PC. Quais seriam os Requisitos?



Placa de Vídeo RTX



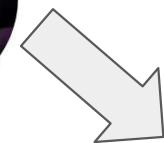
256 GB de SSD



Placa mãe



16 GB de memória RAM



O que são Requisitos?

Um requisito consiste em uma propriedade, característica ou comportamento que um produto deve atender.

E o que são Requisitos de Software ?

Um requisito de software é simplesmente uma declaração do que o sistema deve fazer ou de quais características ele precisa possuir.

**Quais serviços oferece.
Seus Atributos.
Restrições de Funcionamento.**



Gerenciamento de Requisitos

Etapa inicial no processo de desenvolvimento de softwares, fundamental para o sucesso do projeto.

É nessa etapa que são determinados:

- ✓ As funcionalidades e os perfis desejados por clientes e usuários.**
- ✓ O projeto dos desenvolvedores para atingir esses requisitos.**

O que é engenharia de requisitos?

O que é engenharia de requisitos?

O processo de descobrir, analisar, documentar e verificar esses serviços e restrições.(Sommerville, Engenharia de Software, 2011)

Engenharia de Requisitos é o nome que se dá ao conjunto de atividades relacionadas com a descoberta, análise, especificação e manutenção dos requisitos de um sistema.(Marco Túlio Valente. Engenharia de Software Moderna: Princípios e Práticas para Desenvolvimento de Software com Produtividade, 2020.)

E quais problemas a engenharia de requisitos resolve?

A falta de entendimento total ou parcial dos problemas que eles estavam resolvendo.

Os detalhes da solução abordada e a forma correta de resolver.

Aumento da qualidade de entrega do software e redução de custos de manutenções.

Estabelecendo e mantendo um acordo entre cliente/usuário e a equipe do projeto nas mudanças dos requisitos.

Classificações de Requisitos

Requisitos de Usuário

Segundo Sommerville:

Requisitos de Usuário – Descrevem os requisitos... de modo que eles sejam compreensíveis pelos usuários do sistema que não possuem conhecimento técnico detalhado.

Suponha, por exemplo, um sistema bancário. Um requisito de usuário — especificado pelos funcionários do banco — pode ser o seguinte: o sistema deve permitir transferências de valores para uma conta corrente de outro banco, por meio de TEDs.

Tipos de Requisitos

Requisitos de Sistema

Segundo Sommerville:

Requisitos de sistema são descrições mais detalhadas das funções, serviços e restrições operacionais do sistema de software. O documento de requisitos do sistema (às vezes, chamado especificação funcional) deve definir exatamente o que deve ser implementado. Pode ser parte do contrato entre o comprador do sistema e os desenvolvedores de software.

Utilizando o mesmo exemplo do Sistema Bancário Anterior.

Esse requisito dá origem a um conjunto de requisitos de sistema, os quais vão detalhar e especificar o protocolo a ser usado para realização de tais transferências entre bancos. Portanto, requisitos de usuário estão mais próximos do problema, enquanto que requisitos de sistema estão mais próximos da solução.

Os requisitos precisam ser escritos em diferentes níveis de detalhamento para que diferentes leitores possam usá-los de diversas maneiras.

Requisitos de usuário não costumam se preocupar com a forma como o sistema será implementado.

Requisitos de sistema precisam saber mais detalhadamente o que o sistema fará, porque estão interessados em como ele apoiará os processos dos negócios ou estão envolvidos na implementação do sistema.

Leitores de diferentes tipos de especificação de requisitos



Vamos Praticar ?

Exercício Requisitos de Usuário X Requisitos de Sistemas.

Exercício disponível em:

<https://docs.google.com/document/d/1zBKK9RrO3fBt44XGEbak3p5v8NG7yGVR/edit?usp=sharing&ouid=112890355844915434016&rtpof=true&sd=true>.



Requisitos de usuário e de sistema

Esse exemplo, de um Sistema de Gerenciamento da Saúde Mental de Pacientes (MHC-PMS, do inglês Mental Health Care Patient Management System)

1. O MHC-PMS deve gerar relatórios gerenciais mensais que mostrem o custo dos medicamentos prescritos por cada clínica durante aquele mês.

- 1.1 No último dia útil de cada mês deve ser gerado um resumo dos medicamentos prescritos, seus custos e as prescrições de cada clínica.
- 1.2 Após 17:30h do último dia útil do mês, o sistema deve gerar automaticamente o relatório para impressão.
- 1.3 Um relatório será criado para cada clínica, listando os nomes dos medicamentos, o número total de prescrições, o número de doses prescritas e o custo total dos medicamentos prescritos.
- 1.4 Se os medicamentos estão disponíveis em diferentes unidades de dosagem (por exemplo, 10 mg, 20 mg), devem ser criados relatórios separados para cada unidade.
- 1.5 O acesso aos relatórios de custos deve ser restrito a usuários autorizados por uma lista de controle de gerenciamento de acesso.

Vamos Praticar ?

Exercício Requisitos de Usuário X Requisitos de Sistemas.

Exercício disponível em:

<https://docs.google.com/document/d/1xzk4-B6Ftpls80j0iWZYtSaSi1a5jeN1/edit?usp=sharing&ouid=12890355844915434016&rtpof=true&sd=true>



Requisitos de usuário e de sistema

Esse exemplo, O sistema LIBSYS um sistema de biblioteca que fornece uma interface única para uma série de banco de dados de artigos em bibliotecas diferentes .

1.LIBSYS deve manter o acompanhamento de todos os dados exigidos pelas agências de licenciamento de direitos autorais no Reino Unido e em outros lugares .

- 1.1 Ao solicitar um documento ao LIBSYS , deve ser apresentado ao solicitante um formulário que registra os detalhes do usuário e da solicitação feita .
- 1.2 Os formulários de solicitação do LIBSYS devem ser armazenados no sistema durante cinco anos , a partir da data da solicitação .
- 1.3 Todos os formulários do LIBSYS devem ser indexados por usuário , nome do material solicitado e fornecedor da solicitação .
- 1.4 O LIBSYS deve manter um registro de todas as solicitações feitas ao sistema .
- 1.5 Para materiais aos quais se aplicam os direitos de empréstimo dos autores , os detalhes do empréstimo devem ser enviados mensalmente às agências de licenciamento de direitos autorais que se registraram no LIBSYS .

Leitores de diferentes tipos de especificação de requisitos

Os diferentes tipos de leitores de documento exibidos são exemplos de stakeholders do sistema. Incluem qualquer um que seja afetado de alguma maneira pelo sistema.

Exemplos:

- ✓ Médicos responsáveis por avaliar e tratar os pacientes.
- ✓ Recepcionistas que marcam as consultas dos pacientes;
- ✓ Equipe de TI responsável pela instalação e manutenção do sistema;

REVISANDO

- Engenharia de Requisitos.
- Problemas que a Engenharia de Requisitos Resolve.
- Requisitos de Sistema.
- Requisitos de Usuário.

