



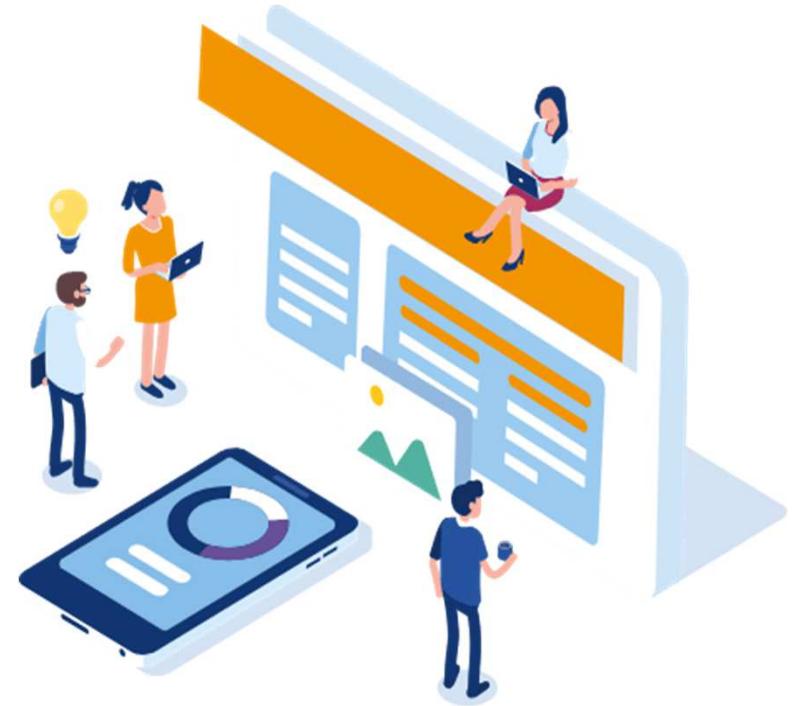
**ENGENHARIA DE SOFTWARE I**  
**SISTEMAS DE INFORMAÇÃO - 5º- PERÍODO - PASSOS/MG**

# ENGENHARIA DE SOFTWARE



## O Que é Por que ela é importante?

- É uma disciplina de engenharia que se preocupa com os aspectos da produção de software, desde a sua concepção inicial até a sua operação e manutenção.
- O desenvolvimento de diferentes tipos de sistemas de software pode exigir diferentes técnicas de engenharia.
- O desenvolvimento de software é uma atividade de engenharia.
- Software é um produto abstrato e intangível.
- **Engenharia de Software: 1968 (crise do software).**

