



Engenharia de Software

Introdução A Engenharia de Software

Conceitos Básicos E Aplicações de Software

Conceitos Básicos

Características que diferenciam o software do hardware:

História da Engenharia de Software

Campos de aplicação do Software:

Software Legado:

Natureza Mutante do Software:

Processos de Software

O que é um Processo de Software?

Quais os Objetivos do Processo de Software?

Modelos de Processos de Software

Atividades Básicas do Processo de Software

Especificação de Software

Projeto e Implementação de Software

Validação de Software

Evolução do Software

Mas porque a manutenção é tão importante?

Metodologias Ágeis

O Manifesto Ágil

Extreme Programmin (XP) - Método Ágil

SCRUM Process - Método Ágil

Cerimônias na SCRUM:

Processos do SCRUM:

Metodologia Tradicional X Metodologia Ágil

Requisitos de Software

Definições de Requisitos de Software

Níveis de Requisitos

Um software é baseado na seguinte premissa:

Tipos de Requisitos

Requisito Funcional (RF)

Requisitos Não Funcionais

Requisitos de Usuário

Requisitos de Sistema

Requisitos de Domínio (RD):

Documento de Requisitos de Software

Estrutura de documento de requisitos:

Engenharia de Requisitos

Levantamento e Análise de Requisitos

Dificuldades do Levantamento e Análise de Requisitos

Especificação de Requisitos

Validação de Requisitos

Modelagem de Sistemas

Introdução a UML (Unified Modeling Language)

Ferramentas CASE (Computer-Aided Software Engineering - Engenharia de Software Auxiliada por Computador)

Atributos de Classes

Diagrama de Sistemas

Diagrama de Sequência

Composição do Diagrama de Sequência

Tipos de Mensagens

Diagrama de Máquina de Estados

Componentes do Diagrama de Estados

Diagrama de Atividades

Componentes do Diagrama de Atividades

Gerenciamento de Software

Qualidade de Software

Fatores da qualidade de software:

Elementos de Garantia da Qualidade de Software

Normas e Modelos de Padrões de Qualidade de Software

CMMI

MPSBr

Teste de Software

Teste de Software

Técnicas de Testes de Software

Testes Funcionais

Testes Estruturais
Evolução de Software
Tipos de manutenção de software:
Distribuição do esforço de manutenção
Configuração de Software

Introdução A Engenharia de Software

Conceitos Básicos E Aplicações de Software

Conceitos Básicos

- **Software** é um segmento de instruções que serão analisadas e processadas pelo computador: é ele quem dará o significado a elas, com o objetivo de executar tarefas específicas.
- Os **softwares** comandam o funcionamento de qualquer computador, uma vez que são a parte lógica que fornece explicações para o hardware do computador.
- O **software** é composto não somente pelos programas, mas também pela documentação associada a esses programas.
- **Software** de computador é a tecnologia mais importante no cenário mundial.
- **Software** é incorporado a sistemas de todas as áreas: transportes, medicina, telecomunicações, militar, industrial, entretenimento, máquinas de escritório... etc.
- **A medida que aumenta a importância do software:** é criado tecnologias que tornam mais fácil, mais rápido e mais barato desenvolver e manter programas de computador de alta qualidade.
- O software distribui o produto mais importante: **a informação**.
- Transforma dados, gerencia informações, fornece um portal de redes mundiais de informação, ameaça a privacidade pessoal e também permite cometer crimes.



“Software é tanto um produto quanto um veículo que distribui um produto.”

Características que diferenciam o software do hardware:

- Software é desenvolvido / processo de engenharia;
- Software continua a ser construído sob encomenda.
- Software não se desgasta, não é suscetível aos fatores ambientais. Mas se deteriora devido a modificações.

História da Engenharia de Software

- A principal característica da **engenharia de software** são seus métodos e técnicas que são utilizados para o desenvolvimento de software.
- O termo **engenharia de software** foi mencionado pela primeira vez em uma conferência da OTAN (Organização do Tratado do Atlântico Norte) conduzida na Alemanha em 1968.
- Essa disciplina ou área do conhecimento surgiu em decorrência da preocupação em contornar a crise que estava se abatendo no software e dar a ele um tratamento de engenharia.
- Quando falamos de **engenharia de software**, não se trata apenas do programa em si: mas de toda a documentação associada e dados de configurações necessários para fazer este programa operar corretamente.
- A **engenharia de software** também inclui diretrizes para a aquisição de conhecimento por parte de seus membros e diretrizes para as práticas comportamentais de seus membros.

Campos de aplicação do Software:

- Software de **Sistema**
- Software de **Aplicação**
- Software de **Engenharia / Científico**
- Software **Embarcado**
- Software para **Linha de Produtos**

- Aplicações **Web / Aplicativos Móveis**
- Software de **Inteligência Artificial**
- **Software Legado**

Software Legado:

- São softwares que foram desenvolvidos décadas atrás e tem sido **continuamente modificados para se adequar a mudanças** dos requisitos de negócio e a plataformas computacionais.
- **Softwares antigos:** programadores que participaram da sua elaboração não estão mais na equipe.
- **Documentação desatualizada ou nenhuma documentação:** não auxiliando em nada a equipe de manutenção.

Natureza Mutante do Software:

Evolução de quatro categorias amplas de software:

- **WebApps:** sistemas e aplicações baseados na Web.
- **Aplicativos Móveis:** o termo aplicativo evoluiu para sugerir software projetado especificamente para residir em dispositivos móveis (iOS, Android ou Windows)
- **Computação em Nuvem:** abrange uma infraestrutura ou “ecossistema” que permite a qualquer usuário, em qualquer lugar, utilizar um dispositivo de computação e compartilhar recursos.
- **Software para Linha de Produtos (LPS):** família de Produtos ou conjunto de sistemas com: características comuns e características variáveis

Processos de Software

O que é um Processo de Software?

- É considerado um **conjunto de atividades** necessárias para definir, desenvolver, testar e manter um produto de software de alta qualidade.

- É uma **abordagem adaptável**
- A intenção é sempre entregar software dentro do **prazo** e com **qualidade**.

Quais os Objetivos do Processo de Software?

- Definir **quais** as atividades que serão executadas ao longo do projeto.
- **Quando, como e por quem** tais atividades serão executadas.

Existem vários processos de desenvolvimento de software diferentes, porém todos envolvem **4 atividades fundamentais**:

1. **Especificação de Software**
2. **Projeto e Implementação de Software**
3. **Validação de Software**
4. **Evolução de Software**

Estas atividades são a **base** sobre a qual o **processo de desenvolvimento** deve ser construído.

Modelos de Processos de Software

- É uma **representação abstrata**, simplificada de um processo de software.
- Incluem as **atividades que fazem parte** do processo de software, os artefatos e os papéis envolvidos.
- **Existem diversos** modelos na literatura.

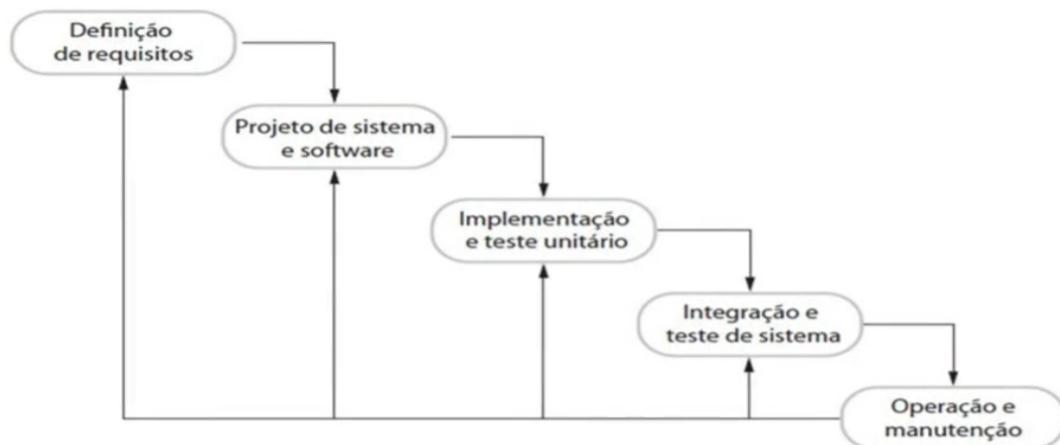
Iremos trabalhar com os três principais modelos:

1. Modelo em Cascata

- a. **Ciclo de vida Clássico**: Paradigma mais antigo da Engenharia de Software;
- b. Requer uma **abordagem sistemática, sequencial** ao desenvolvimento de Software. O resultado de uma fase se constitui na entrada da outra.

Principais Estágios do Modelo em Cascata:

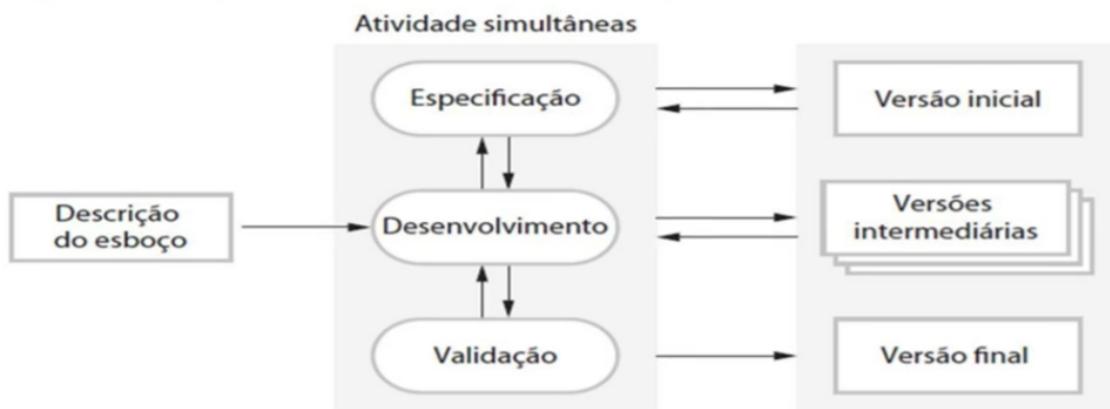
- Análise e definição de requisitos:** as funções, as restrições e os objetivos do sistema são estabelecidos por meio da consulta aos usuários do sistema.
- Projeto de sistemas e de software:** estabelece uma arquitetura do sistema geral. O projeto de software envolve a identificação e a descrição das abstrações fundamentais do sistema de software e suas relações.
- Implementação e testes de unidades:** durante esse estágio, o projeto de software é implementado (codificado) e o teste de unidades é realizado.
- Integração e teste de sistemas:** as unidades de programa ou programas individuais são integrados e testados como um sistema completo, a fim de se garantir que os requisitos de software foram atendidos. Depois dos testes, o sistema de software é entregue ao cliente.
- Operação e manutenção:** essa é a fase mais longa do ciclo de vida. O sistema é instalado e colocado em operação. A manutenção envolve corrigir erros, melhorar a implementação das unidades de sistema e aumentando, consequentemente, as funções desse sistema à medida que novos requisitos são descobertos.



2. Desenvolvimento Incremental

- O objetivo é **trabalhar junto do usuário para descobrir seus requisitos**, de maneira incremental, até que o produto final seja obtido.

- b. Este **modelo** é importante quando é difícil estabelecer a priori uma **especificação** detalhada dos requisitos.



Vantagens em relação ao Modelo em Cascata:

- Se o cliente mudar seus requisitos:** o custo será reduzido, pois a quantidade de análise e documentação a ser refeita é menor do que no modelo em cascata.
- É mais fácil obter um retorno dos clientes** sobre o desenvolvimento que foi feito, pois os clientes vão acompanhando o desenvolvimento do software à medida que novas versões são apresentadas a eles.
- Os clientes podem começar a utilizar o software logo** que as versões iniciais são disponibilizadas, o que não é possível no modelo em cascata.

Desvantagens em relação ao Modelo em Cascata

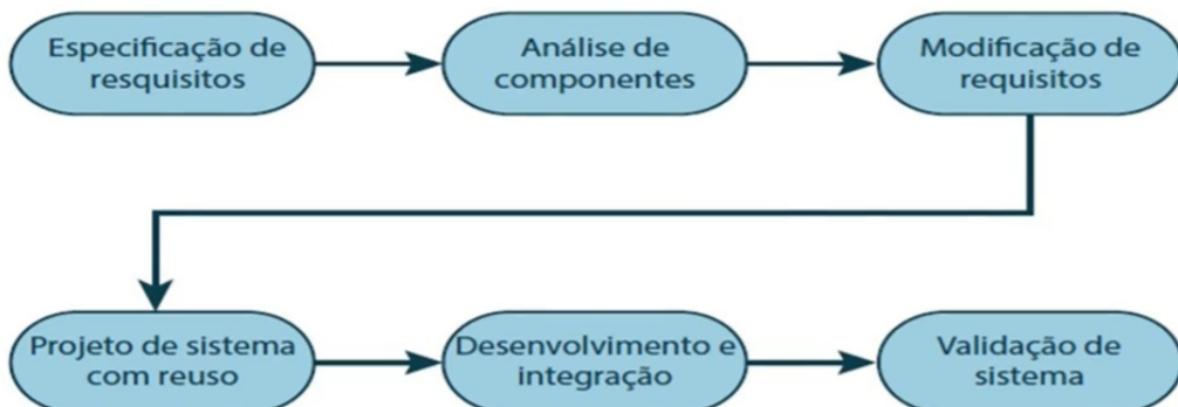
- O processo não é visível:** Dificuldade em medir o progresso e inviabilidade de documentar o sistema por conta do tamanho.
- Os sistemas frequentemente são mal estruturados:** Mudar constantemente pode corromper a estrutura do software e modificações constantes causam custos.
- Podem ser exigidas ferramentas e técnicas especiais:** Você pode utilizar ferramentas de desenvolvimento rápido mas com isto pode se ter poucos profissionais com habilitação para utilizá-las.