Engenharia de Software I

**Aula: Requisitos Funcionais e
Não Funcionais.**

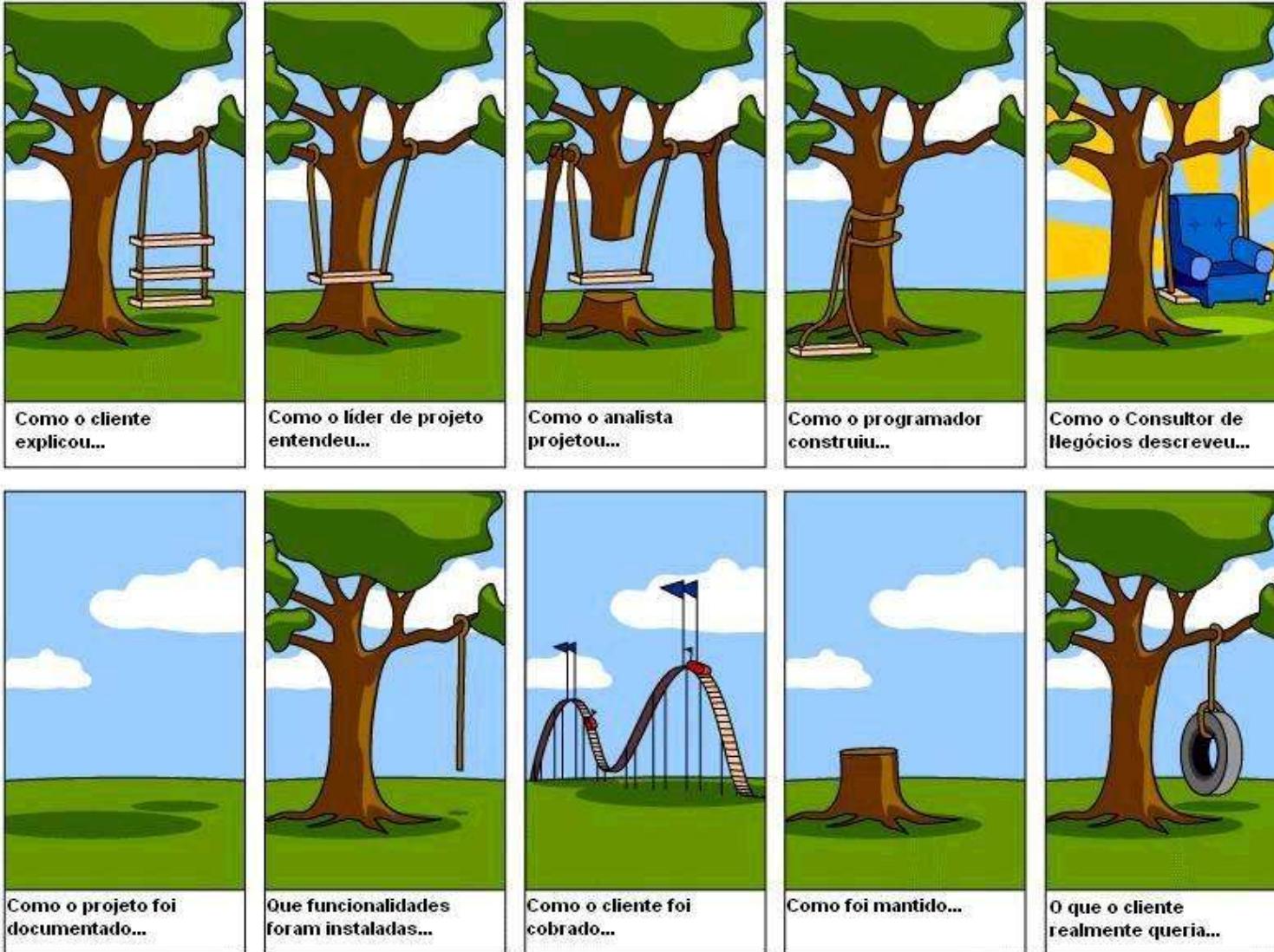


Prof. Anderson Augusto Bosing

Revisando a Ultima Aula

- **Como entender e atender as necessidades do cliente ?**
- **O que são Requisitos ?**
- **O que são Requisitos de Software ?**
- **Engenharia de Requisitos.**
- **Problemas que a Engenharia de Requisitos resolve.**
- **Requisitos de usuário x Requisitos de Sistema**
- **Leitores diferentes para cada um dos requisitos.**

Qual a motivação para gerenciar os requisitos ?



Requisitos funcionais e não funcionais

Os requisitos de software são frequentemente classificados como requisitos funcionais e requisitos não funcionais:

Requisitos funcionais. São declarações de serviços que o sistema deve fornecer, de como o sistema deve reagir a entradas específicas e de como o sistema deve se comportar em determinadas situações. Em alguns casos, os requisitos funcionais também podem explicitar o que o sistema não deve fazer.

Requisitos funcionais

Os requisitos funcionais de um sistema descrevem o que ele deve fazer.(Sommerville, Engenharia de Software, 2011).

Ex:

Incluir/Excluir/Alterar os dados de clientes.
Efetuar cobranças de compras efetuadas pelos clientes.
Consultar saldos de estoque de um produto.

Requisitos funcionais

Por exemplo, requisitos funcionais para o sistema MHC-PMS, usados para manter informações sobre os pacientes em tratamento por problemas de saúde mental:

- 1. Um usuário deve ser capaz de pesquisar as listas de agendamentos para todas as clínicas.**
- 2. O sistema deve gerar a cada dia, para cada clínica, a lista dos pacientes para as consultas daquele dia.**
- 3. Cada membro da equipe que usa o sistema deve ser identificado apenas por seu número de oito dígitos.**

Requisitos funcionais e não funcionais

Requisitos não funcionais. São restrições aos serviços ou funções oferecidos pelo sistema. Incluem restrições de timing, restrições no processo de desenvolvimento e restrições impostas pelas normas.

Ao contrário das características individuais ou serviços do sistema, os requisitos não funcionais, muitas vezes, aplicam-se ao sistema como um todo.

Requisitos Não Funcionais

Os requisitos não funcionais como o nome sugere, são requisitos que não estão diretamente relacionados com os serviços específicos oferecidos pelo sistema a seus usuários. Eles podem estar relacionados às propriedades emergentes do sistema, como confiabilidade, tempo de resposta, desempenho, plataformas suportadas.(Sommerville, Engenharia de Software, 2011)

Requisitos Não Funcionais

Tipos:

- Aparência
- Usabilidade
- Desempenho
- Operacional
- Manutenção e suporte
- Segurança
- Cultura organizacional
- Legal

Requisitos Não Funcionais

Aparência

- O produto deve ter um estilo igual ao dos outros produtos da empresa.
- O produto deve ser atrativo para usuários adolescentes.
- O produto deve ser identificável com a empresa onde será usado.

Requisitos Não Funcionais

Usabilidade

- O produto deve limpar o campo e exibir mensagem de erro quando o usuário entrar com dados incorretos. (facilidade de uso)
- O produto deve ser especialmente intuitivo de usar para crianças com 4 anos de idade. (facilidade de aprendizagem)
- O produto deve ser apresentado ao usuário na língua portuguesa e inglesa. (personalização)

Requisitos Não Funcionais

Desempenho

- O produto deve identificar o funcionário com base na foto em menos de 3 segundos. (tempo de resposta)
- O produto deve trabalhar em modo local, se perder a conexão com o servidor (disponibilidade).
- O produto deve calcular os juros até os décimos de centavos. (precisão dos resultados)

Requisitos Não Funcionais

Operacional

- O produto deve operar debaixo de água até a profundidade de 30 metros.
- O produto deve carregar os dados em lote por meio de arquivos texto.
- O produto deve exportar o *curriculum vitae* no formato PDF.

Requisitos Não Funcionais

Manutenção e suporte

- O programa do produto deve conter comentários.
- O produto deve estar preparado para ser utilizado em qualquer língua.
- O produto deve disponibilizar um tutorial que explique como operá-lo.

Requisitos Não Funcionais

Segurança

- Os dados da avaliação de desempenho de um funcionário devem ser fornecidos apenas ao próprio funcionário e aos seus superiores.
- O produto deve garantir que só usuários registrados tenham acesso aos dados clínicos dos pacientes.
- O produto deve solicitar senha de acesso com no mínimo 8 dígitos, que contenha letras, números e caracteres especiais.