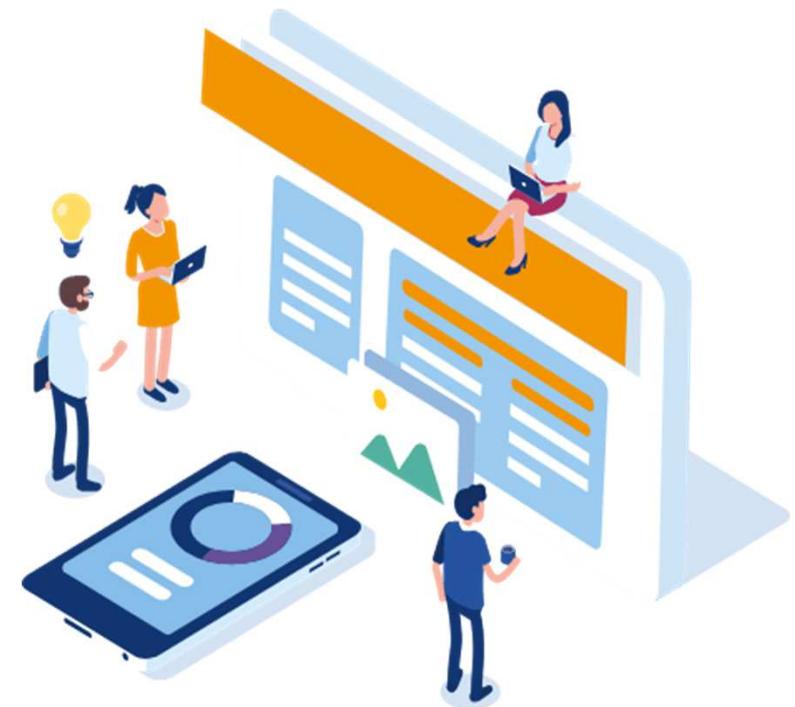


# ENGENHARIA DE SOFTWARE

## Crise de Software



- Foi um termo que surgiu nos anos 70 que expressava as dificuldades do desenvolvimento de software ao rápido crescimento da demanda por software, da complexidade dos problemas a serem resolvidos e da inexistência de técnicas estabelecidas para o desenvolvimento de sistemas que funcionassem adequadamente ou pudessem ser validados.

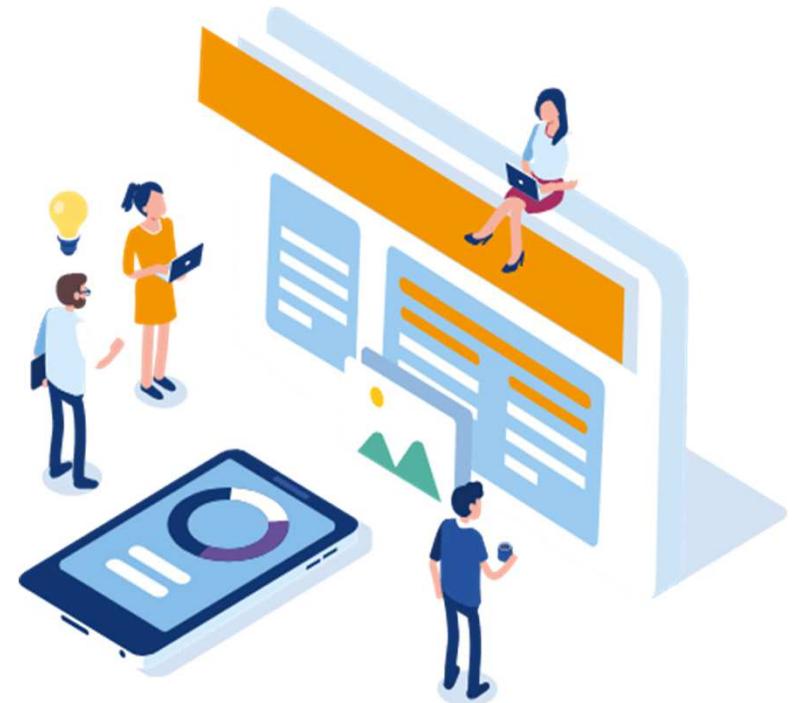


# ENGENHARIA DE SOFTWARE

## Crise de Software



- No Início dos anos 70, quando vivia-se a terceira era do software, houveram muitos problemas de prazo e custo no desenvolvimento de software, devido a baixa produtividade, baixa qualidade e difícil manutenção do software.
- A criação da Engenharia de Software surgiu numa tentativa de contornar a crise do software e dar um tratamento de engenharia(mais sistemático e controlado) ao desenvolvimento de sistemas de software complexos.