3. Engenharia de Software Orientada a Reuso

- a. Foco no **reuso** de software.
- b. Tem a vantagem de propiciar a **entrega mais rápida** do software.

Etapas da Engenharia de Software Orientada a Reuso:

1. **Análise de componentes:** busca por componentes.
2. **Modificação de requisitos:** análise dos componentes encontrados.
3. **Projeto do sistema com reuso:** projeção do framework do sistema sendo levado em consideração os componentes.
4. **Desenvolvimento e Integração:** integração entre o sistema e os sistemas adquiridos e analisados.



Atividades Básicas do Processo de Software

- **Especificação** de Software.
- **Projeto e implementação** de Software.
- **Validação** de Software.
- **Evolução** de Software.

Especificação de Software

1. **Estudo de viabilidade:** considera-se as condições orçamentárias.

2. **Levantamento e análise de requisitos:** conversa, entrevista e prototipação.
3. **Especificação de requisitos:** tradução/filtragem dos requisitos
4. **Validação de requisitos:** verificação da pertinência, consistência e integralidade.



Entender o que o cliente quer está entre as tarefas mais difíceis enfrentadas por um Engenheiro de Software.

“Eu sei que você pensa que entendeu o que eu disse, mas o que você não entende é que aquilo que eu disse não era o que eu quis dizer”.

Projeto e Implementação de Software

- O **projeto de Software** cria uma representação ou modelo do software. Fornecendo detalhes sobre: a arquitetura do software, estruturas de dados, interfaces e componentes do sistema.
- Na **implementação**, o sistema é codificado, ou seja, ocorre a tradução da descrição computacional obtida na fase de projeto em código executável, mediante o uso de uma ou mais linguagens de programação.

Validação de Software

- Destina-se a mostrar que um **software está de acordo com suas especificações** e que atende as expectativas do usuário.
- Envolve **verificar cada processo em cada estágio**, desde a definição dos requisitos dos usuários, até o desenvolvimento de cada um dos programas que compõem o sistema.

Evolução do Software

- O software **evolui de modo a atender as modificações das necessidades** dos usuários
- **Após a implantação** do sistema, é inevitável que ocorram **mudanças**:
 - Pequenos **ajustes pós-implantação**.
 - **Melhorias** substanciais.
 - Força da **legislação**.
 - Atender **novos requisitos** dos usuários.
 - Por estar gerando **erros**.

Mas porque a manutenção é tão importante?

- Pense em como o **mundo muda rapidamente**.
- As **demandas por tecnologias** de informação impõem pressão competitiva nas empresas.
- Mudanças nas **legislações**.

Por essa razão, o software deve ser mantido continuamente, ou seja, passar por manutenções sempre.

Metodologias Ágeis

- Os métodos ágeis são uma abordagem ao **modelo de gestão tradicional** de projetos.
- O modelo de entrega ágil é baseado em **ciclos iterativos e incrementais**, o que traz **flexibilidade e adaptabilidade**.
- Uma característica importante é a **inspeção e adaptação** dos ciclos e iterações, focados em gerar melhoria contínua para as equipes e processos.
- Estão **focados** no que **agrega valor** ao cliente.

O Manifesto Ágil



Em 2001, um grupo de 17 pessoas se reuniram para discutir sobre uma nova abordagem para a gestão de projetos de software.

- Esse manifesto é composto por quatro valores, são eles:
 - Indivíduos e interações mais que processos e ferramentas
 - Software em funcionamento mais que documentação abrangente.
 - Colaboração com o cliente mais que negociação de contratos.
 - Responder a mudanças mais que seguir o plano.
- Principais Metodologias Ágeis:
 - Scrum
 - FDD (Feature Driven Development)
 - Kanban
 - XP (Extreme Programming)
 - Crystal
 - TDD (Teste Driven Development)

Extreme Programming (XP) - Método Ágil

- A Extreme Programming usa uma **abordagem orientada a objetos** enquanto paradigma de desenvolvimento de software.
- A XP é considerada uma metodologia ágil, em que os **projetos são conduzidos com base em requisitos que se modificam rapidamente**.
- A metodologia XP tem, como meta, atender as necessidades dos clientes com **mais qualidade, mais rapidez e de forma simples**.
- **Valores Fundamentais:**
 - **Comunicação** foca em construir um entendimento da pessoa com o problema.
 - **Simplicidade:** faz aquilo que é mais simples hoje e criar um ambiente em que o custo de mudanças no futuro seja baixo.

- **Feedback Rápido:** programadores e clientes se comunicando frequentemente.
- **Coragem:** alterar código já escrito e que está funcionando.
- **Presumir simplicidade:** todo problema deve ser tratado para ser resolvido da forma mais simples possível
- **Mudanças Incrementais:** mudanças devem ser incrementais e feitas aos poucos.
- **Abraçar Mudanças:** mudanças devem ser sempre bem-vindas.
- **Trabalho de Qualidade:** ter um sistema que atenda aos requisitos do cliente, que rode 100% dos casos de teste e que agregue o maior valor possível para o negócio do cliente.

SCRUM Process - Método Ágil

- As metodologias consideradas ágeis, assim como Scrum, são **fortemente influenciadas pelas práticas da indústria japonesa** (adotadas pelas empresas **Toyota e Honda**)
- A origem do termo Scrum surgiu de um artigo escrito em **1986** por **Takeuchi e Nonaka**, intitulado **The New Product Development Game** (o novo jogo de desenvolvimento de produtos).
- A **metodologia** Scrum foi desenvolvida em **1990** por **Jeff Sutherland e Ken Schwabe**.
- Scrum é um **framework** para desenvolver e manter produtos complexos.
- É uma metodologia que **concentra as suas atenção no produto final**, com um rápido desenvolvimento, e nas interações dos indivíduos.
- Segue os princípios do **Manifesto Ágil**: são usados para orientar as atividades de desenvolvimento dentro de um processo que incorpora as seguintes atividades metodológicas: **requisitos, análise, projeto, evolução e entrega**.

Características Importantes do Processo SCRUM

- Flexibilidade de resultados e prazos.
- Trabalho com equipes pequenas

- Uso de revisões frequentes
- Colaboração dos interessados
- Se baseia na orientação a objetos

É conhecido pelo fato de:

- Estruturar seu funcionamento por **ciclos**, chamados de **sprints**, que representam iterações de trabalho com duração variável.
- As **tarefas** desenvolvidas dentro de um sprint são **adaptadas ao problema** pela equipe scrum e são realizadas em um **período curto de tempo** durante o desenvolvimento do projeto.
- Scrum **estipula um conjunto de práticas e regras** que devem ser seguidas pela equipe (papéis, cerimônias e artefatos).

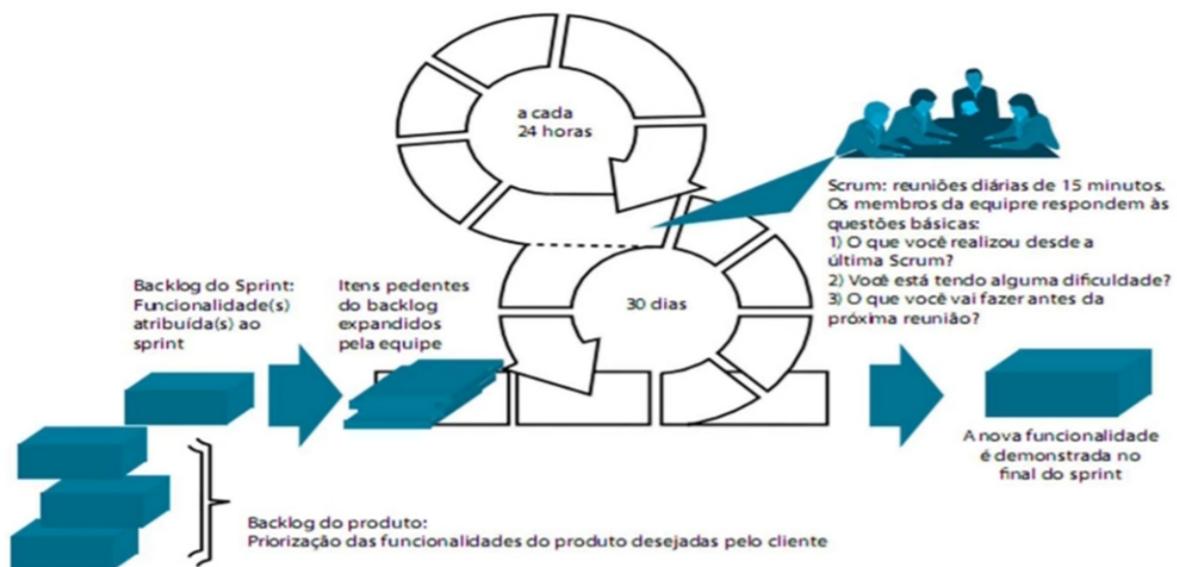
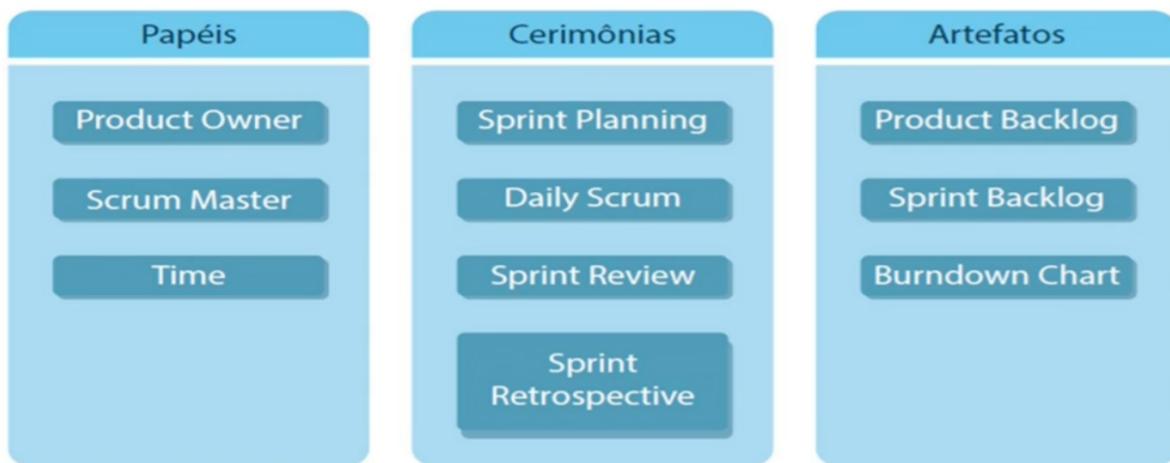
Cerimônias na SCRUM:

1. **Sprint Planning:** Reunião inicial com o *Product Owner*, onde se elabora uma lista de prioridades.
2. **Daily Meeting:** Reunião diária mais curta com o time para compartilhar o que cada um realizou, os próximos passos e se há algum impedimento.
3. **Sprint Review:** Balanço sobre tudo que foi feito durante uma sprint. Nela, o time deve mostrar apenas os resultados concluídos da sprint para o *Product Owner*.
4. **Sprint Retrospective:** Tudo o que pode ser melhorado na sprint, se atenta ao que deu errado e o que pode ser ajustado na sprint, a fim de otimizá-la.

Processos do SCRUM:

1. **Product Backlog:** Lista de prioridades dos requisitos ou de funcionalidades para o projeto. Estimam-se os custos e riscos, também definem-se as ferramentas de desenvolvimento, a equipe e as datas de entrega para os resultados.
2. **Sprint Backlog:** Lista de funcionalidades que serão realizadas no próximo sprint, onde também são definidas as funções de cada membro da equipe.

3. Burndown Chart



Metodologia Tradicional X Metodologia Ágil

▼ Qual é a melhor?



Depende do contexto do software a ser desenvolvido.

- Mesmo considerando as vantagens das metodologias ágeis no desenvolvimento de software, sempre devemos pensar “no que” e “para que” será desenvolvido.
- Para os softwares que são críticos: precisão, risco de confiabilidade são decisivos, não podemos dispensar as recomendações que são impostas pelas regras e normas das metodologias tradicionais, principalmente em relação a documentação.

ASPECTO	METODOLOGIA TRADICIONAL	METODOLOGIA ÁGIL
Objetivo	Orientado por atividades e foco nos processos.	Orientado por produto e foco nas pessoas.
Projeto	Estável e inflexível a mudança.	Adaptável a mudanças.
Tamanho	Qualquer tamanho, efeito em projetos de longa duração.	Pequeno, mas pode ser usado em projetos de maior porte.
Gerente de Projeto	Controle total do projeto.	Papel de facilitador ou coordenador.
Equipe do Projeto	Atuação com papéis claros e bem definidos em todas as atividades.	Atuação colaborativa em todas as atividades.
Cliente	Participa nas fases iniciais do levantamento de requisitos	Parte integrante da equipe do projeto.
Planejamento	Detalhado e os envolvidos não participam.	Curto e com a participação de todos os envolvidos.

Requisitos de Software

Definições de Requisitos de Software



Entender os requisitos de um problema está entre as tarefas mais difíceis enfrentadas por um engenheiro de software.