Requisitos Não Funcionais

Cultural e organizacional

- O produto deve usar português do Brasil.**
- O produto deve mostrar os feriados locais no calendário.**
- O produto deve usar componentes fabricados no Mercosul.**

Requisitos Não Funcionais

Legal

- O produto deve estar alinhado ao referencial de gestão do PMBOK.
- O produto deve ser certificado pela Autoridade Tributária e Aduaneira.
- O produto deve atender às normas estabelecidas na Lei nº 8112/90.

Requisitos Não Funcionais

Ex:

Uso de Design responsivo nas interfaces gráficas(Possibilita o uso em diferentes dispositivos como tablets, smartphones e computadores).

Compatibilidade com sistemas operacionais Windows e Linux.

Deve possuir conformidade com as normativas da LGPD.

Requisitos Não Funcionais

Por exemplo, requisitos não funcionais para o sistema MHC-PMS, usados para manter informações sobre os pacientes em tratamento por problemas de saúde mental: O MHC-PMS deve estar disponível para todas as clínicas durante as horas normais de trabalho (segunda a sexta-feira, 8h30 às 17h30). Períodos de não operação dentro do horário normal de trabalho não podem exceder cinco segundos em um dia.

Requisitos Não Funcionais

Usuários do sistema MHC-PMS devem se autenticar com seus cartões de identificação da autoridade da saúde.

O sistema deve implementar as disposições de privacidade dos pacientes, tal como estabelecido no HStan-03-2006-priv.

Requisitos Não Funcionais

A equipe médica deve ser capaz de usar todas as funções do sistema após quatro horas de treinamento.

Após esse treinamento, o número médio de erros cometidos por usuários experientes não deve exceder dois por hora de uso do sistema.

Requisito de software - Classificação

Podem ser classificados ainda, seja funcional ou não funcional como:

- **Requisitos primários** –provém diretamente de alguma parte interessada, ou seja, foi solicitado por uma pessoa ou entidade.
- **Requisitos secundários** –são obtidos por refinamento de um requisito primário .

Requisito de software - Classificação

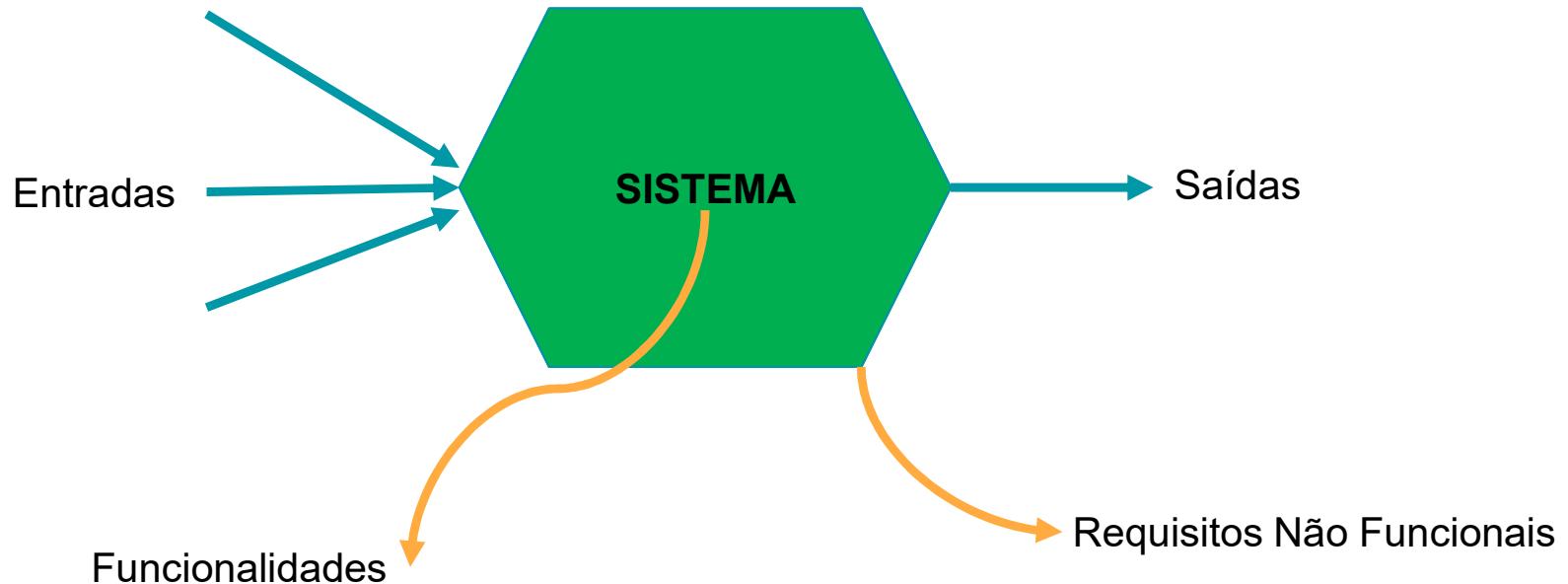
Requisitos não funcionais podem ser divididos em:

- Requisitos de produto – caracterizam aspectos do funcionamento do produto, ex: confiabilidade, desempenho, eficiência, portabilidade, usabilidade, testabilidade, manutenibilidade.
- Requisitos organizacionais – provém de estratégias e procedimento estabelecidos no contexto do fabricante do produto ou da organização do cliente, ex: normas a serem seguidas, requisitos de implementação, como a linguagem de programação a usar.
- Requisitos externos – tem origem em fatores externos ao produto ou ao desenvolvimento, ex: requisitos de interoperabilidade que definem como os produtos interagem com outros sistemas, requisitos legais, requisitos éticos.

Métricas para especificar requisitos não funcionais.

Propriedade	Medida
Velocidade	Transações processadas/segundo Tempo de resposta de usuário/evento Tempo de atualização de tela
Tamanho	Megabytes Número de chips de memória ROM
Facilidade de uso	Tempo de treinamento Número de frames de ajuda
Confiabilidade	Tempo médio para falha Probabilidade de indisponibilidade Taxa de ocorrência de falhas Disponibilidade
Robustez	Tempo de reinício após falha Percentual de eventos que causam falhas Probabilidade de corrupção de dados em caso de falha
Portabilidade	Percentual de declarações dependentes do sistema-alvo Número de sistemas-alvo

O que os Requisitos de Software Especificam?



Gerenciamento de Requisitos Não é Fácil, Porque...?

- Nem sempre são óbvios
- Podem vir de muitas fontes
- Nem sempre é fácil de expressar claramente com Palavras.
- Relatado por um e entregue por outro no processo de engenharia de software.
- Tem propriedade única.
- Mudam.
- Em grande quantidade são difíceis de controlar.

Qualidades do Requisito de Software

- **Verificável.**
- **Priorizável por importância.**
- **Modificável.**
- **Rastreável.**
- **Compreensível.**
- **Correto.**
- **Completo.**
- **Consistente.**
- **Não Ambíguo.**
- **Testável.**

Como especificar Requisitos ?

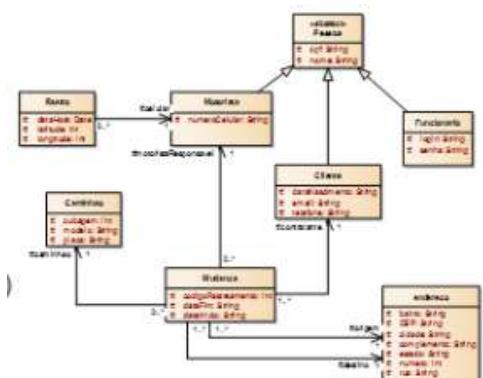
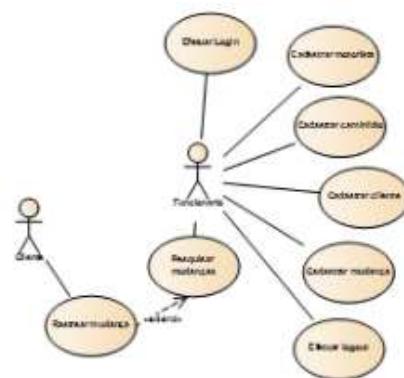


Como especificar Requisitos ?