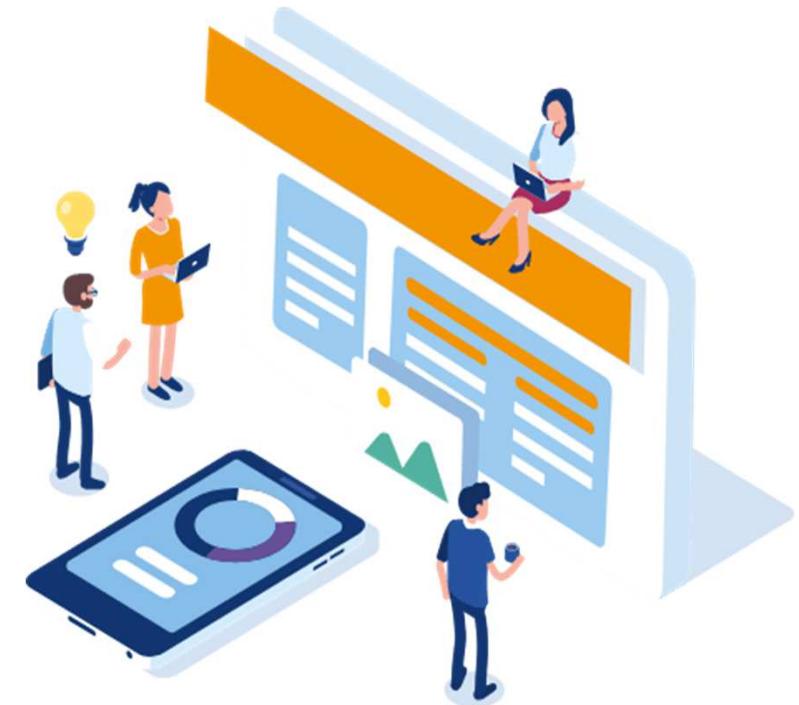


# ENGENHARIA DE SOFTWARE



## Os problemas no desenvolvimento de software

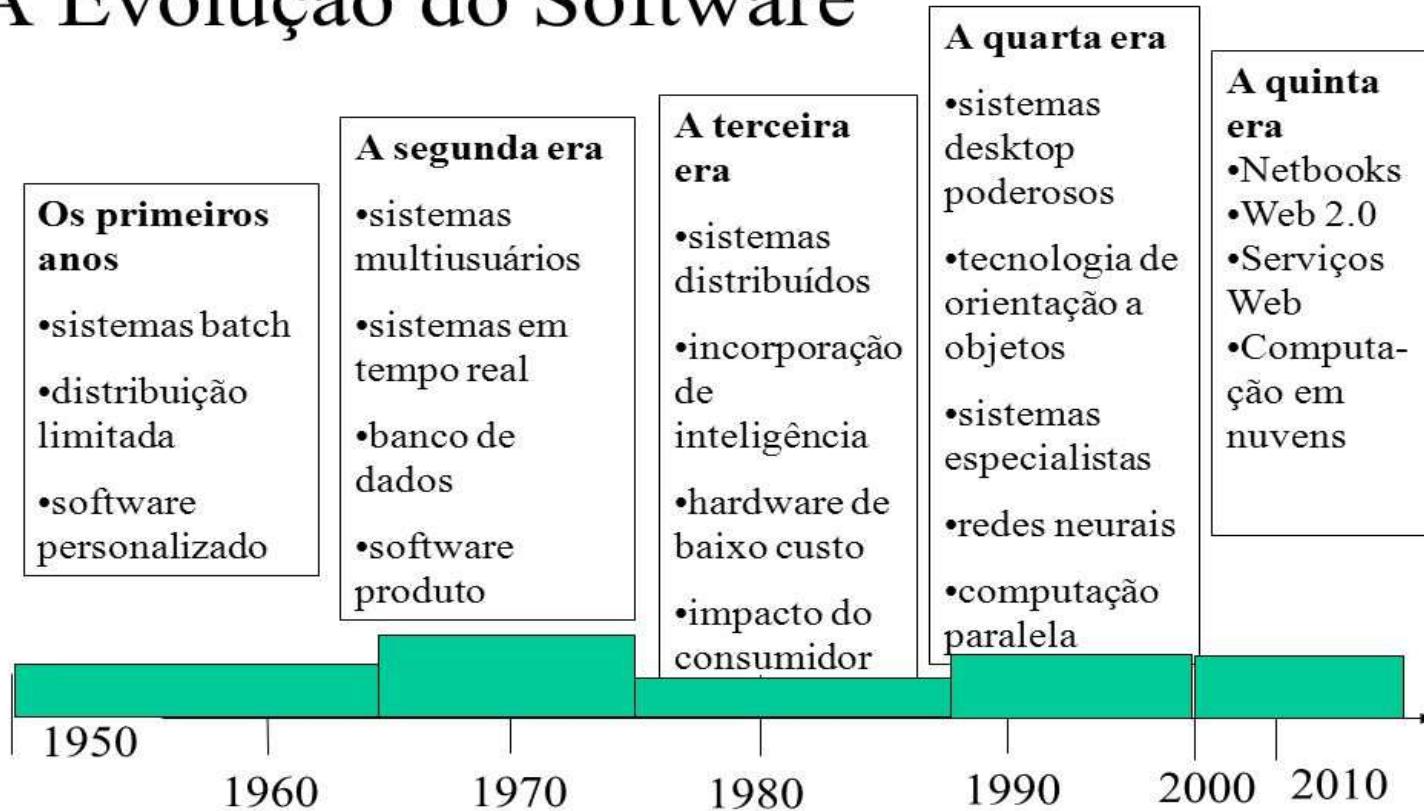
- Estimativas de prazo e de custo imprecisas.
- Produtividade das pessoas da área de software não acompanha a demanda.
- Prazos ultrapassados.
- Custos acima do previsto.
- A facilidade de manutenção não era enfatizada como um critério importante, gerando assim custos de manutenção elevados.
- Não atendimento dos requisitos do usuário.
- 1/3 dos projetos eram cancelados.
- 2/3 dos projetos extrapolavam o orçamento.