- **Engenharia de Requisitos**
 - Descobrir
 - Analisar
 - Documentar
 - Verificar
- A Engenharia de Requisitos constrói uma ponte entre o projeto e a construção.
- O processo de estabelecer as **funções** que um cliente requer de um sistema e as **restrições** sob as quais ele deve funcionar e ser desenvolvido.
- Mesmo se os clientes e usuários finais fossem explícitos quanto às suas necessidades, essas mudariam ao longo do projeto.
- Os requisitos são descrições das **funções** e **restrições** que são geradas durante o processo de engenharia de requisitos.
- Um requisito compreende uma **característica ou funcionalidade** que o sistema deve possuir ou uma **restrição** que deve satisfazer para atender uma necessidade do usuário.
- É a parte mais crítica e propensa a erros no desenvolvimento de software.

Níveis de Requisitos

- **Requisitos do Usuário:** se destinam às pessoas envolvidas no uso e na aquisição do sistema. Devem ser escritos usando linguagem natural, tabelas e diagramas de modo que sejam comprehensíveis.
- **Requisitos do Sistema:** se destinam a comunicar, de modo preciso as funções que o sistema tem de fornecer. Podem ser escritos: em linguagem estruturada ou linguagem com base em alguma linguagem de programação.
- **Requisitos de Projeto (Especificação de projeto):** é a definição do projeto de software em nível mais técnico - modelagem.
- Por que é difícil entender os requisitos? Porque temos diferentes níveis de descrição.

Requisitos de Usuário

Requisitos de Sistema

O sistema **deve** gerar relatórios mensais que mostram o custo dos medicamentos prescritos por clínica durante cada mês.

1. No último dia de cada mês deve ser gerado um resumo dos medicamentos prescritos por clínica.
2. Um **relatório** por clínica deve ser gerado, listando nome dos medicamentos, total de prescrições e o custo total.

- Quando os requisitos não são declarados de forma precisa, podem surgir vários problemas.
- Requisitos ambíguos podem ser interpretados de diferentes maneiras pelos desenvolvedores e usuários.

*Exemplo: considere o termo **telas do sistema apropriadas**:*

- **Usuário:** telas especiais diferentes para cada tipo de documento.
- **Desenvolvedor:** fornecer uma tela texto que mostre o conteúdo do documento.

Um software é baseado na seguinte premissa:



- O que **entra**, como dados, informações, eventos é um requisito.
- O **processamento** em si a ser realizado tem regras, fórmulas, normas, critérios, arquivos, tabelas, etc. para que seja executado.
- A **saída** desse processamento também é composta de dados e informações ou disparos de eventos.

Tipos de Requisitos



- **Requisitos funcionais:** dizem respeito à definição das funções que um sistema ou um componente de sistema deve fazer (entradas e saídas).
- **Requisitos não funcionais:** dizem respeito às restrições, aspectos de desempenho, interfaces com o usuário, confiabilidade, segurança, manutenibilidade, portabilidade e Padrões.
- **Requisitos de domínio/negócio:** requisitos derivados do domínio da aplicação e descrevem características do sistema e qualidades que refletem o domínio (regra do negócio).



Requisito é uma **exigência, solicitação, desejo, necessidade**.

Requisito Funcional (RF)

- Quando falamos de Requisitos Funcionais, estamos nos referindo à requisição de uma funcionalidade que um software deverá atender/realizar. Ou seja, exigência, solicitação, desejo, necessidade, que um software deverá materializar.
- Dizem respeito à definição das funções que um sistema ou componente de sistema deve fazer.
- Exemplos:
 - [RF001] O Sistema deve cadastrar médicos profissionais (entrada).
 - [RF002] O Sistema deve emitir um relatório de clientes (saída).
 - [RF003] O Sistema deve passar um cliente da situação “em consulta” para “consultado” quando o cliente terminar de ser atendido (mudança de

estado).

- [RF004] O cliente pode consultar seus dados no sistema.

Requisitos Não Funcionais

- Dizem respeito às restrições, aspectos de desempenho, interfaces com o usuário, confiabilidade, segurança, manutenibilidade, portabilidade e Padrões.
- Os requisitos não funcionais, sempre que possível, devem ser escritos quantitativamente para que possam ser objetivamente testados.
- Exemplos:
 - [RNF001] O sistema deve imprimir o relatório em até 5 segundos
 - [RNF002] Todos os relatórios devem seguir o padrão de relatórios especificado pelo setor XYZ.
 - [RNF003] O sistema deve ser implementado em Java.
 - [RNF004] O sistema deve ser protegido para o acesso de usuário.

Requisitos de Usuário

- Devem descrever os RFs e RNFs.
- Especificam o comportamento externo do sistema, onde os usuários não possuem conhecimento técnico detalhado.

Requisitos de Sistema

- São as descrições mais detalhadas dos Requisitos de Usuários
- Servem como base para um contrato destinado à implementação do sistema e, portanto, devem ser uma especificação completa e consistente de todo o sistema.

Requisitos de Domínio (RD):

- [RD001] O cálculo da média final de cada aluno é dado pela fórmula: $(\text{Nota1} * 2 + \text{Nota 2} * 3)/5$.
- [RD002] O valor do IPI é calculado em relação ao valor da nota fiscal da mercadoria despachada, que pode eventualmente incluir valores sobre o frete e despesas acessórias (juros, taxas e outras).
- [RD003] O cálculo de comissão dos vendedores é de 15% sobre o total líquido das vendas no mês.

Documento de Requisitos de Software

- Documento de acordo (entre quem solicita e quem desenvolve).
- Estabelece o escopo do software (conjunto de funcionalidades que ele deverá oferecer).
- Garante uma rastreabilidade mínima.
- Deve servir de referência para o desenvolvimento, testes, manutenção e evolução do sistema (mudanças).
- O **RUP (Rational Unified Process)** sugere modelos para documento de requisitos mais complexo.
- O guia **PMBOK** tem uma sugestão para a documentação de requisitos.



O desafio é decidir qual o modelo que melhor se adapte ao seu projeto, a sua empresa, a sua equipe.

Qual o nível certo de documentação?

1. O documento deve ser capaz de garantir o entendimento dos stakeholders.
2. A equipe técnica deve estar ciente de que esse documento é a única garantia que atende às solicitações do cliente.

Qual modelo ou padrão usar?

- Temos vários modelos e modelos são adaptáveis.

- Definir junto com equipe qual o padrão que utilizarão para a documentação dos requisitos.

Estrutura de documento de requisitos:

Capítulo	Descrição
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 usuários	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 a serem seguidos devem ser especificados.
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.

- ***Exemplo: Locadora de Filmes***

- A locadora possui muitos títulos em seu acervo e não consegue controlar de maneira eficiente as locações, devoluções e reservas dos filmes.
- Ela deseja ter um sistema informatizado que controle todas as locações, devoluções e reservas de maneira eficiente, para aperfeiçoar o seu atendimento com o cliente e seu controle interno.

- A locadora possui: uma ficha para o cadastro de clientes com os seguintes dados: nome do cliente, fone residencial, fone celular, sexo, RG, CPF, endereço completo, data de nascimento, estado civil e nomes de cinco dependentes e o grau de parentesco de cada dependente (o dependente pode locar filmes em nome do cliente)

Necessidades do sistema:

1. **Manter o cadastro de filmes:** nome do filme, duração, sinopse, classificação, gênero, diretor, elenco. Para cada cópia do filme é necessário saber o fornecedor da mesma, a data da compra, o valor pago e o tipo (VHS ou DVD).

2. **Controlar locações:**

- A locação feita mediante a verificação de cadastro do cliente. Se o cliente for cadastrado então se efetua a locação, se não, é feito o cadastro do cliente.
- Caso a locação seja efetuada pelo dependente do cliente, é necessário deixar registrado qual o dependente e qual o cliente.
- Verificações: Se o filme está disponível e se o cliente possui pendências financeiras ou atraso de devolução, caso uma das alternativas seja afirmativa bloqueia-se a operação, sendo liberada somente após a devida regularização.
- Emitir comprovante de locação com a data prevista para devolução de cada filme, discriminação dos filmes e se o pagamento foi ou não efetuado.
- A data prevista para devolução deve ser calculada desconsiderando domingos e feriados. Cada categoria pode ter um prazo diferente para que o cliente possa ficar com o filme.
 - i. Ex: a categoria LANÇAMENTO permite que o cliente fique com o filme por 2 dias.

3. **Controlar devoluções**

- Verificar se a devolução está no prazo correto e se o pagamento foi efetuado, caso o prazo esteja vencido calcular a multa incidente. Efetuado o pagamento, emite-se o recibo de devolução.
- Não esquecer que não pode ser cobrada multa caso seja domingo ou feriado.

4. Controlar reservas

- a. Verificar se o cliente já está cadastrado, caso contrário o sistema permite o cadastro do cliente no momento da reserva. Também é verificado se o filme desejado está disponível para reserva.
- b. Reservar somente para clientes sem pendências financeiras e devoluções vencidas.

5. Consultar Filmes Locados Por Cliente

- a. O sistema deve ter uma consulta em que seja informado um determinado cliente e sejam mostrados todos os filmes já locados por esse cliente e mostre também a data em que cada filme foi locado.

6. Consultar reservas por filme:

- a. O sistema deve ter uma consulta em que seja informado um determinado filme e sejam mostradas todas as reservas efetuadas para aquele filme, no período informado.

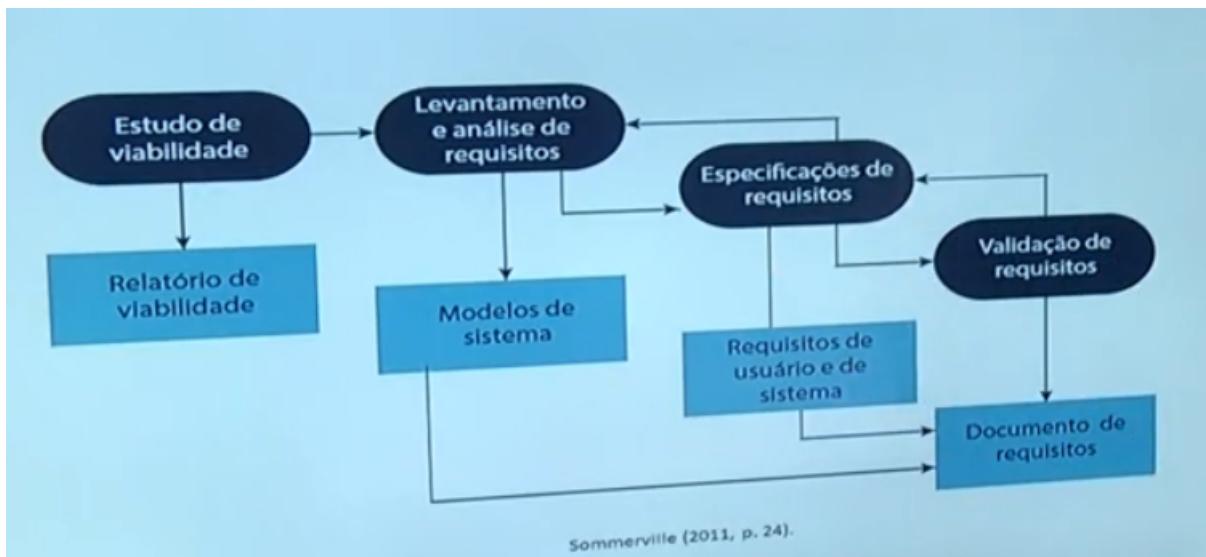
7. Emitir os seguintes relatórios:

- a. **Relatório Geral de Clientes:** em que conste o código, nome, endereço, telefone e dependentes do cliente.
- b. **Etiquetas com códigos de barras** para a identificação das cópias no processo de locação e devolução.
- c. **Relatório de filmes por gênero** em que conste o código do filme, o nome do filme, o nome do diretor do filme, os nomes dos atores do filme, o total de cópias, o total de cópias locadas e o total de cópias disponíveis. O relatório deve ser agrupado por gênero, mostrando também o código e a descrição do gênero.
- d. **Relatório de filmes locados por cliente por período.** Para cada cliente devem ser emitidas todas as cópias que estão locadas para ele. Deve sair no relatório: o código e o nome do cliente, o código do filme, o nome do filme, o código da cópia (exemplar), a data de locação e o valor da locação. O relatório deve ser agrupado por cliente e devem sair somente as cópias locadas e não devolvidas.
- e. **Relatório de cópias não devolvidas**, em que conste o código do filme, o nome do filme, o código da fita, o nome do cliente, o telefone do cliente, a data de locação, a data prevista para devolução e o número de dias em atraso.

- f. **Relatório dos filmes mais locados**, em que conste o código do filme, o nome do filme, a descrição do gênero e o número total de locações. O relatório deve ser agrupado por mês/ano, ou seja, para um determinado mês/ano, devem ser emitidos os 10 (dez) filmes mais locados.
- g. **Relatório de Reservas por período**, em que conste o código do cliente, o nome do cliente, o telefone do cliente, o código do filme reservado, o nome do filme, a data em que foi feita a reserva (data em que o cliente telefonou para a locadora dizendo que queria fazer a reserva).
- h. **Relatório de valores das locações mensais**, deverá mostrar os valores das locações de determinado mês, separado por data e somatória de valores de cada dia, somando-se assim ao final, uma totalidade de locações. Nele deve-se conter a data e a soma das locações nesta data.

Engenharia de Requisitos

- Processo que envolve todas as atividades necessárias para a criação e manutenção de um documento de requisitos de software.
- Existem **quatro** atividades genéricas de processo de engenharia de requisitos que são de alto nível:
 1. O estudo de viabilidade do sistema.
 2. O levantamento e análise de requisitos.
 3. A especificação de requisitos e sua documentação finalmente.
 4. A validação desses requisitos.