- Como funcionário gostaria de consultar os motoristas da empresa disponíveis para realizar uma mudança.
 - Como funcionário gostaria de cadastrar os caminhões da empresa para controlar a frota.
 - Como funcionário gostaria de registrar o contrato de mudança para controlar os serviços da empresa.
 - Como cliente gostaria de rastrear a minha mudança para saber quando chegará ao destino.

- O sistema deve permitir ao funcionário cadastrar um motorista da empresa.
 - O sistema deve permitir ao funcionário cadastrar um caminhão da empresa.
 - O sistema deve permitir ao funcionário cadastrar um cliente.
 - O sistema deve permitir ao funcionário cadastrar um serviço de mudança contra
 - O sistema deve permitir ao funcionário e o cliente rastrear uma mudança.

Linguagem Natural



Como especificar Requisitos ? Linguagem Natural.

A linguagem natural é a forma de documentação de requisitos mais aplicada na prática.

Vantagens:

Nenhum stakeholder precisa aprender uma nova notação.
Qualquer requisito pode ser expresso em linguagem natural.

Desvantagens:

Pode resultar em requisitos ambíguos.
Requisitos de diferentes tipos pode ser misturados.
Em ambas as situações acima podem ocorrer mesmo que não intencionalmente.

Como especificar Requisitos ? Linguagem Conceitual.

Ao documentar requisitos por meio de modelos, linguagens de notação especiais para modelagem devem ser utilizadas(UML, BPMN).

Vantagens:

Modelos retratam os requisitos de forma mais compacta, sendo de compreensão mais fácil para um leitor treinado.

Oferecem menor grau de ambiguidade devido ao seu maior grau de formalidade.

Desvantagens:

Exigem conhecimentos específicos de modelagem.

Retratam apenas uma perspectiva específica.

VAMOS PRATICAR ?

Levante alguns requisitos com base na descrição abaixo:

O Professor Anderson precisa de um sistema para que sejam armazenadas as notas e os dados dos alunos. Assim que fechada a média final os alunos devem receber um e-mail automaticamente com a sua média. O Professor Anderson deseja utilizar este software tanto no seu notebook quanto em seu smartphone android.



Correção do Exercício Anterior:

“O Professor Anderson precisa de um sistema para que sejam armazenadas as notas e os dados dos alunos. Assim que fechada a média final os alunos deve receber um e-mail automaticamente com a sua média. Ele deseja utilizar este software tanto no seu notebook quanto em seu smartphone android.”

- 1. Cadastrar Alunos:** O sistema deve permitir o cadastro de alunos, contendo informações como nome, e-mail e outras informações pessoais. **Cadastro de Alunos:** O sistema deve permitir o cadastro de alunos, contendo informações como nome, e-mail e outras informações pessoais.
- 2. Gerenciar Notas:** O sistema deve permitir que o professor gerencie as notas de cada aluno por disciplina.
- 3. Cálculo da Média Final:** O sistema deve calcular a média final dos alunos automaticamente com base nas notas inseridas.

Correção do Exercício Anterior:

“O Professor Anderson precisa de um sistema para que sejam armazenadas as notas e os dados dos alunos. Assim que fechada a média final os alunos deve receber um e-mail automaticamente com a sua média. Ele deseja utilizar este software tanto no seu notebook quanto em seu smartphone android.”

- 4. Envio de E-mails:** O sistema deve enviar automaticamente um e-mail para cada aluno com a sua média final assim que ela for fechada.
- 5. Sistema deve ser híbrido(WEB)** e possível de ser utilizado em diversas plataformas como desktop e mobile.

Atividade de Levantamento de Requisitos de software



Engenharia de Software I

**Aula: Exercício e O
documento de requisitos de
software.**



Prof. Anderson Augusto Bosing

Revisando a Última Aula

- Requisitos funcionais e Não funcionais
- Qualidades de um bom Requisito.

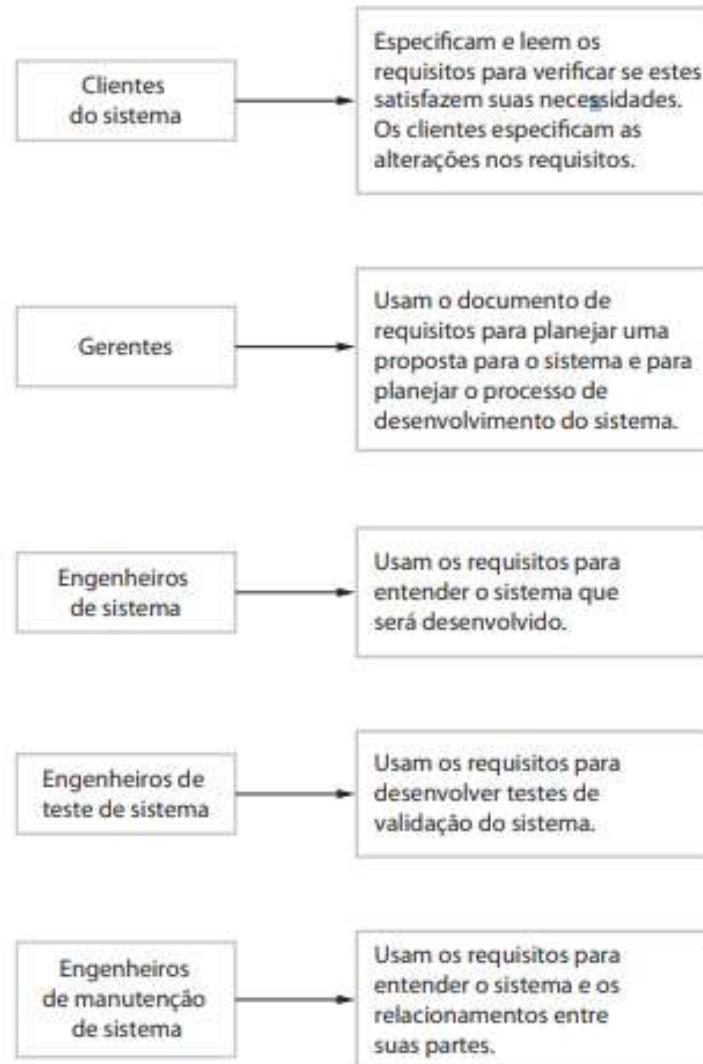
O Documento de Requisitos de software

O documento de requisitos de software, às vezes chamado Especificação de Requisitos de Software (SRS — do inglês Software Requirements Specification), é uma declaração oficial de o que os desenvolvedores do sistema devem implementar. Deve incluir tanto os requisitos de usuário para um sistema quanto uma especificação detalhada dos requisitos de sistema.

O Documento de Requisitos de software

O documento de requisitos tem um conjunto diversificado de usuários, que vão desde a alta administração da organização que está pagando pelo sistema até os engenheiros responsáveis pelo desenvolvimento do software. A Figura a seguir, mostra os possíveis usuários do documento e como eles o utilizam.

O Documento de Requisitos de software



A estrutura de um documento de requisitos.