# ENGENHARIA DE SOFTWARE



## A Evolução do Software



# ENGENHARIA DE SOFTWARE



## O que é software?

- Segundo Pressman (2006), sob o ponto de vista da Engenharia de Software, um software é um conjunto composto por instruções de computador, estruturas de dados e documentos. As instruções (ou código) produzem os resultados esperados, de acordo com os requisitos definidos.



# ENGENHARIA DE SOFTWARE



## O que é software?

- Programas de computador e documentação associada.
- Produtos de Software podem ser desenvolvidos para um cliente específico ou para o mercado.
- Produtos de software podem ser:
  - Genéricos que são produzidos e vendidos no mercado a qualquer cliente.
  - Produtos sob encomenda (ou personalizados) que são encomendados por um cliente em particular (especificação é desenvolvida e controlada pela organização que está comprando o software)



# ENGENHARIA DE SOFTWARE



## Quais são os atributos de um bom software?

- O software deve atender os requisitos funcionais e desempenho que foram solicitados pelo usuário e além disso deve atender requisitos não funcionais, tais como facilidade de manutenção, nível de confiança, eficiência e facilidade de uso.
- Facilidade de manutenção
  - Software deve ser escrito de modo que possa evoluir para atender as necessidades mutáveis.



# ENGENHARIA DE SOFTWARE



## Quais são os atributos de um bom software?

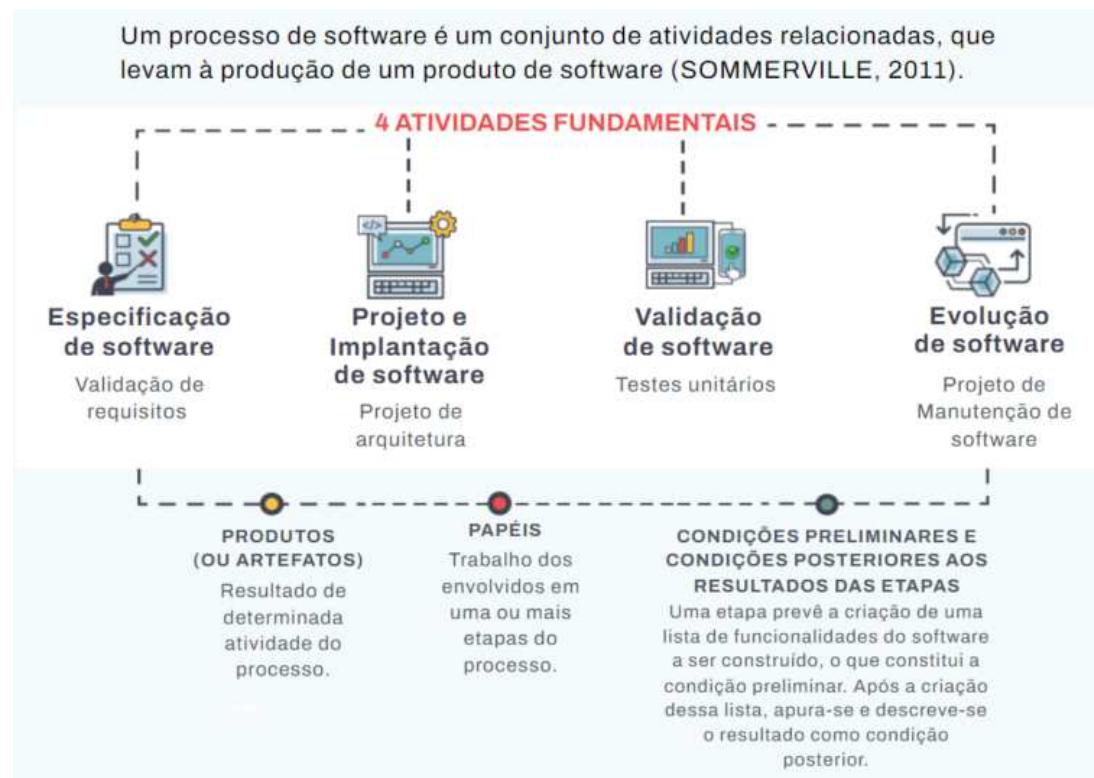
- **Nível de confiança**
  - Software confiável não deve ocasionar danos físicos ou econômicos, no caso de um defeito no sistema.
- **Eficiência**
  - O Software não deve desperdiçar os recursos do sistema.
- **Facilidade de uso**
  - O Software deve ser utilizável, sem esforços indevidos, pelo usuário para quem foi projetado.



# ENGENHARIA DE SOFTWARE



## Quais são as atividades fundamentais da Engenharia de Software?



# ENGENHARIA DE SOFTWARE



## Qual é a diferença entre Engenharia de Software e Sistemas de Informação?

- A área de sistemas de informação está relacionada ao planejamento e implementação de sistemas já a engenharia de software se preocupa com as questões práticas de desenvolver e entregar software útil.



# ENGENHARIA DE SOFTWARE



## Quais os principais desafios enfrentados pela Engenharia de Software?

- Lidar com sistemas legados, lidar com a diversidade crescente e lidar com a crescente demanda e reduzir o tempo de entrega.
- Sistemas legados
- Sistemas antigos, porém úteis devem ser mantidos e atualizados
- Heterogeneidade
- Sistemas são distribuídos e inclui uma mistura de hardware e software. Deve-se desenvolver técnicas para construir softwares confiáveis e flexíveis.
- Fornecimento
- Existe pressão crescente para uma entrega rápida do software.



# ENGENHARIA DE SOFTWARE

## Quais são os melhores métodos e técnicas da Engenharia de Software?



- Ainda que todos os projetos de software devam ser gerenciados e desenvolvidos profissionalmente, técnicas diferentes são adequadas para tipos diferentes de sistemas.
- Por exemplo, jogos devem ser sempre desenvolvidos usando uma série de protótipos, enquanto sistemas de controle críticos em segurança requerem o desenvolvimento de uma especificação completa e analisável.
- Não há métodos ou técnicas que sejam bons para todos os casos.