- O estudo de viabilidade é um estudo rápido, direcionado, que se destina a responder a algumas perguntas:
 1. O sistema contribui para os objetivos gerais da organização?
 2. O sistema pode ser implementado com a utilização de tecnologia atual dentro das restrições de custo e de prazo?
 3. O sistema pode ser integrado com outros sistemas já em operação?

Levantamento e Análise de Requisitos

- Nessa atividade os membros da equipe técnica de desenvolvimento de software trabalham com o cliente e os usuários finais do sistema para descobrir mais informações sobre o domínio da aplicação, que serviços o sistema deve fornecer, o desempenho exigido no sistema, as restrições de hardware etc.
- O levantamento e a análise de requisitos podem envolver diferentes tipos de pessoas em uma organização.
- O termo **stakeholder** é utilizado para se referir a qualquer pessoa que terá alguma influência direta ou indireta sobre os requisitos do sistema.

Dificuldades do Levantamento e Análise de Requisitos

1. Os **stakeholders** frequentemente não sabem na realidade o que querem do sistema computacional.
2. Os **stakeholders** em um sistema expressam naturalmente os requisitos em seus próprios termos e com o conhecimento implícito de sua área de atuação.
3. Diferentes **stakeholders** têm em mente diferentes requisitos e podem expressá-los de maneiras distintas.
4. Fatores políticos podem influenciar os requisitos do sistema.
5. O ambiente econômico e de negócios, no qual a análise de requisitos ocorre, pode mudar durante o processo de análise

Especificação de Requisitos

- Durante o levantamento de requisitos (levantamento de dados), a equipe de desenvolvimento tenta entender o domínio (contexto/problema).
- Utiliza-se o **DOCUMENTO DE REQUISITOS** ou **ESPECIFICAÇÃO DE REQUISITOS** que engloba requisitos funcionais, requisitos não funcionais, de usuário e do sistema.

Validação de Requisitos

- **Objetivo:** mostrar que os requisitos realmente definem o sistema que o cliente deseja.
- A validação deve se ocupar da elaboração de um esboço completo do documento de requisitos.
- É importante porque a ocorrência de erros em um documento de requisitos pode levar a grandes custos relacionados ao retrabalho.

Verificações de Validação:

1. **Verificações de validade:** Trata-se da verificação das diversas funções utilizadas podendo neste estágio verificar a necessidade de funções adicionais.
2. **Verificações de consistência:** Os requisitos em um documento não devem ser conflitantes, não devendo existir descrições diferentes para uma mesma função do sistema.

3. **Verificação de completeza:** Verificação se o documento de requisitos estejam definindo todas as funções e restrições exigidas pelos usuários do sistema.
4. **Verificações de realismo:** Verificados, a fim de assegurar que eles realmente podem ser implementados, levando-se também em conta o orçamento e os prazos para o desenvolvimento do sistema.
5. **Facilidade de verificação:** Para diminuir as possíveis divergências entre cliente e fornecedor, os requisitos do sistema devem sempre ser escritos de modo que possam ser verificados. Isso implica na definição de um conjunto de verificações para mostrar que o sistema entregue cumpre com esses requisitos.

Técnicas de Validação

1. **Revisões de Requisitos:** Os requisitos são analisados sistematicamente por uma equipe de revisores, a fim de eliminar erros e inconsistências.
2. **Prototipação:** Cria um executável do sistema e é mostrado aos usuários finais e clientes.
3. **Geração de casos de teste:** Permite a criação de ambientes de teste para que percorra o fluxo real de informações.

Modelagem de Sistemas

Introdução a UML (Unified Modeling Language)

“UML é uma linguagem padrão para elaboração da estrutura de projetos de software. Ela poderá ser empregada para a visualização, a especificação, a construção e a documentação de artefatos que façam uso de sistemas complexos de software” (BOOCH, 2005, p.13).

- A UML é uma **linguagem de modelagem**.
- **Meta:** auxiliar os engenheiros de software a definirem as características do software, tais como seus requisitos, seu comportamento, sua estrutura lógica, a

dinâmica de seus processos e até mesmo suas necessidades físicas em relação ao equipamento em que o sistema deverá ser implantado.

- A UML é independente do processo de software, visto que pode ser utilizada em modelo cascata, desenvolvimento evolucionário ou em qualquer outro processo que esteja sendo utilizado para o desenvolvimento de software.
- A notação UML utiliza diversos símbolos gráficos, existindo uma semântica bem definida para cada um deles, o que permite elaborar diversos modelos.
- UML é a linguagem-padrão de modelagem mais popular do momento
- A UML poderá ser utilizada para:
 - Visualização
 - Especificação
 - Construção de modelos e diagramas
 - Documentação

Ferramentas CASE (Computer-Aided Software Engineering - Engenharia de Software Auxiliada por Computador)

- É um software que apoia as atividades do processo de software, como a engenharia de requisitos, o projeto, o desenvolvimento de programas e os testes.
- As ferramentas CASE podem incluir editores de texto, dicionários de dados, compiladores, depuradores, ferramentas de construção de sistemas e entre outros.
- Alguns exemplos de atividades que podem ser automatizadas utilizando a CASE:
 - O desenvolvimento de modelos gráficos de sistemas enquanto parte das especificações de requisitos ou do projeto de software
 - A compreensão de um projeto utilizando um dicionário de dados que carrega informações sobre as entidades e sua relação em um projeto.
 - A geração de interfaces com usuários, a partir de uma descrição gráfica da interface, que é criada interativamente pelo usuário.

- A depuração de programas pelo fornecimento de informações sobre um programa em execução.
 - A tradução automatizada de programas, a partir de uma antiga versão de uma linguagem de programação, como Cobol, para uma versão mais recente
- Upper CASE ou U-CASE ou Front-End
 - Voltada para as primeiras fases do processo de desenvolvimento de sistemas, como análise de requisitos, projeto lógico e documentação
- Lower CASE ou L-CASE ou Back-End
 - Suporte nas últimas fases do desenvolvimento, como a codificação, os testes e a manutenção.
- Integrated CASE ou I-CASE
 - Englobam Upper e Lower CASE's, ainda oferecem outras características, como por exemplo, controle de versão.
- Best in Class ou Kit de Ferramenta:
 - Semelhante as I-CASE, os Kits de Ferramenta também possuem a propriedade de conjugar sua capacidade a outras ferramentas externas complementares.

Exemplos de Ferramentas CASE:

- IBM - Rational Rose
- Astah
 - Free Student Academic License
 - <http://astah.net/student-license-request>
- ArgoUML
- Microsoft Virtual Modeler
- Visual Paradigm

Modelagem de Sistemas

- Processo de desenvolvimento de modelos abstratos de um sistema, em que cada modelo apresenta uma visão ou perspectiva, diferente do sistema.

- Os modelos são usados durante o processo de engenharia de requisitos para ajudar a extrair os requisitos do sistema durante o processo de projeto.
- **Modelagem de sistemas** é a atividade de construção de modelos que expliquem/ilustrem a forma de funcionamento de um software.

Com a modelagem alcançamos alguns objetivos:

- Os modelos ajudam a visualizar o sistema como ele é ou como desejamos que seja
- Os modelos permitem especificar a estrutura ou o comportamento de um sistema
- Os modelos proporcionam um guia para a construção do sistema
- Os modelos documentam o sistema
- Modelagem visual significa modelar com a utilização de notações padrão

Diagramas de Casos de Uso



É um dos diagramas da UML mais abstrato, flexível e informal

- **Objetivo:** modelar as funcionalidades e serviços oferecidos pelo sistema
- **Composto por:** atores, casos de uso e relacionamentos.
- **Atores:** qualquer elemento externo que interaja com o sistema. (imagem abaixo)



Cliente



Caixa Eletrônico



Sistemas de Contas a Pagar e a Receber



Funcionário



Gerente

Casos de Uso

- Referem-se aos serviços, tarefas ou funções que podem ser utilizadas de alguma maneira pelos usuários do sistema.
- Representados na forma de elipses com texto interno descrevendo a que serviço/tarefa/função o caso de uso se refere.
- Nome = Verbo + Substantivo (indicador de ação) [imagem abaixo]

Abrir Conta

Comprar Produtos

Efetuar Login

- **Exemplo:**

- **Caso de uso:** Comprar produtos
 - **Atores:** Cliente, atendente
 - **Descrição:** Um cliente chega ao ponto de vendas para comprar produtos. O atendente registra a compra e coleta o pagamento. Cliente vai embora com a compra.



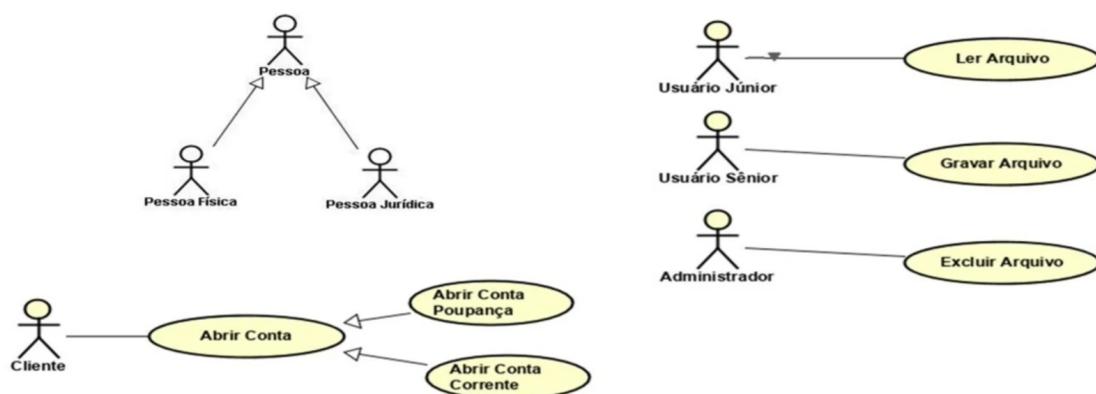
Associações

- As associações entre atores e casos de uso podem conter setas indicando a naveabilidade desta. Sentido em que as informações trafegam.
- Tipos de Associações: comunicação (associação), generalização ou especialização, inclusão e extensão.

Comunicação: Associação entre um ator e um caso de uso.

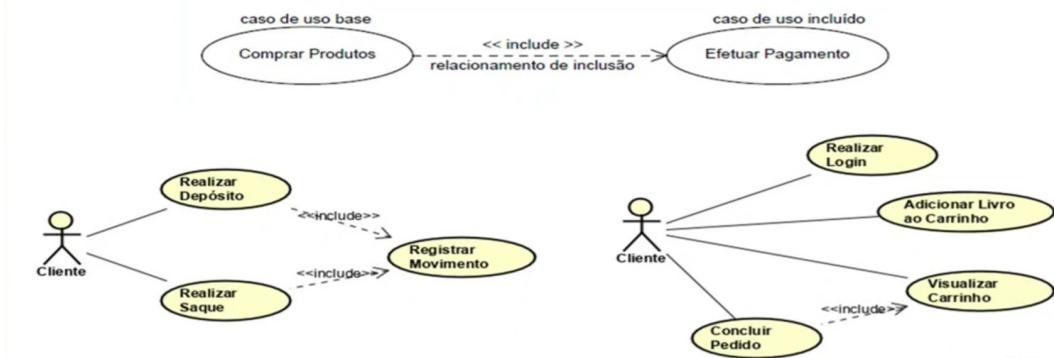


Generalização/Especialização: relaciona casos de uso com características semelhantes e pequenas diferenças entre si.

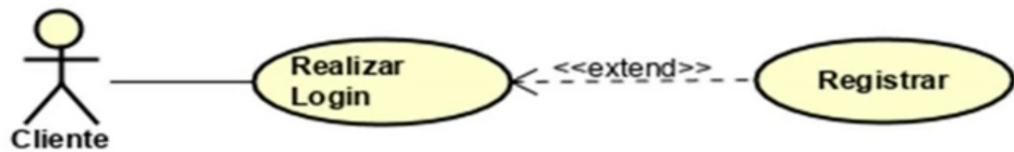


Inclusão: <<include>> relacionamento com outro caso de uso que sempre será executado.

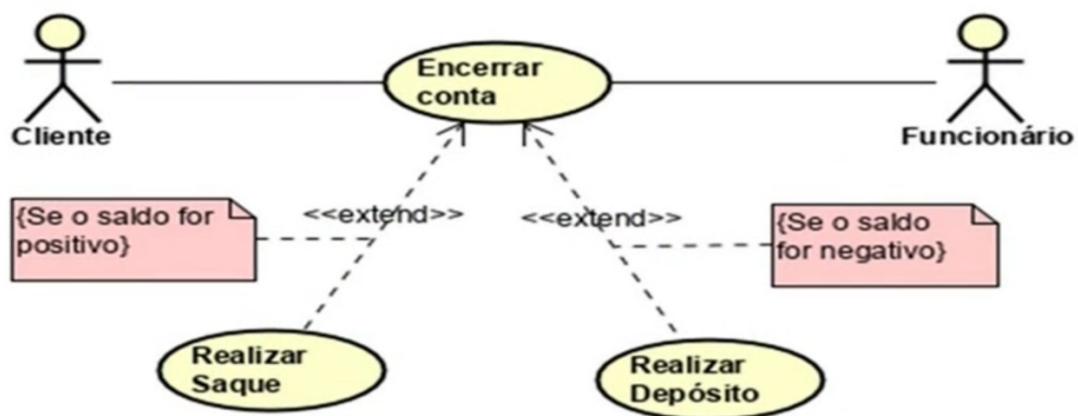
Exemplo: quando um caso de uso A possui relacionamento de inclusão com outro caso de uso B, a execução de A implica na execução de B.