Prefácio	Deve definir os possíveis leitores do documento e descrever seu histórico de versões, incluindo uma justificativa para a criação de uma nova versão e um resumo das mudanças feitas em cada versão.
Introdução	Deve descrever a necessidade para o sistema. Deve descrever brevemente as funções do sistema e explicar como ele vai funcionar com outros sistemas. Também deve descrever como o sistema atende aos objetivos globais de negócio ou estratégicos da organização que encomendou o software.
Glossário	Deve definir os termos técnicos usados no documento. Você não deve fazer suposições sobre a experiência ou o conhecimento do leitor.
Definição de requisitos de usuário	Deve descrever os serviços fornecidos ao usuário. Os requisitos não funcionais de sistema também devem ser descritos nessa seção. Essa descrição pode usar a linguagem natural, diagramas ou outras notações compreensíveis para os clientes. Normas de produto e processos que devem ser seguidos devem ser especificados.
Arquitetura do sistema	Deve apresentar uma visão geral em alto nível da arquitetura do sistema previsto, mostrando a distribuição de funções entre os módulos do sistema. Componentes de arquitetura que são reusados devem ser destacados.

A estrutura de um documento de requisitos.

Especificação de requisitos do sistema	Deve descrever em detalhes os requisitos funcionais e não funcionais. Se necessário, também podem ser adicionados mais detalhes aos requisitos não funcionais. Interfaces com outros sistemas podem ser definidas.
Modelos do sistema	Pode incluir modelos gráficos do sistema que mostram os relacionamentos entre os componentes do sistema, o sistema e seu ambiente. Exemplos de possíveis modelos são modelos de objetos, modelos de fluxo de dados ou modelos semânticos de dados.
Evolução do sistema	Deve descrever os pressupostos fundamentais em que o sistema se baseia, bem como quaisquer mudanças previstas, em decorrência da evolução de hardware, de mudanças nas necessidades do usuário etc. Essa seção é útil para projetistas de sistema, pois pode ajudá-los a evitar decisões capazes de restringir possíveis mudanças futuras no sistema.
Apêndices	Deve fornecer informações detalhadas e específicas relacionadas à aplicação em desenvolvimento, além de descrições de hardware e banco de dados, por exemplo. Os requisitos de hardware definem as configurações mínimas ideais para o sistema. Requisitos de banco de dados definem a organização lógica dos dados usados pelo sistema e os relacionamentos entre esses dados.
Índice	Vários índices podem ser incluídos no documento. Pode haver, além de um índice alfabético normal, um índice de diagramas, de funções, entre outros pertinentes.

VAMOS PRATICAR ?

**Com base no exercício feito na aula anterior
mapeie no documento os requisitos funcionais
e não funcionais.**

VAMOS PRATICAR ?

Pesquise e elabore um resumo sobre as seguintes técnicas de elicitação de requisitos:

- **Entrevista**
- **Questionário**
- **Análise de Documentos**
- **Cenários**
- **Observação direta (etnografia / shadowing)**
- **Brainstorming**
- **JAD (Joint Application Design / Development**

Para cada técnica:

Descrição da técnica

Quando aplicar

Vantagens Limitações

Exemplo de uso real ou hipotético

VAMOS PRATICAR ?



Engenharia de Software I

**Aula: Técnicas de Elicitação
de Requisitos**



Prof. Anderson Augusto Bosing

Técnicas de Elicitação

As “primeiras perguntas” darão somente um entendimento básico do problema;

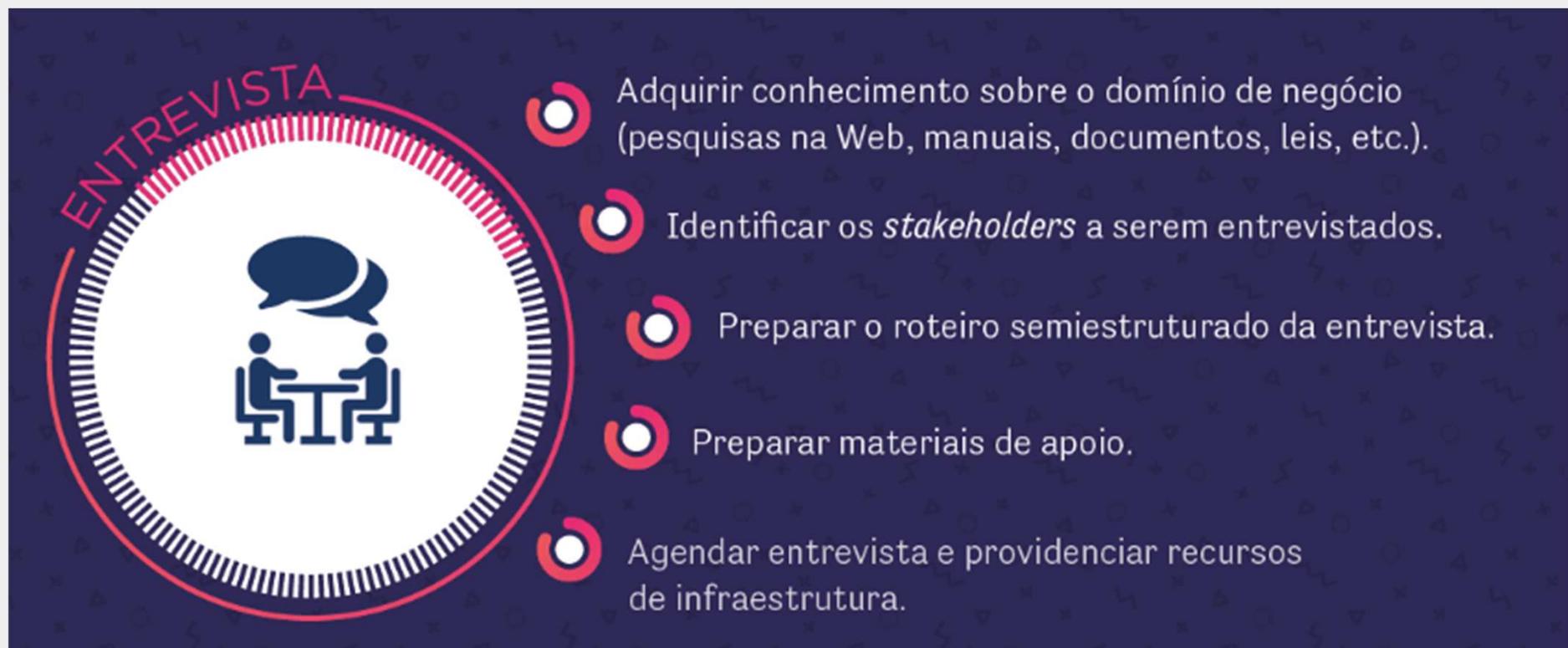
Diversas técnicas podem ser utilizadas no levantamento de requisitos, as quais podem possuir diferentes objetos de investigação ou podem ter foco em tipos diferentes de requisitos.