# ENGENHARIA DE SOFTWARE



## Quais as diferenças que a internet trouxe para a Engenharia de Software?

- A internet não só levou ao desenvolvimento de sistemas massivos, largamente distribuídos, baseados em serviços (APIs), como também deu base para a criação de uma indústria de aplicativos (APPs) para dispositivos móveis que mudou a economia de software



## Por que Engenharia de Software?

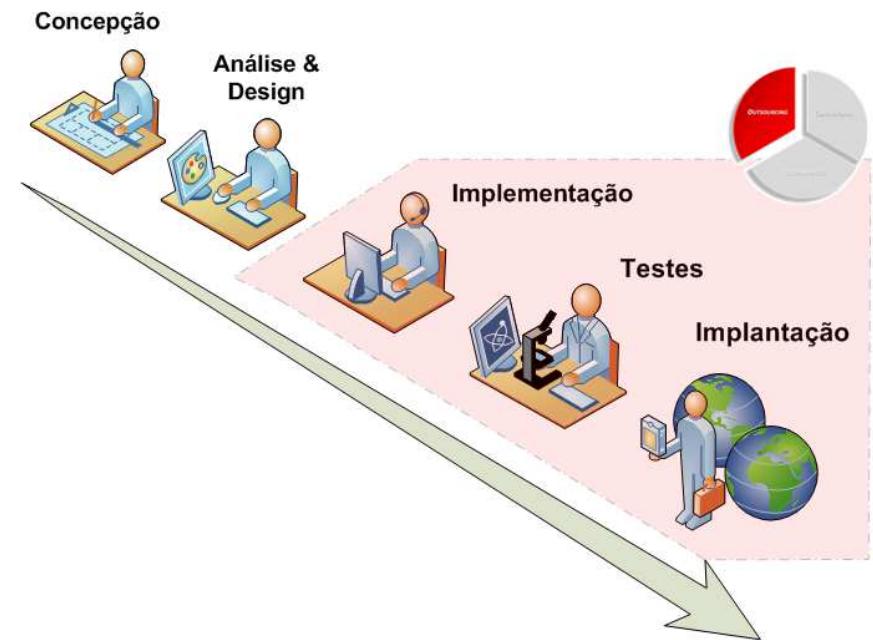
- Ela é importante pois queremos um software com qualidade, baixo custo/orçamento e entregue em pouco tempo/cronograma.