Extensão: <<extend>> relacionamento com outro caso de uso que pode ou não ser executado.



Restrições em Associações de Extensão:



Conceitos Básicos de Orientação a Objetos

- A UML é totalmente baseada no paradigma de Orientação a Objetos (OO)
- Para compreendê-la corretamente, precisamos estudar alguns dos conceitos relacionados a orientação a objetos.
 - Entre eles: objetos, classes, atributos, métodos, visibilidade, herança e polimorfismo

Objeto

- Qualquer coisa concreta ou abstrata que existe no mundo real, em que se pode individualizá-la por possuir comportamentos e características próprias
- São exemplos de objetos:
 - Cliente
 - Professor
 - Carteira
 - Caneta
 - Carro
 - Disciplina
 - Curso
 - Caixa de diálogo.
- Os objetos possuem características e comportamentos.



Objetos Concretos

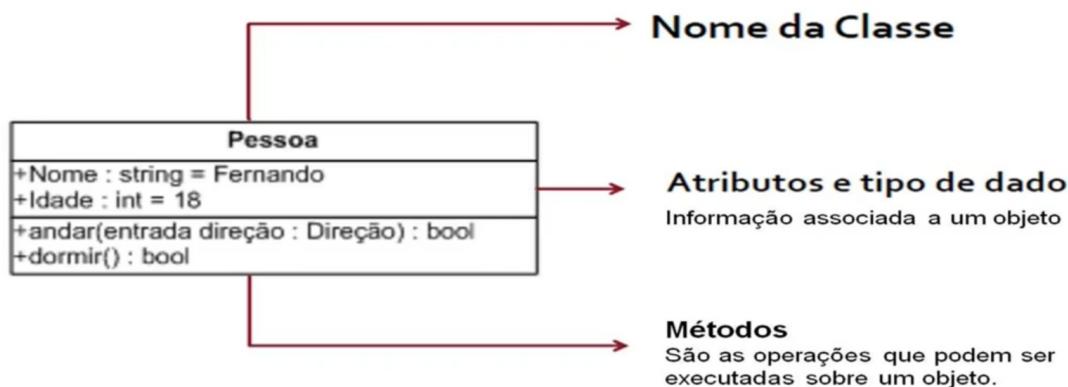


Objetos Abstratos

Abstração: Abstraímos quando definimos um objeto conceitual partindo de **OBJETOS** do mundo real, com os mesmos comportamentos e características, os quais são classificados como um de mesmo tipo.

Classe

- Representa a **ABSTRAÇÃO** de um conjunto de **OBJETOS** do mundo real que possui comportamentos e características comuns.
- Descrição de uma coleção de objetos que possuem propriedades semelhantes (atributos, métodos, associações)

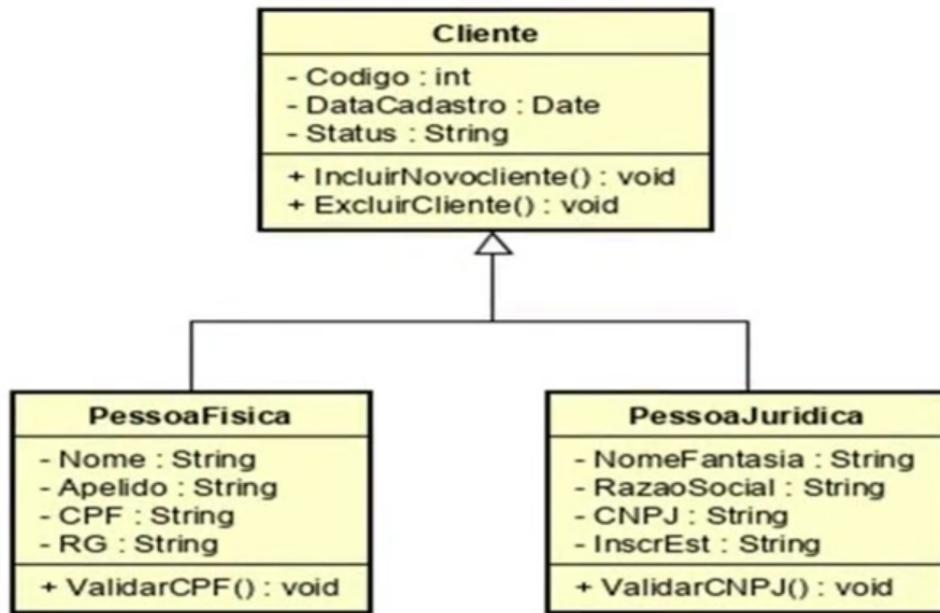


Visibilidade:

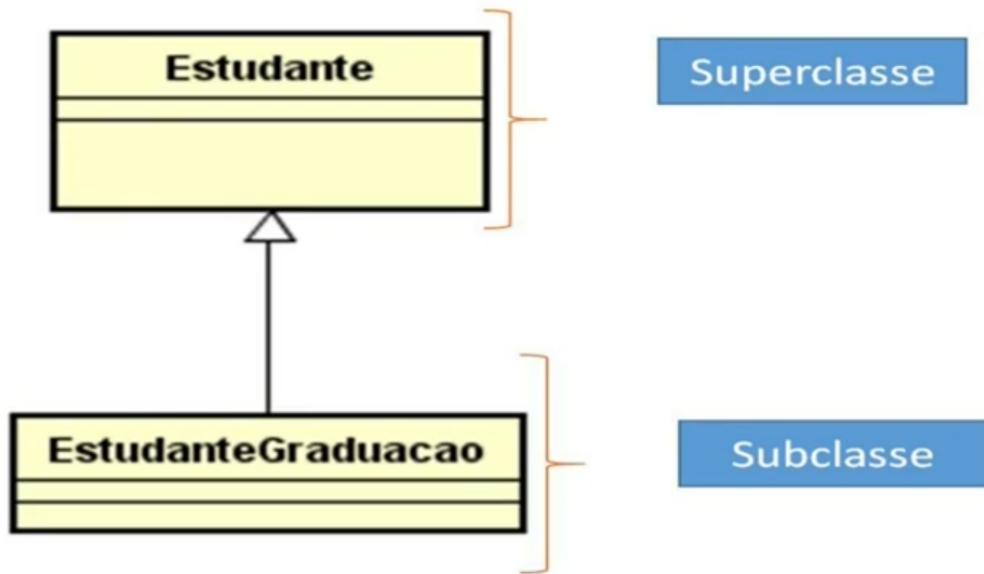
- O símbolo de (+) indica visibilidade pública, ou seja, significa que o atributo ou método pode ser utilizado por objetos de qualquer classe.
- O símbolo de (#) indica que a visibilidade é protegida, ou seja, determina que apenas objetos da classe possuidora do atributo ou método ou de suas subclasses podem acessá-lo.
- O símbolo de (-) indica que a visibilidade é privada, ou seja, somente os objetos da classe possuidora do atributo ou método poderão utilizá-lo.

Herança:

- É a propriedade que possibilita que a classe herde características e comportamento de outra classe.



- A herança acontece entre classes gerais (chamadas de superclasses ou classes-mãe) e classes específicas (chamadas subclasses ou classes-filha).



Polimorfismo:

- É a capacidade que Objetos de Classes diferentes possuem de se comportarem de forma diferente em uma mesma operação

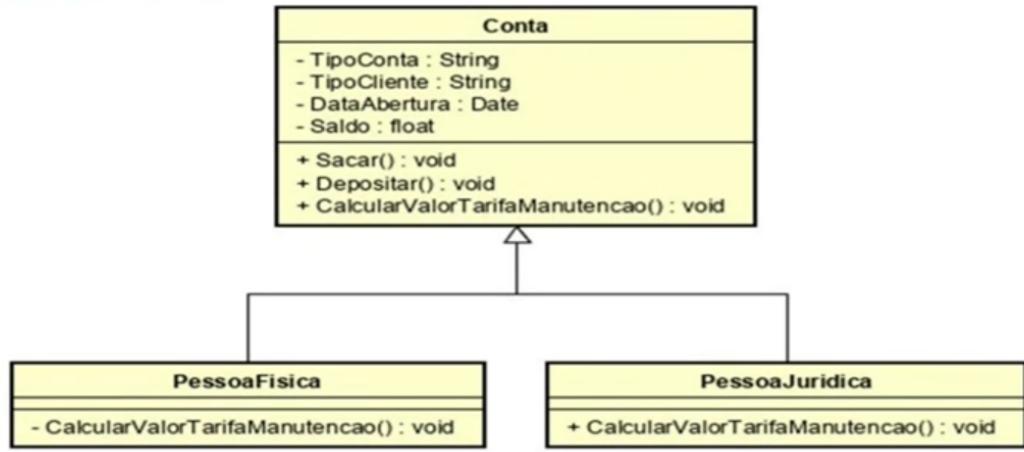


Diagrama de Classes

- Mostra um conjunto de classes e seus relacionamentos
- É o diagrama central da modelagem orientada a objetos
- Graficamente, as classes são representadas por retângulos incluindo nome, atributos e métodos.

Notação que permite descrever:

- Classes
- Atributos e métodos
- Relacionamentos entre as classes
- Detalhes de implementação (código)

Relacionamentos

Os relacionamentos possuem:

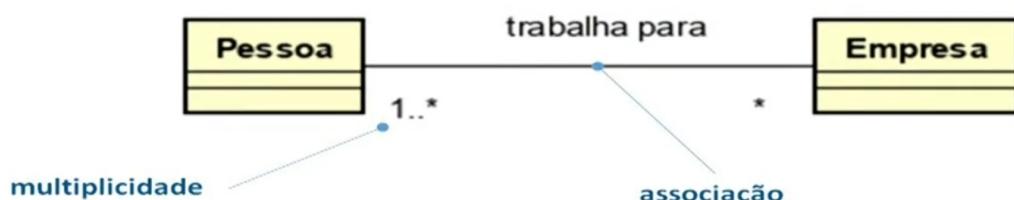
- Nome: descrição dada ao relacionamento (faz, tem, possui,...)
- Sentido de leitura
- Multiplicidade: 0..1, 0..*, 1, 1..*, 2, 3..7 ...
- Tipo: associação (agregação, composição) generalização e dependência.
- Uma associação é um relacionamento estrutural que indica que os objetos de uma classe estão vinculados a objetos de outra classe.

- Uma associação é representada por uma linha sólida conectando duas classes:



Multiplicidade

- Especifica o nível de dependência de um objeto e o número máximo e mínimo de instâncias envolvidas.
- Existem situações em que é necessário restringir o número de objetos associados através de uma associação a um objeto determinado (restringir a cardinalidade).



MULTIPLICIDADE	SIGNIFICADO
0..1	No <u>mínimo zero (nenhum)</u> e no <u>máximo um</u> . Indica que os objetos das classes associadas não <u>precisam</u> obrigatoriamente estar relacionadas, mas se houver relacionamento indica que apenas uma instância da classe se relaciona com as instâncias da outra classe.
1..1	Um e somente um. Indica que apenas um objeto da classe se relaciona com os objetos da outra classe.
0..*	No <u>mínimo nenhum</u> e no <u>máximo muitos</u> . Indica que pode ou não haver instância da classe participando do relacionamento.
*	<u>Muitos</u> . Indica que muitos objetos da classe estão envolvidos no relacionamento
1..*	No <u>mínimo 1</u> e no <u>máximo muitos</u> . Indica que há pelo menos um objeto envolvido no relacionamento, podendo haver muitos objetos envolvidos.
2..5	No <u>mínimo 2</u> e no <u>máximo 5</u> . Indica que existe pelo menos 2 instâncias envolvidas no relacionamento e que pode ser 3, 4 ou 5 as instâncias envolvidas, mas não mais do que isso.



Um professor pode orientar quantos alunos?



Um professor orienta vários alunos (multiplicidade)



Agregação

- Informa que uma classe faz parte de outra classe, mas não de forma exclusiva. A classe “todo-parte” recebe um diamante vazio e a outra

ponte (classe “parte”) uma linha. Agregação é chamada de “é-partde-de”

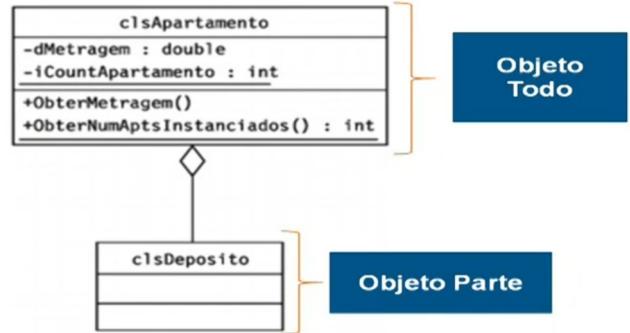


Para saber se é agregação:

- Pergunta ao objeto A se cabe como todo-parte
- Depois pergunta ao objeto B se vive sem objeto A
- Se a resposta for sim em ambas as perguntas, temos uma agregação

Exemplo de Agregação:

A existência do Objeto-Parte faz sentido, mesmo não existindo o Objeto-Todo.



Composição

- Informa que uma classe faz parte de outra classe de forma exclusiva.
 - Os objetos parte só podem pertencer a um único objeto “todo” e têm o seu tempo de vida coincidente com o dele quando o “todo” morre todas as suas “partes” também morrem.

Para saber se é composição:

- Pergunta ao objeto A se cabe como todo-parte.
- Depois pergunta ao objeto B se vive sem objeto A
- Se a resposta for sim para a primeira pergunta e não para segunda, temos uma composição.

Exemplo de Composição:

- Uma venda é composta de vários detalhes (itens):
 - Quando a venda é criada, os itens de venda também são
 - Quando a venda é eliminada, os itens também são eliminados.