Assim, é útil empregar várias dessas técnicas concomitantemente, de modo a se ter um levantamento de requisitos mais eficaz. Dentre as várias técnicas, podem ser citadas:

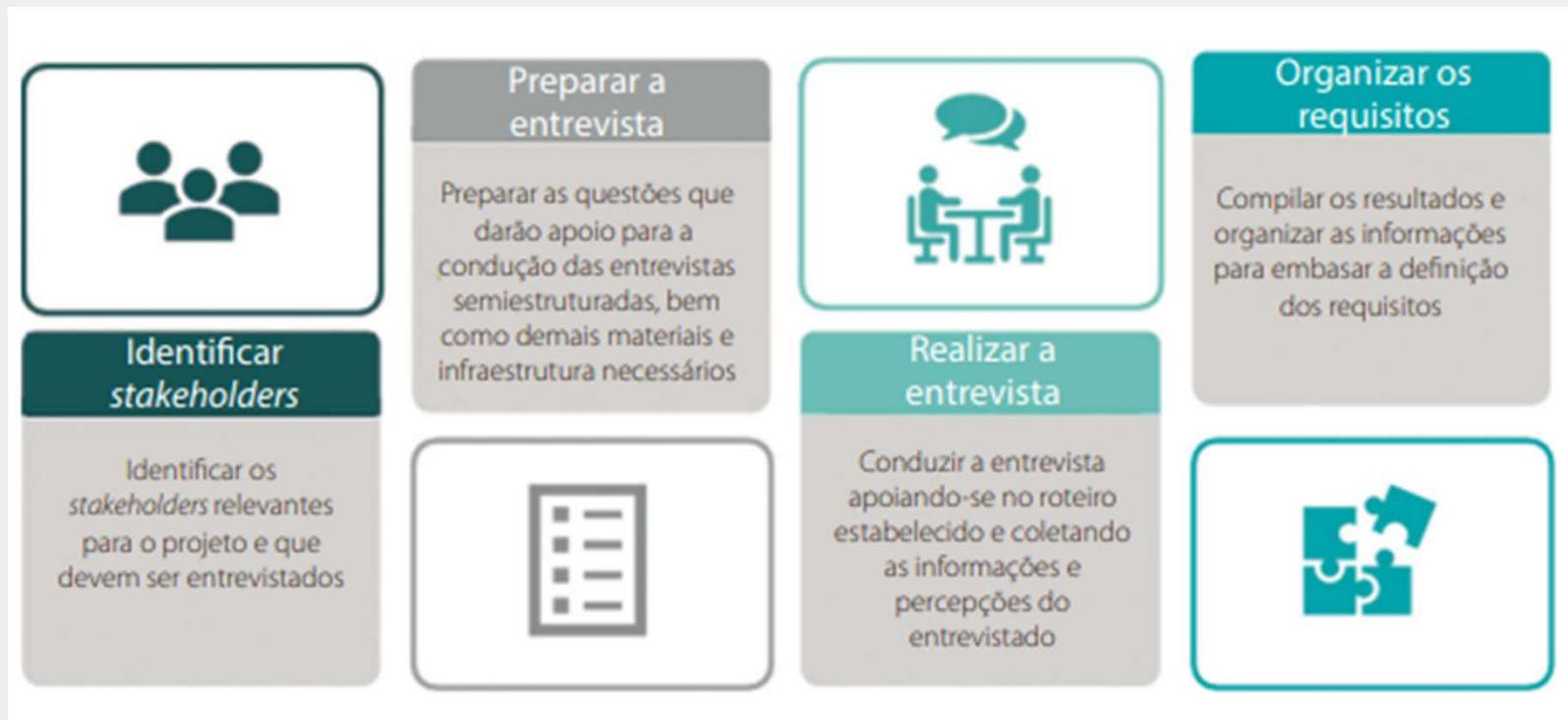
Entrevista;
Questionário;
Cenários.
Prototipação;
Análise de Documentos;

Entrevistas

Técnica amplamente utilizada, que consiste em conversas direcionadas com um propósito específico e com formato “pergunta-resposta”. Seu objetivo é descobrir problemas a serem tratados, levantar procedimentos importantes e saber a opinião e as expectativas do entrevistado sobre o sistema.