# ENGENHARIA DE SOFTWARE

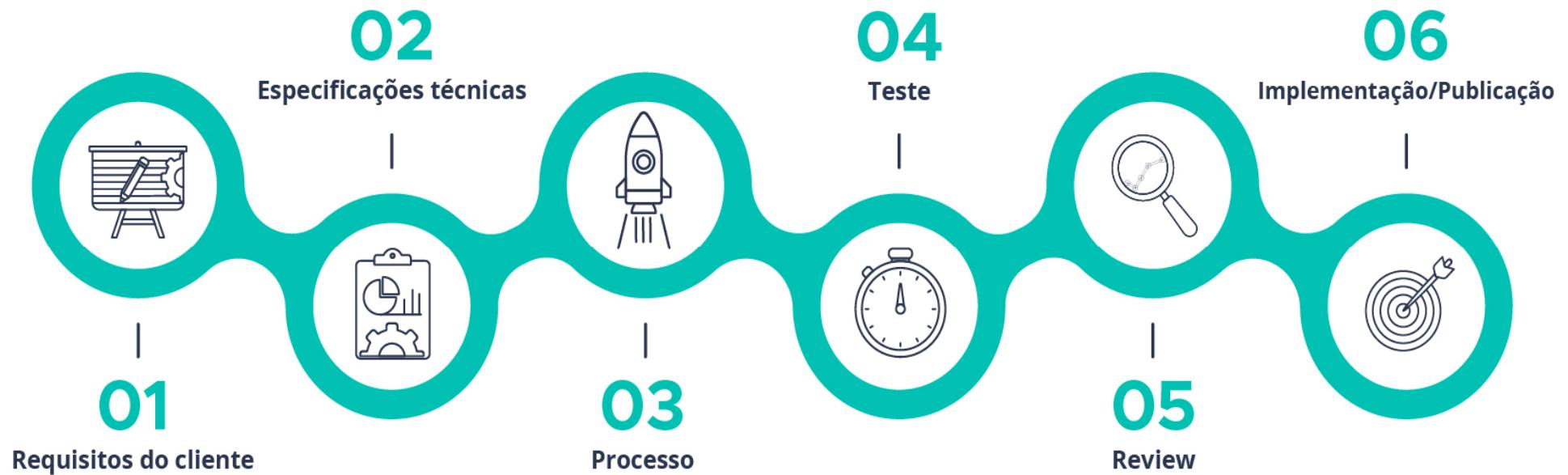


## Processo de Software

- São um conjunto de tarefas objetivas e atividades que visam a criação de um software estruturado e com qualidade, para um boa manutenção e funcionalidade do software.
- Processos de software formam a base para o controle gerencial de projetos de software e estabelece o conteúdo no qual os métodos técnicos são aplicados, os produtos de trabalho (modelos, documentos, dados, relatórios, formulários, etc.) são produzidos, os marcos são estabelecidos, a qualidade é assegurada e as modificações são adequadamente geridas (PRESSMAN, 2006).



# ENGENHARIA DE SOFTWARE



Fonte: <https://artia.com/wp-content/uploads/2022/03/6-passos-do-processo-de-desenvolvimento-de-software.jpg>

# ENGENHARIA DE SOFTWARE



## Atividades do Processo de Software

- Em cada fase de um processo de software definido são executadas as atividades básicas para que sejam atingidos os objetivos propostos. Segundo Pressman (2006) estas atividades constituem um conjunto mínimo para se obter um produto de software.
- **Atividades genéricas a todos processos de software são:**
  - **Especificação:** a funcionalidade do software e as restrições em sua operação devem ser definidas
  - **Desenvolvimento:** produção do software de modo que atenda a suas especificações.
  - **Validação:** o software deve ser validado para garantir que ele faz o que o cliente deseja.
  - **Evolução:** o software deve evoluir para atender às necessidades mutáveis do cliente.



# ENGENHARIA DE SOFTWARE



- Projeto de Software
  - **Projeto Arquitetural:** onde é desenvolvido um modelo conceitual para o sistema, composto de módulos mais ou menos independentes.
  - **Projeto de Interface:** onde cada módulo tem sua interface de comunicação estudada e definida.
  - **Projeto Detalhado:** onde os módulos em si são definidos, e possivelmente traduzidos para pseudocódigo.
- Desenvolvimento/Implementação
  - **Codificação:** a implementação em si do sistema em uma linguagem de computador.



# ENGENHARIA DE SOFTWARE



- **Validação/Homologação**
  - Teste de Unidade e Módulo: a realização de testes para verificar a presença de erros e comportamento adequado a nível das funções e módulos básicos do sistema.
  - Integração: a reunião dos diferentes módulos em um produto de software homogêneo, e a verificação da interação entre estes quando operando em conjunto.
- **Manutenção e Evolução**
  - Nesta fase, o software em geral entra em um ciclo iterativo que abrange todas as fases anteriores.
  - Desta forma as atividades relacionadas a um processo de software estão diretamente vinculadas com a produção do software como produto final . Afim de especificar quais atividades devem ser executadas e em qual ordem temos diversos modelos de desenvolvimento de software.