Agregação x Composição: diferença

- **Agregação:** se excluir a classe responsável pelo relacionamento, não deve excluir a classe que ele possui relacionamento. Agregações são assimétricas: se um objeto A é parte de um objeto B, B não pode ser parte de A.
- **Composição:** se excluir a classe responsável pelo relacionamento, então deve excluir a classe que ele possui relacionamento.

Atributos de Classes

Classes em Negrito

- **Paciente:**
 - Código
 - Nome
 - Endereço
 - CEP
 - Cidade
 - UF
 - Telefone
 - Data de Nascimento
 - RG
 - CPF
- **Exame**

- Código
 - Descrição
 - Valor
 - Procedimentos
 - Grupo ao qual pertence o Exame
- **Pedido de Exame**
 - Código
 - Nome do Paciente
 - Nome do Médico
 - Nome do convênio
 - Nomes dos exames que serão realizados.
 - Data e Hora da realização de cada exame
 - Data e hora em que cada exame ficará pronto
 - Valor de cada exame
 - **Resultado de Exame**
 - Descrição do resultado
 - **Médico**
 - CRM
 - Nome
 - **Convênio**
 - Código
 - Nome
 - **Cidade**
 - Código
 - Nome
 - DDD
 - **UF**
 - Sigla

- Nome
- **Grupo de Exame**
 - Código
 - Descrição

Diagrama de Sistemas

Diagrama de Sequência

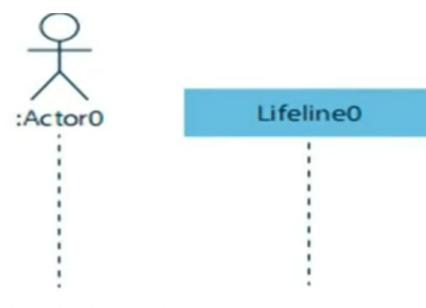
- Segundo Guedes, o “diagrama de sequência procura determinar a **sequência de eventos** que ocorrem em um determinado **processo**, identificando quais **mensagens** devem ser **disparadas** entre os elementos envolvidos e em ordem”.
- Pode ser baseado no diagrama de caso de uso e no diagrama de Classes:
 - **Diagrama de caso de uso:** cada caso especificado refere-se a um processo disparado por um ator.
 - **Diagrama de classe:** as classes contidas nele se tornam objetos no diagrama de sequência.

Composição do Diagrama de Sequência

- **Atores:** são usuários ou pessoas do meio externo que participam, tendo algum papel dentro do sistema.
 - Ator pode ser aproveitado com base no diagrama de casos de uso, gerando, assim, eventos iniciais dos processos.
 - A representação de um ator se dá por meio de bonecos idênticos aos do diagrama de casos de uso, mas contendo uma linha de vida logo abaixo.
- **Linha de vida (lifelines):** é uma linha pontilhada que representa o tempo que um objeto existe durante um processo. A linha pode estar ligada a objetos, atores e entre outros.
 - A linha de vida é composta por uma instância de uma classe que participa de uma interação.

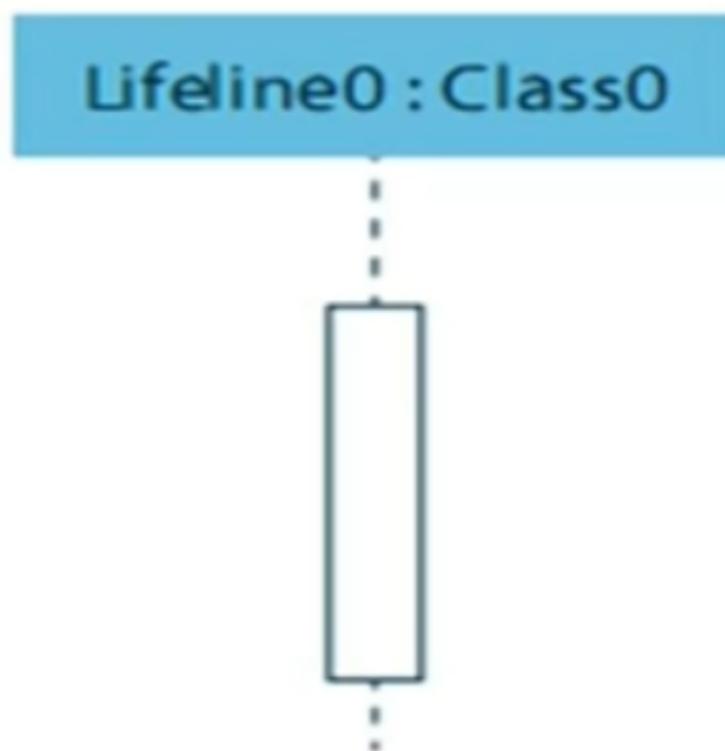


Exemplo de *lifeline* interrompida



Exemplo de *lifeline* com ator e classe

- **Foco de controle ou ativação:** para representar os momentos ativos em que um objeto está executando um ou mais métodos, é utilizado o foco de controle.
 - É representado por uma linha mais grossa sobre a linha pontilhada.



Exemplo de foco de controle

- **Mensagens:** tem o objetivo de mostrar a ocorrência de eventos.
 - Forçam a chamada de um método entre os objetos envolvidos no processo

MENSAGEM	EXEMPLO
Um Ator e outro Ator	<pre> sequenceDiagram participant A as Representante participant B as Representante A->>B: 1: Solicitar o cadastro de um novo cliente() </pre> <p>Figura 16 – Mensagens simples entre atores</p>

MENSAGEM	EXEMPLO
Um Ator e um Objeto	<pre> sequenceDiagram participant A as Atendente participant B as Pessoa A->>B: 1: ConsultaCliente(ConeCpf(string)) </pre> <p>Figura 17 – Mensagens simples entre ator e objeto</p>
Um Objeto e outro Objeto	<pre> sequenceDiagram participant A as Nota_Fiscal participant B as Estoque A->>B: 1: Apresentação da NF, ocorre baixa no Estoque() </pre> <p>Figura 18 – Mensagens simples entre ator e objeto</p>
Um Objeto e um Ator	<pre> sequenceDiagram participant A as Nota_Fiscal participant B as Cliente A->>B: 1: Envio da NF para o Cliente após emissão() </pre> <p>Figura 19 – Mensagens simples entre ator e objeto</p>

Tipos de Mensagens

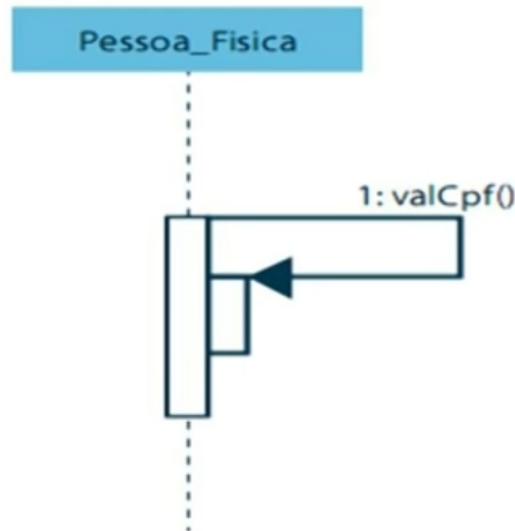
- **Mensagens síncronas:** no momento em que um objeto (chamador) realizar o envio de uma mensagem síncrona, o objeto deverá aguardar que ela seja finalizada para que seja continuado o fluxo.
 - Representação: uma flecha com o desenho da ponta preenchido.



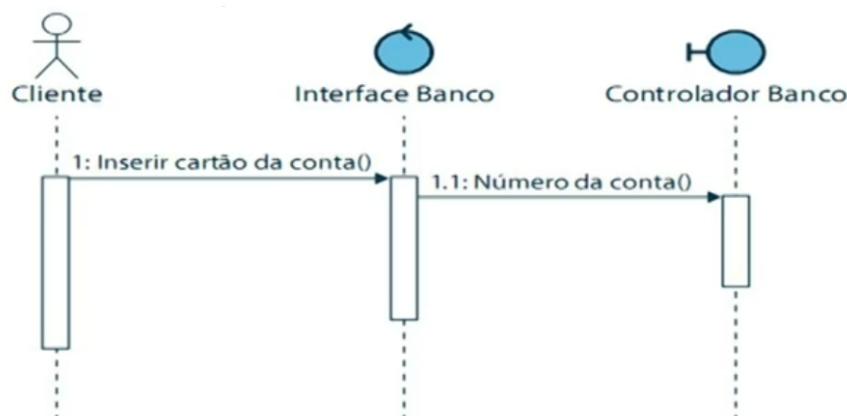
- **Mensagens assíncronas:** são utilizadas para indicar que a execução ocorre em paralelo a outros processos e que ela pode ter um processamento contínuo, o que retira a necessidade de aguardar por uma resposta para que o processo continue.
 - Representação: é a ponta de seta tipo “pé de galinha”



- **Autochamadas ou autodelegações:** são mensagens que o objeto envia para ele mesmo.
 - Representação: a seta parte da linha da vida e retorna ao próprio objeto.



- **Estereótipo <<boundary>>**: também conhecido como estereótipo de fronteira, o estereótipo <<boundary>> é utilizado para identificação de uma classe que comunica os atores externos e o próprio sistema.
 - O uso de classes <<boundary>> é feito quando se quer definir uma interface para o sistema.
- **Estereótipo <<control>>**: trata-se de uma classe intermediária entre as classes <<boundary>> e as demais.
 - Objetivo: realizar uma interpretação de eventos realizados pelo objeto <<boundary>>, como as ações do mouse e a execução de botões, por exemplo, o que permite retransmitir ações aos demais objetos do sistema.



- **Exemplo de Diagrama de Sequência:** Processo de emissão de saldo

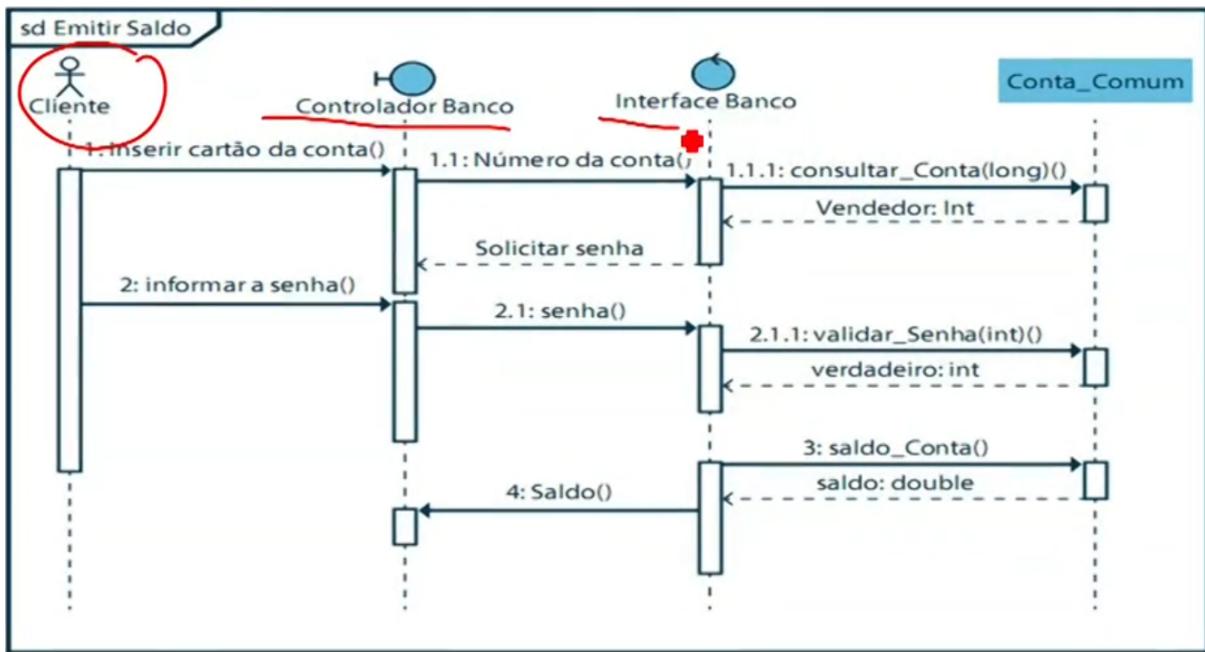


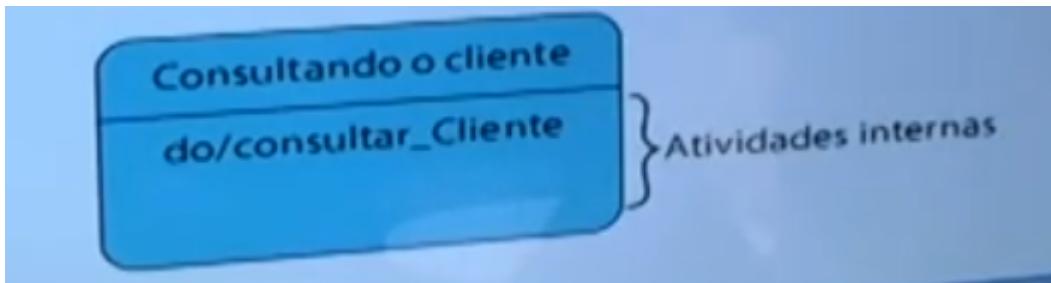
Diagrama de Máquina de Estados

- Segundo Guedes, “o diagrama de máquina de estados demonstra o comportamento de um elemento por meio de um conjunto finito de transições de estado, ou seja, uma máquina de estados”.
- O diagrama de máquina de estados é um diagrama de **comportamentos**.
- Usado para especificar o comportamento de vários elementos, seja uma instância de uma classe ou um diagrama de caso de uso.