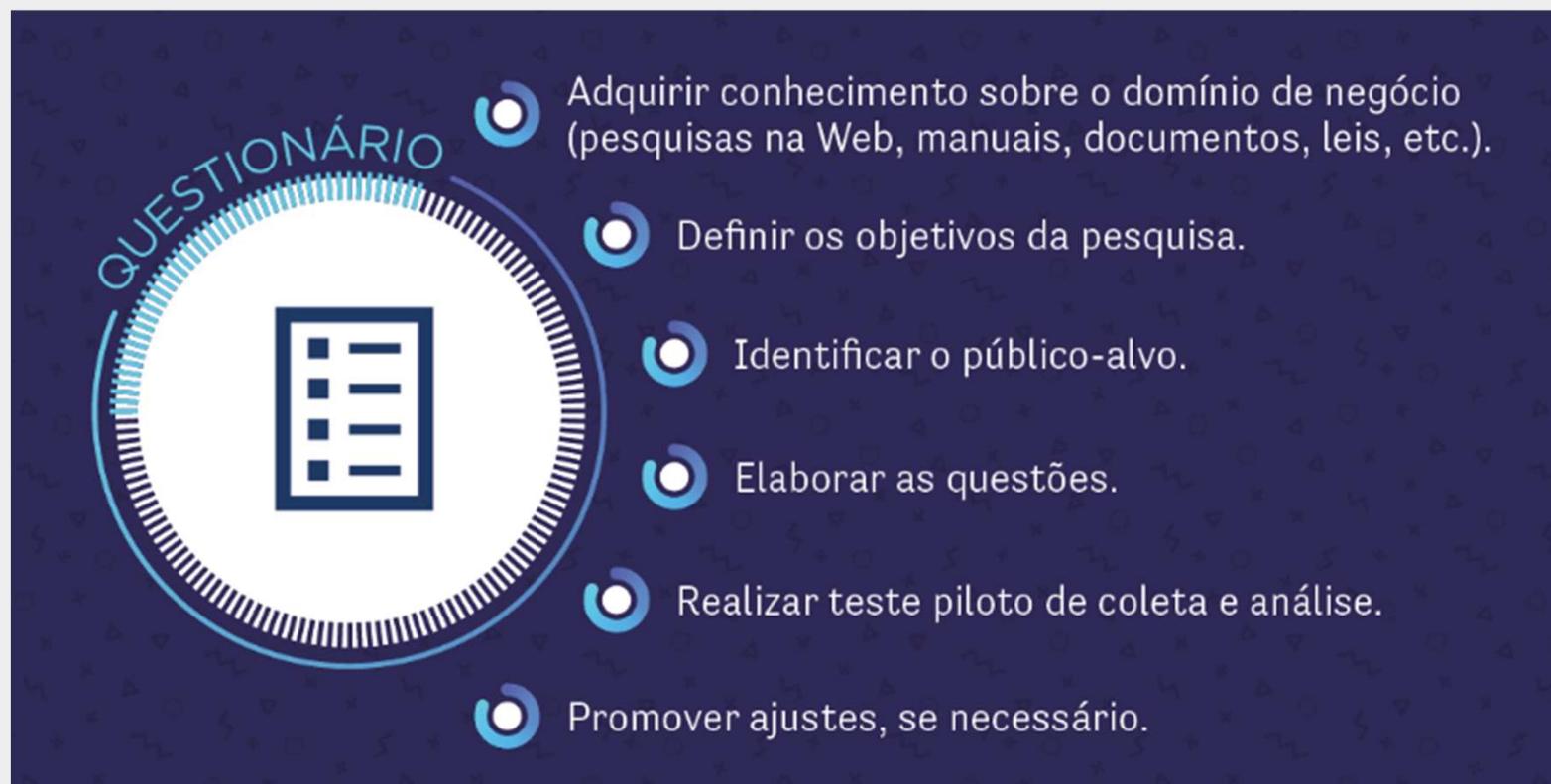


Entrevistas

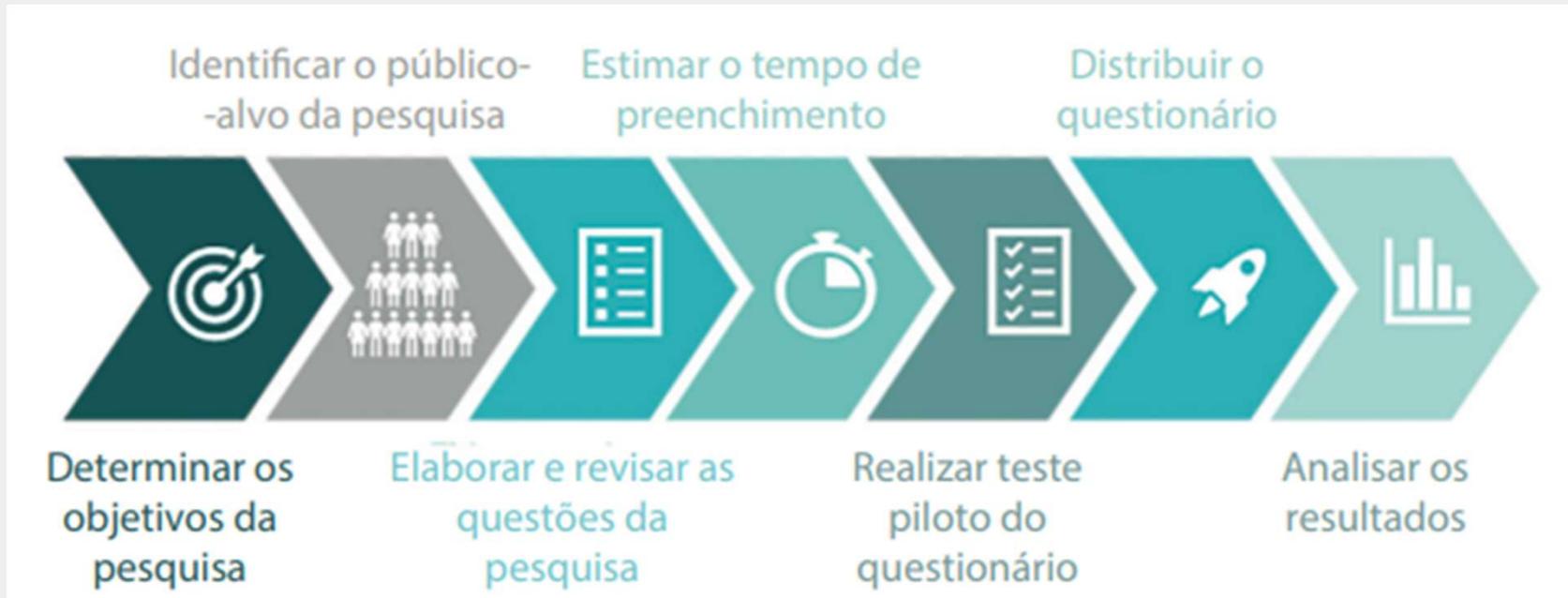


Questionários

O uso de questionários possibilita ao analista obter informações como postura, crenças, comportamentos e características de várias pessoas que serão afetadas pelo sistema.



Questionários



Análise de documentos

Pela análise de documentos existentes na organização, analistas capturam informações e detalhes difíceis de conseguir por entrevista e observação. Documentos revelam um histórico da organização e sua direção.

Cenários

Com o uso desta técnica, um cenário de interação entre o usuário final e o sistema é montado e o usuário simula sua interação com o sistema nesse cenário, explicando ao analista o que ele está fazendo e de que informações ele precisa para realizar a tarefa descrita no cenário. O uso de cenários ajuda a entender requisitos, a expor o leque de possíveis interações e a revelar facilidades requeridas.

Prototipagem

Um protótipo é uma versão preliminar do sistema, muitas vezes não operacional e descartável, que é apresentada ao usuário para capturar informações específicas sobre seus requisitos de informação, observar reações iniciais e obter sugestões, inovações e informações para estabelecer prioridades e redirecionar planos.

Principal / Livros Cadastrados / Novo Livro

Cadastro de Livro

Nome do Livro ISBN Editora
Novatec

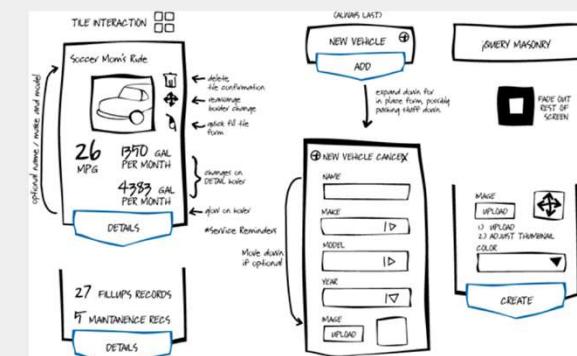
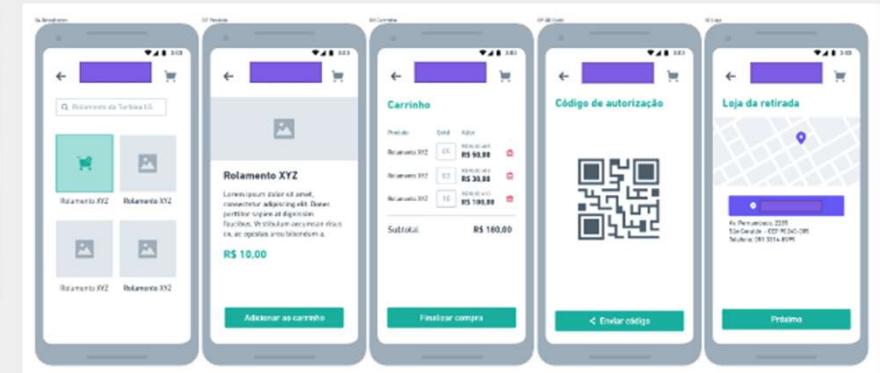
Categoria Autor Ano de Publicação
Novatec / /

Imagens Nova Imagem Tamanho mínimo: 800x600px

Quantidade Produto Ativo

Voltar Salvar Limpar

Principal
Livros
Categorias
Editoras
Relatórios
Usuários
Vendas
Estoque



Atividade de Levantamento de Requisitos de software



Engenharia de Software I

Aula: User Story.



Prof. Anderson Augusto Bosing

Histórias de Usuário (User Stories)

Maneira utilizada pelos Métodos Ágeis para documentar requisitos/casos de uso do sistema;

- Uma história de usuário é uma descrição em linguagem natural e informal dos recursos de um sistema de software. Elas são escritas da perspectiva de um usuário final ou usuário de um sistema e podem ser registradas em cartões, notas post-its ou digitalmente no software de gerenciamento do projeto.
- Dependendo do projeto, as histórias de usuário podem ser escritas por diferentes partes interessadas, como cliente, usuário, gerente ou equipe de desenvolvimento

Histórias de Usuário (User Stories)

As histórias de usuário são um tipo de objeto de fronteira. Eles facilitam a criação de sentido e a comunicação; e pode ajudar as equipes de software a documentar sua compreensão do sistema e seu contexto.

- As histórias de usuário são escritas por ou para usuários ou clientes para descrever a funcionalidade do sistema que está sendo desenvolvido. Em algumas equipes, o gerente de produto (ou proprietário do produto no Scrum) é o principal responsável por formular histórias de usuário e organizá-las em um backlog do produto.
- Em outras equipes, qualquer pessoa pode escrever uma história de usuário. As histórias de usuários podem ser desenvolvidas por meio de discussão com as partes interessadas, baseadas em personas ou simplesmente inventadas.