Componentes do Diagrama de Estados

- Estado:** representa, dentro do diagrama, os momentos em que um componente (objeto) está ou pode vir a estar.
 - Dentro de um processo, podemos ter um ou vários estados ocorrendo de forma simultânea.
 - Representação: utilizamos um retângulo.
- Atividades internas:** dentro das atividades internas, pode haver as seguintes variações:
 - Entry: determina que a atividade descrita será executada no momento em que o objeto entra em um estado.

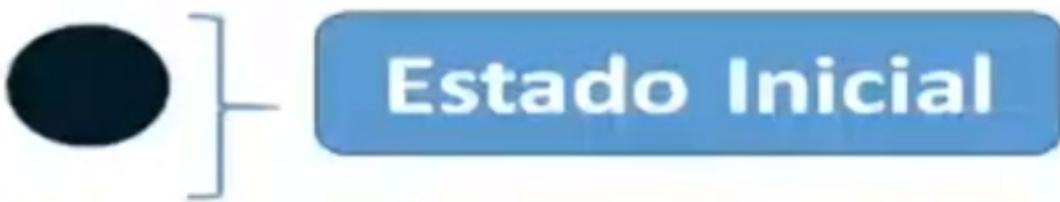
- Exit: determina que a atividade descrita será executada no momento em que o objeto sai de um estado.
- Do: determina que a atividade descrita será executada no período em que o estado for executado.



- **Transições:** evidenciam uma alteração no estado entre um objeto e outro, a fim de permitir a geração de um novo estado.
 - Uma transição se dá por meio de uma flecha, que pode, ou não, conter uma descrição a respeito.



- **Estado Inicial:** simboliza o processo que está começando a ser modelado. Constituído por um círculo preenchido.



- **Estado final:** simboliza que a modelagem do processo chegou ao fim. Constituído de um círculo preto preenchido o qual está envolto de um círculo não preenchido.



Exemplo de Diagrama de Estados:

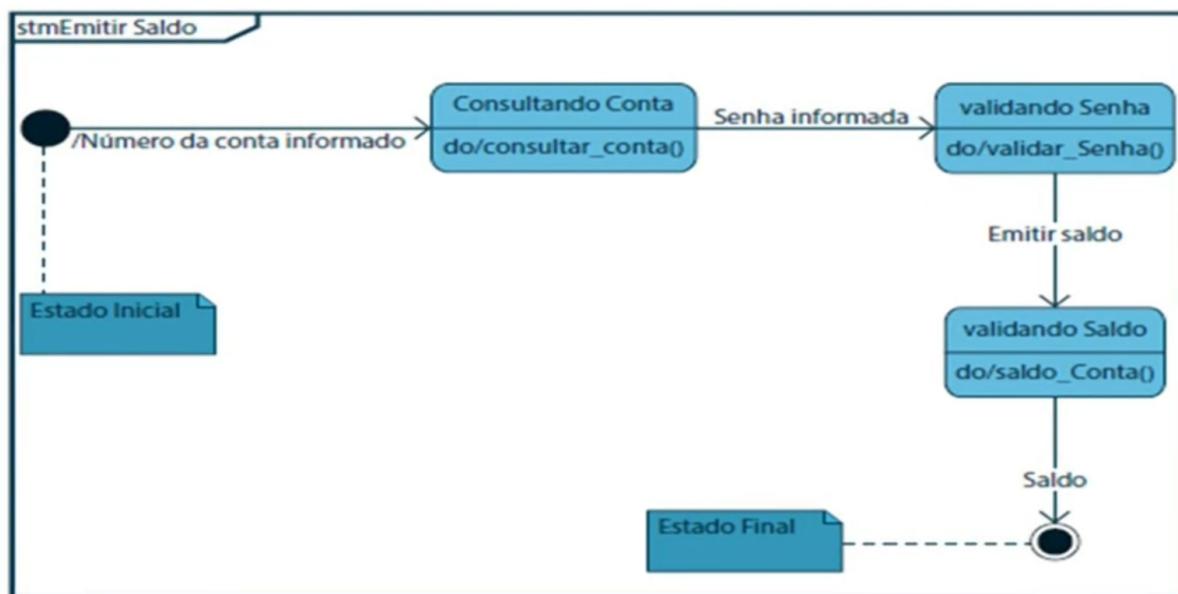
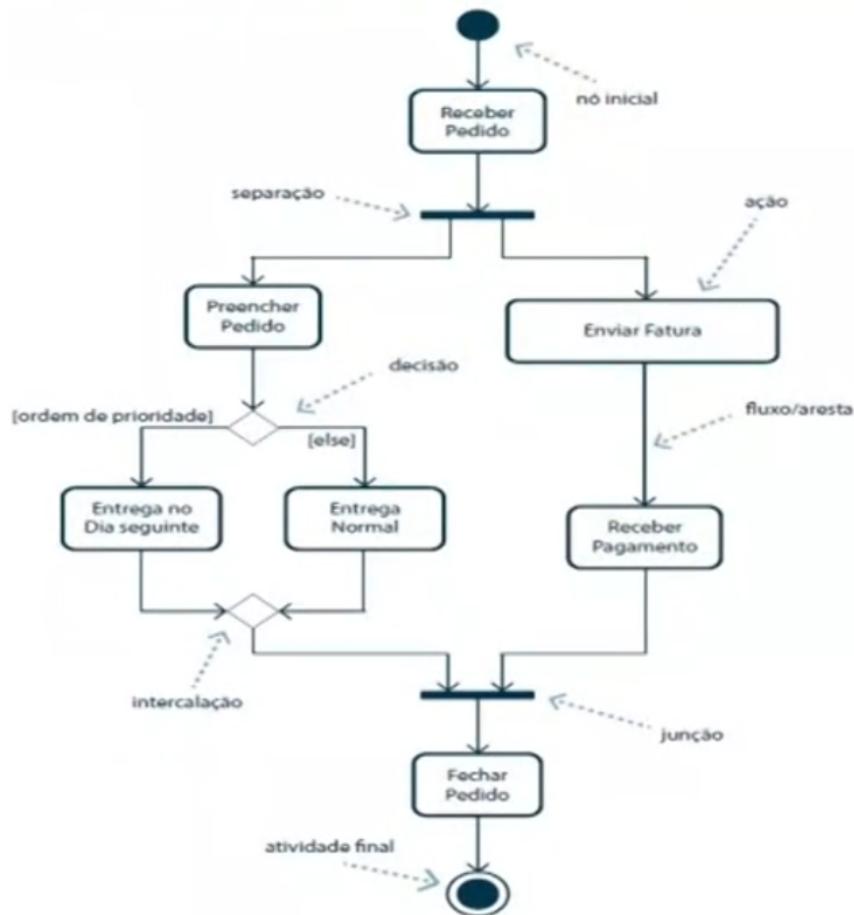


Diagrama de Atividades

- Segundo Guedes, o “diagrama de atividades ajuda na modelagem de atividades que podem ser um método, ou um algoritmo, ou mesmo um processo completo”.
- Está ligado, também, a descrição de computação procedura, a modelagem organizacional para engenharia de processos e ao workflow.

Componentes do Diagrama de Atividades

- **Atividade:** representada por um retângulo com bordas arredondadas. Pode receber vários tipos de comportamentos, ocorrências de funções aritméticas, comportamento de atividades, ações de comunicação, a leitura e gravação de atributos ou até mesmo a instanciação.
- **Nó de Ação:** um nó de ação representa um passo, uma etapa que deve ser executada em uma atividade. Um nó de ação é atômico, não podendo ser decomposto.
- **Fluxo de Controle:** representado por uma seta que realiza a ligação entre dois nós, local onde passam sinais de controle do nó antigo apontando para o novo.
 - No fluxo de controle, pode-se, também, descrever a condição ou restrição.
- **Nó Inicial:** representado por um círculo preenchido, o nó inicial visa demonstrar o início do fluxo da atividade invocada. Temos o nó inicial acoplado a um fluxo de um sistema.
- **Nó Final:** tipo nó de controle e representado por um círculo preenchido por outro, o nó de final de atividade é utilizado para representar que o fluxo de uma atividade acabou.
- **Nó de Decisão:** grande maioria dos fluxos de um sistema oferece possibilidades de percurso ou condições.
 - Representação: símbolo de losango para simbolizar as possíveis condições que aquele ponto do sistema nos oferece.
- **Partição de Atividade**



Gerenciamento de Software

Qualidade de Software

- Para que o software possa ser funcional e prático, é preciso ter qualidade em sua produção e manutenção.
- Qualidade, em software, refere-se as características dos produtos que atendem os requisitos de funcionalidade.
- Tais requisitos estão ligados diretamente com a área de utilização do programa e, para cada um.
- Sendo eles: funcionalidade, confiabilidade, usabilidade, eficiência, manutenibilidade, portabilidade, estabilidade.

O termo qualidade possui várias definições, as quais variam de acordo com a abordagem utilizada. A seguir, algumas definições utilizadas na literatura:

- **Conformidade com as especificações:** o gerenciamento da qualidade deve ser feito desde o início do desenvolvimento, para tentar evitar defeitos e diminuir o retrabalho.
- **Adequação ao uso:** significa que as expectativas do cliente são atendidas. Dois fatores a considerar: a qualidade obrigatória e a qualidade atrativa.
- **NBR ISO 8402:** qualidade é a totalidade das características de uma entidade que lhe confere a capacidade de satisfazer às necessidades explícitas e implícitas.
- Os fatores de qualidade representados concentram-se em três importantes aspectos: **revisão, transição e operação** do produto de software.

Fatores da qualidade de software:

Fator	Descrição
CORREÇÃO	O quanto um programa satisfaz a sua especificação e atende aos objetivos da missão do cliente.
CONFIABILIDADE	O quanto se pode esperar que um programa realize a função pretendida com a precisão exigida.
EFICIÊNCIA	A quantidade de recursos computacionais e código exigidos por um programa para desempenhar sua função.
INTEGRIDADE	O quanto o acesso ao software ou dados por pessoas não autorizadas pode ser controlado.
USABILIDADE	Esforço necessário para aprender, operar, preparar a entrada de dados e interpretar a saída de um programa.
FACILIDADE DE MANUTENÇÃO	Esforço necessário para localizar e corrigir um erro em um programa.

Elementos de Garantia da Qualidade de Software

Elemento	Descrição
PADRÕES	O IEEE, a ISO e outras organizações de padronização produziram uma ampla gama de padrões para engenharia de software e seus respectivos documentos.
REVISÕES E AUDITORIAS	As revisões técnicas são uma atividade de controle de qualidade realizada por engenheiros de software. Seu intuito é o de revelar erros.
TESTES	Os testes de software são uma função de controle de qualidade com um objetivo principal: o de descobrir erros.
COLETA E ANÁLISE DE ERROS/DEFEITOS	A única forma de melhorar é medir o nosso desempenho.
ADMINISTRAÇÃO DA SEGURANÇA	Garantir que o software não tenha sido alterado internamente, sem autorização.
PROTEÇÃO	Devemos avaliar o impacto de falhas de software e por iniciar as etapas necessárias para redução de riscos.

Normas e Modelos de Padrões de Qualidade de Software

- As normas e modelos de padrões de qualidade de software. São a principal chave para a garantia da qualidade. São elas quem definem as características que todos os componentes de software devem possuir e como o processo de software deve ser conduzido, de forma a assegurar a qualidade do produto de software.
- Na literatura, encontramos várias normas e modelos. Entretanto, na disciplina, vamos estudar o **CMMI** e **MPSBr**.

CMMI

- O **CMMI** (Modelo Integrado de Maturidade e Capacitação): foi desenvolvido pelo **SEI** (Software Engineering Institute) e é um modelo desenvolvido para a melhoria da maturidade dos processos de desenvolvimento de software.
 - Trata-se de um modelo criado para **avaliar e melhorar a capacitação** das empresas que desenvolvem software, **propondo etapas** que levam a empresa a se aprimorar continuamente em busca de soluções para o seu crescimento com qualidade

Objetivos principais do CMMI envolvem:

- Auxiliar as empresas a se conhecerem e com isso espera-se que elas melhorem seus processos de desenvolvimento e a manutenção do software.

- Fornecer, às empresas, um controle de seus processos por meio de uma estrutura conceitual e, com isso, obter a melhoria contínua de seus produtos de software.

O modelo não diz como implementar determinadas práticas, ele apenas indica o que deve ser feito.

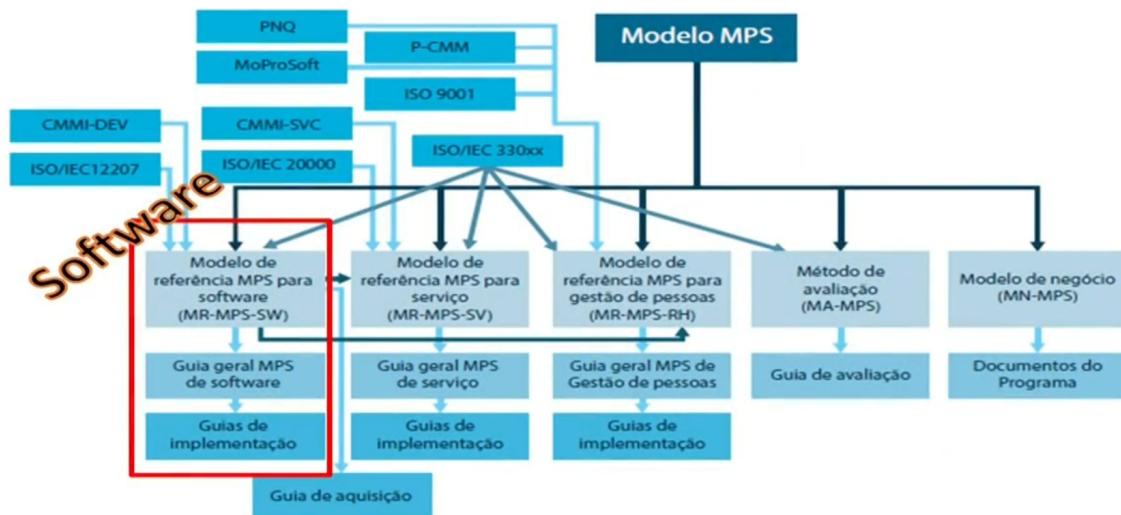
- **Representação Contínua:** níveis de capacidade. O modelo descreve um caminho de melhoria de maturidade por meio de cinco níveis distintos. Sendo eles:

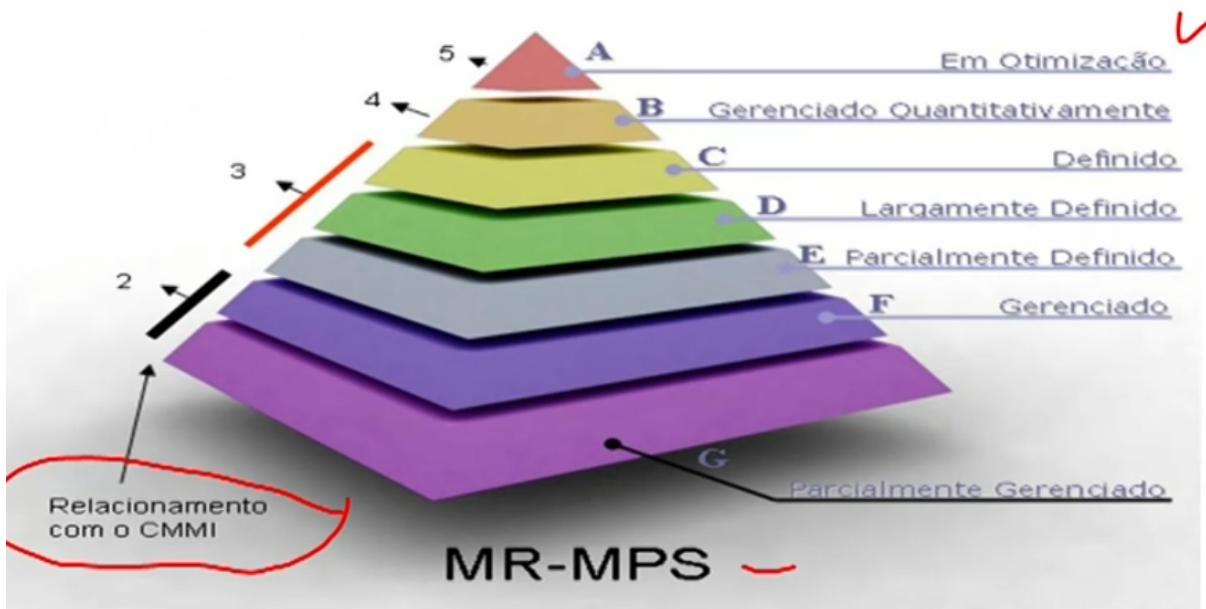


- **Representação em Estágio:** foca nas melhores práticas que uma empresa pode utilizar. A maturidade é medida por um conjunto de processos

NÍVEL DE MATURIDADE	ÁREAS DE PROCESSO
Nível 2 – Gerenciado	Gestão de requisitos Planejamento de projeto Monitorização e controle de projeto Gestão de acordo com o fornecedor Mediação e análise Garantia da qualidade de processo e do produto Gestão de configurações

- É um modelo de qualidade de processos de software voltado para as características das empresas **brasileiras**.
- Ele auxilia as empresas a implantarem seus processos de software em conformidade com os principais padrões e modelos internacionais de qualidade de software.
- Ele é baseado no padrão CMMI, nas normas ISO/IEC 12207 e SPICE, e principalmente na realidade do mercado brasileiro.
- **Vantagens:**
 - Uma das principais vantagens do modelo é seu custo reduzido de certificação, sendo ideal para micro, pequenas e médias empresas.
- **Dividido em três componentes:**
 - **Modelo de Referência (MR-MPS):** para serviço e gestão de pessoas.
 - **Método de Avaliação (MA-MPS).**
 - **Modelo de Negócio (MN-MPS).**





Níveis de Maturidade:

- **Nível G - Parcialmente Gerenciado** - primeiro nível do modelo, é composto pelos processos de gerência de projeto e gerência de requisitos;
- **Nível F - Gerenciado** - é composto pelo nível de maturidade G acrescido dos processos de gerência de configuração, garantia da qualidade, medição e aquisição;
- **Nível E - Parcialmente Definido** - é composto pelo nível de maturidade F, acrescido dos processos de treinamento, definição do processo organizacional, avaliação e melhoria do processo organizacional e adaptação do processo para gerência de projetos;
- **Nível D - Largamente Definido** - é composto pelo nível de maturidade E, acrescido dos processos de desenvolvimento de requisitos, solução técnica, validação, verificação, integração de produto, instalação de produto e liberação de produto;
- **Nível C - Definido** - é composto pelo nível de maturidade D, acrescido dos processos de gerência de riscos e análise de decisão e resolução;
- **Nível B - Gerenciado Quantitativamente** - é composto pelo nível de maturidade C, acrescido dos processos de desempenho do processo organizacional e gerência quantitativa de projeto;

- **Nível A - Em Otimização** - é o nível mais elevado do modelo MPSBr; é composto pelo nível de maturidade B, acrescido dos processos de inovação e implantação na organização e análise e resolução de causas.

Cada nível de maturidade possui **áreas do processo**, nas quais são analisados os:

- **Processos fundamentais:** aquisição, gerência de requisitos, desenvolvimento de requisitos, solução técnica, integração, instalação e liberação do produto.
- **Processos organizacionais:** gerência de projeto, adaptação do processo para gerência de projeto, análise de decisão e resolução, gerência de riscos, avaliação e melhoria do processo organizacional, definição do processo
- **Processos de apoio:** garantia de qualidade, gerência de configuração, validação, medição, verificação e treinamento.

Níveis de Capacidade:

- **AP1.1** - O processo é executado.
- **AP1.2** - O processo é gerenciado.
- **AP2.2** - Os produtos de trabalho do processo são gerenciados.
- **AP3.1** - O processo é definido
- **AP3.2** - O processo está implementado.

Teste de Software

Teste de Software

- **Teste de software** é o processo que visa executar o sistema de forma controlada, com o objetivo de **avaliar o seu comportamento**, baseado no que foi **especificado**.
- É destinado a **mostrar** que um programa **faz o que é proposto a fazer** e para **descobrir defeitos** do programa.
- Por que testar?
 - Quando um código defeituoso é executado, falhas ocorrem.

- Um sistema com falhas gera insatisfação, ferimentos ou até mesmo a morte dos clientes e usuários.
- Testamos para:
 - Verificar se o sistema está fazendo o que foi solicitado que ele fizesse no requisito.
 - Garantir que o negócio não vai correr riscos provocados por defeitos em produção.
 - Assegurar a Qualidade do sistema.
- Será que o teste prova que tudo está bem e funcionando adequadamente?
 - Os desenvolvedores querem provar que “algo funciona”.
 - Os testadores querem provar que “algo não funciona”.

Visão do desenvolvedor:

- Objetivo provar que as coisas estão funcionando
- Testa os cenários positivos.

Visão do Testador:

- Objetivo de provar a não adequação de algo.
- Testa os cenários positivos e negativos e de stress.

Quando ocorrem os erros?

- Um defeito pode ocorrer em função de desvios do que foi levantado na análise de requisitos.
- Defeitos decorrentes da falta de concordância com a especificação do produto.
- Defeitos decorrentes de situações inesperadas, mas não definidas nas especificações do produto.

Verificação: respeito ao conjunto de tarefas que visa garantir que o software teve suas funções específicas implementadas corretamente.

- “Estamos construindo corretamente o sistema?

Validação: constitui como um conjunto de tarefas que objetiva assegurar que o software foi criado e pode ser verificado com base nos requisitos do cliente.

- “Estamos construindo o sistema correto?”

Técnicas de Testes de Software

- As técnicas de teste são procedimentos técnicos e gerenciais que ajudam na avaliação e nas melhorias do processo de software.
- Temos uma grande variedade de técnicas de teste que podem ser aplicadas em diferentes cenários.
- Podem ser utilizadas para classificar:
 - Diferentes conceitos de testes de software.
 - Técnicas que envolvem o design de testes e suas situações.
 - Técnicas de execução de teste e organizações de testes de software.
- Formas sistemáticas de exercitar o produto de software com o intuito de identificar seus defeitos.
- As Técnicas podem ser do tipo:
 - **Funcionais:** baseado em especificações.
 - **Estruturais:** baseado na análise da estrutura interna de um componente ou sistema.

Testes Funcionais

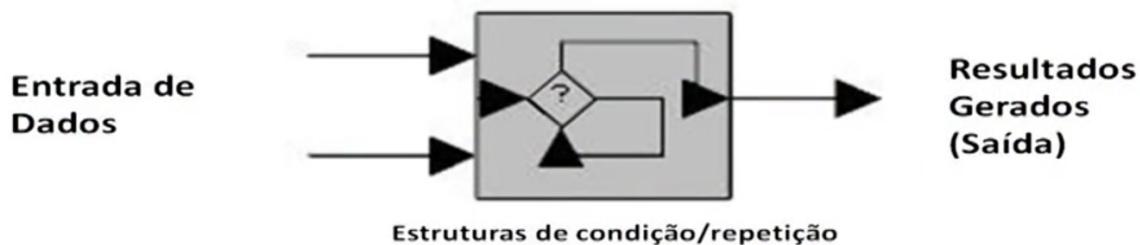
- Garantem que os requisitos foram atendidos, ou seja, que os requisitos estão corretamente codificados. São conhecidos como os testes de **Caixa Preta**:



Se o tipo de saída não ocorre, então houve uma situação de defeito.

Testes Estruturais

- Garantem que os sistemas sejam estruturalmente sólidos e que funcionem no contexto técnico onde serão instalados. São conhecidos como testes de **Caixa Branca**.



Evolução de Software

- Depois que o software é implantado, com certeza ocorrerão mudanças que levarão a realização de sua alteração.

Essas mudanças podem ocorrer para:

- Correção de erros não detectados durante a etapa de validação do software.
- Adaptação a um novo ambiente.
- O cliente solicita novas características ou funções.
- Quando a aplicação passa por um processo de reengenharia para proporcionar benefício em um contexto moderno.

Atualmente, os estágios de **desenvolvimento** e **manutenção** têm sido considerados **integrados** e **contínuos**:



- Após a implantação de um sistema, é inevitável que ocorram mudanças: pequenos ajustes após a implantação, para melhorias substanciais, por força da legislação, para atender novos requisitos dos usuários, ou por estar com erros.
- Para muitas empresas, os processos de manutenção são considerados menos desafiadores do que o desenvolvimento de um software original, ainda que tenha um custo mais elevado.
- A manutenção sempre vai existir e consumirá bastante tempo por parte da equipe de desenvolvimento.
- Os custos de manutenção podem ser maiores do que os custos de desenvolvimento inicial.
- Troca das pessoas que compõem as equipes de desenvolvimento.
- Falta de documentação ou incompleta ou desatualizada.

Por que a manutenção do software é importante?

- Ajustes pós-implantação
- Melhorias substanciais
- Atualização na Legislação
- Atender novos requisitos
- Erros no Software

Tipos de manutenção de software:

- Correção de defeitos:

Erros de codificação são relativamente baratos para serem corrigidos e erros de projeto são mais caros, pois podem implicar na reescrita de vários componentes de programa.

Erros de requisitos são os mais caros para se corrigir, devido ao extenso reprojeto de sistema que pode ser necessário.

- Adaptação ambiental

Esse tipo de manutenção é necessário quando algum aspecto do ambiente do sistema, como o hardware, a plataforma do sistema operacional ou outro software de apoio sofre uma mudança. O sistema de aplicação deve ser modificado para se adaptar a essas mudanças de ambiente.

- Adição de funcionalidade

Esse tipo de manutenção é necessário quando os requisitos de sistema mudam em resposta às mudanças organizacionais ou de negócios. A escala de mudanças necessários para o software é, frequentemente, muito maior do que para os outros tipo de manutenção.

Distribuição do esforço de manutenção

- As porcentagens podem variar de uma empresa para outra.
- A correção de defeitos não é a atividade de manutenção mais cara. Evoluir o software para lidar com os ambientes de tecnologias e novos ou alterar os requisitos consomem mais esforços de manutenção.

Configuração de Software

O Gerenciamento de Configuração de Software (SCM - Software Configuration Management) ou GCS é uma atividade de apoio destinada a:

- Controlar as mudanças
- Gerenciar as mudanças
- Realizar o controle de versão

- Analisar as relações entre os artefatos
- Identificar os artefatos que precisam ser alterados.
- Auditá e relatar todas as alterações feitas no software.

Qual é a diferença entre suporte de software e gerenciamento de configuração de software?

- **Suporte:** é um conjunto de atividades de engenharia que ocorrem depois que o software foi fornecido ao cliente e posto em operação.
- **Gestão de configuração:** é um conjunto de atividades de rastreamento e controle iniciadas quando um projeto de engenharia de software começa e termina apenas quando o software sai de operação.