Histórias de Usuário (User Stories)

Writing a user story

- 1 Define your **end user**
- 2 Specify what **they want**
- 3 Describe **the benefit**
- 4 Add **acceptance criteria**

Histórias de Usuário (User Stories)

Como um <PAPEL>
posso/devo/gostaria/quero
<FUNÇÃO>
Para/de
<RESULTADO para o NEGÓCIO>

Histórias de Usuário (User Stories)

Como um vendedor, gostaria de consultar o estoque de um determinado produto para oferecer a um cliente

Como um vendedor, devo registrar a venda realizada para um cliente para manter o histórico de vendas

Como um diretor, gostaria de obter o volume de vendas do mês para acompanhar o atingimento das metas

Histórias de Usuário (User Stories)

Para definir boas Stories elas devem ser:

I independent

Uma User Story representa basicamente a visão do usuário no projeto, sendo assim, deve conter informação suficiente sem depender da finalização de outras. Podendo ser assimilada por qualquer pessoa e não necessitando de nenhuma fonte externa ou da leitura de outras User Stories para ser compreendida.

N negotiable

Uma boa Story deve ser flexível. Seus detalhes devem ser discutidos entre todos os envolvidos no projeto. Esta característica compreende às "conversas" dos 3 C's (cartão, conversa e confirmação) das User Stories.

V valuable

A User Story deve fornecer valor para o cliente, para a equipe do projeto ou para o produto. O valor agregado a ela contribui para priorizar o backlog, se alguma delas não possuirem um valor associado, devem ser colocadas no final da lista.

E estimable

Deve reunir um número de detalhes suficiente através das conversas, para que seja estimado o seu tempo total de realização, o custo de desenvolvimento, e outras métricas que sejam necessárias. Caso a estória não possa ser estimada, é conveniente reavaliá-la e se necessário dividi-la.

S small

Boas estórias de Usuários são pequenas, normalmente menores que o trabalho de uma pessoa por um mês. Quanto menor é a estória, mais provável é a precisão das estimativas e da estória ser entendida.

T testable

Uma estória deve ser testável para que seja aceita. O time deve ser capaz de testá-la, apenas com o critério de aceitação que foi descrito na estória.

Histórias de Usuário (User Stories)

Para definir boas Stories elas devem conter critérios de aceitação.

- Critérios de aceitação são definidos como "notas sobre o que a história deve fazer para que o proprietário do produto a aceite como completa."
- Eles definem os limites de uma história de usuário e são usados para confirmar quando uma história está concluída e funcionando como pretendido.
- A quantidade apropriada de informações a serem incluídas nos critérios de aceitação varia de acordo com a equipe, programa e projeto.
- Para que uma história seja considerada concluída ou concluída, todos os critérios de aceitação devem ser atendidos

Histórias de Usuário (User Stories) vs Casos de Uso

- Apesar de uma user story deixar claro que uma funcionalidade é requisitada no sistema por seus usuários, ela não dá ideia de como a funcionalidade vai ser (i.e., como o sistema vai se comportar), enquanto que os casos de uso dão essa ideia.
- No geral, uma user story é só uma forma de registrar a existência de um requisito, sem muitos detalhes, enquanto que um caso de uso é a documentação do resultado de se pensar em como aquele requisito vai ser tratado pelo sistema na prática.
- User stories são centradas no resultado e no benefício da coisa que se está descrevendo, enquanto casos de uso são mais granulares e descrevem como o sistema irá agir.
- User stories são apenas o início do processo de entendimento dos requisitos do sistema. Para os desenvolvedores que implementarão o sistema, casos de uso parecem um jeito muito melhor de descrever requisitos do que user stories. Mas, descrever casos de uso demanda esforço.

Atividade de Levantamento de Requisitos de software



Engenharia de Software I

**Aula: Exercício e O
documento de requisitos de
software.**



Prof. Anderson Augusto Bosing

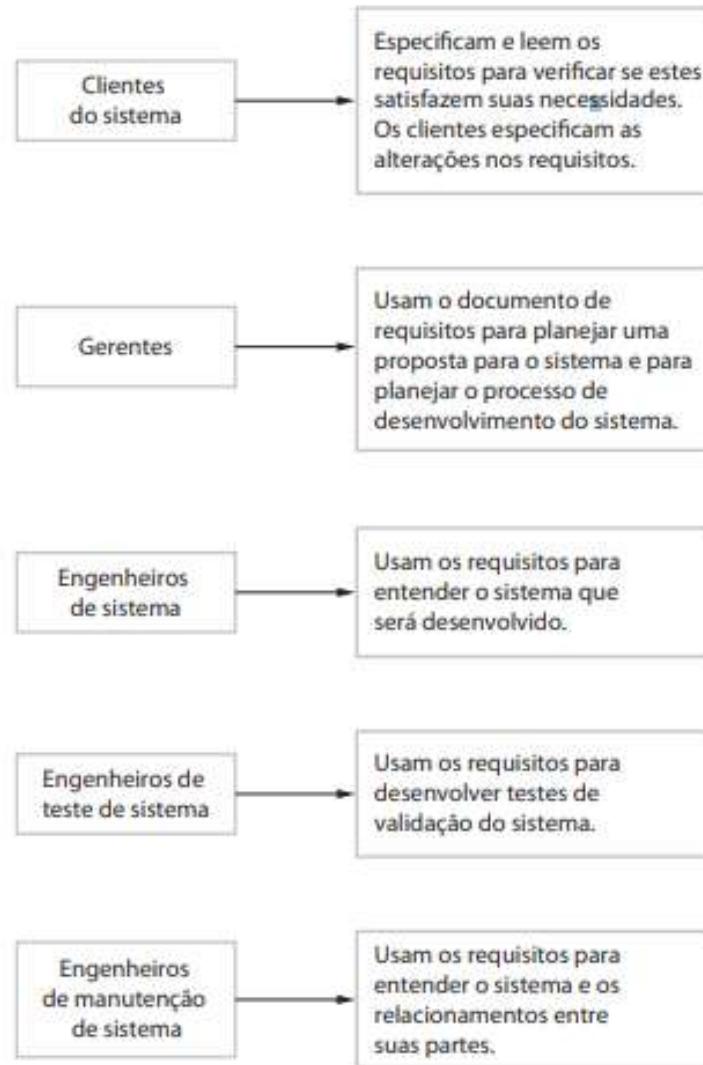
O Documento de Requisitos de software

O documento de requisitos de software, às vezes chamado Especificação de Requisitos de Software (SRS — do inglês Software Requirements Specification), é uma declaração oficial de o que os desenvolvedores do sistema devem implementar. Deve incluir tanto os requisitos de usuário para um sistema quanto uma especificação detalhada dos requisitos de sistema.

O Documento de Requisitos de software

O documento de requisitos tem um conjunto diversificado de usuários, que vão desde a alta administração da organização que está pagando pelo sistema até os engenheiros responsáveis pelo desenvolvimento do software. A Figura a seguir, mostra os possíveis usuários do documento e como eles o utilizam.

O Documento de Requisitos de software



A estrutura de um documento de requisitos.

Prefácio	Deve definir os possíveis leitores do documento e descrever seu histórico de versões, incluindo uma justificativa para a criação de uma nova versão e um resumo das mudanças feitas em cada versão.
Introdução	Deve descrever a necessidade para o sistema. Deve descrever brevemente as funções do sistema e explicar como ele vai funcionar com outros sistemas. Também deve descrever como o sistema atende aos objetivos globais de negócio ou estratégicos da organização que encomendou o software.
Glossário	Deve definir os termos técnicos usados no documento. Você não deve fazer suposições sobre a experiência ou o conhecimento do leitor.
Definição de requisitos de usuário	Deve descrever os serviços fornecidos ao usuário. Os requisitos não funcionais de sistema também devem ser descritos nessa seção. Essa descrição pode usar a linguagem natural, diagramas ou outras notações compreensíveis para os clientes. Normas de produto e processos que devem ser seguidos devem ser especificados.
Arquitetura do sistema	Deve apresentar uma visão geral em alto nível da arquitetura do sistema previsto, mostrando a distribuição de funções entre os módulos do sistema. Componentes de arquitetura que são reusados devem ser destacados.