# ENGENHARIA DE SOFTWARE

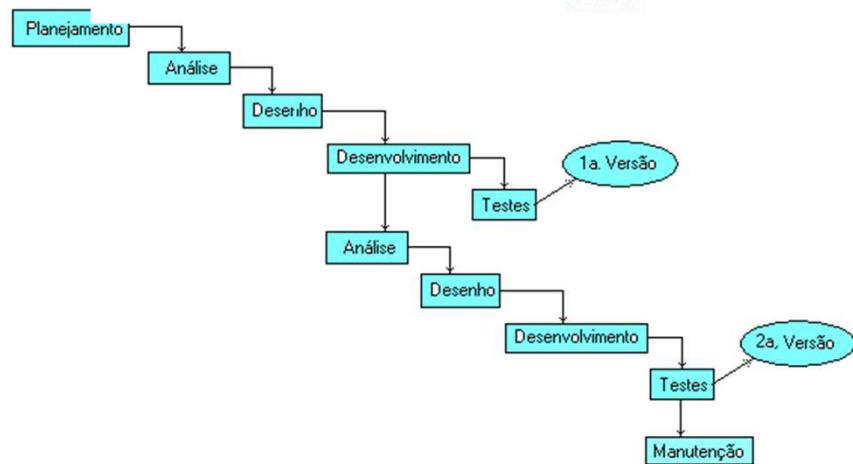
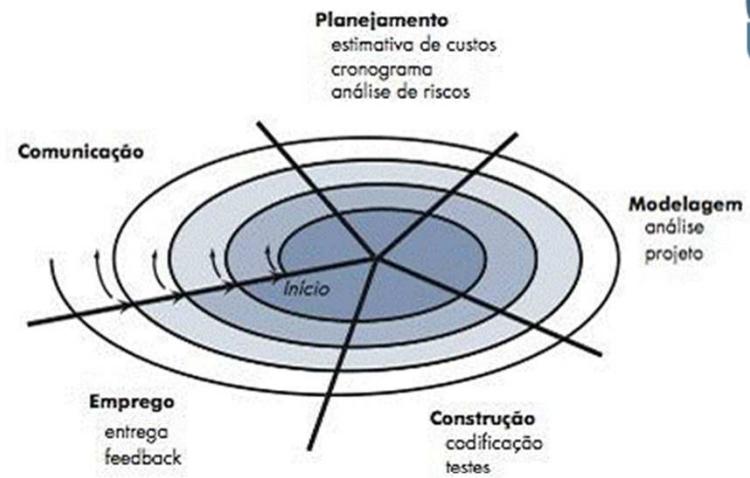
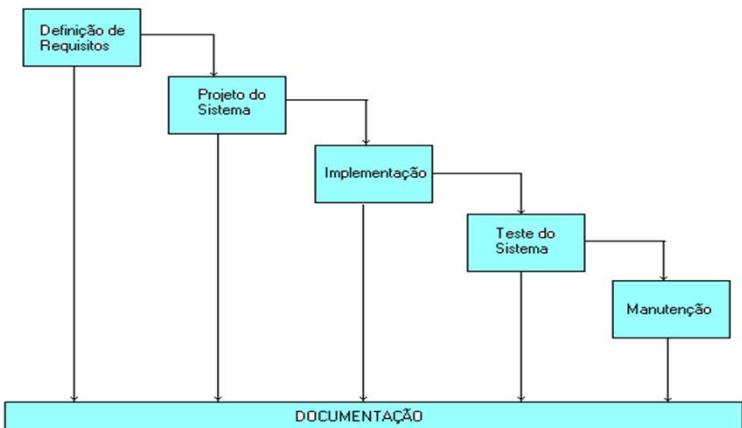


## Modelos de Processos de Software

- Surgiram pela necessidade de dar resposta às situações a analisar, porque só na altura em que enfrentamos o problema é que podemos escolher o modelo. Cascata (Royce, 1970), Espiral (Boehm, 1988), Iterativo e Incremental (RAD, 1991 RUP, 2003), Prototipagem (Brooks, 1975).
- Nos modelos de processo de software é dado uma atenção especial à representação abstrata dos elementos do processo e sua dinâmica, não estabelecendo métodos de desenvolvimento, pois este trabalha num nível mais alto de abstração do que os modelos de ciclo de vida



# ENGENHARIA DE SOFTWARE

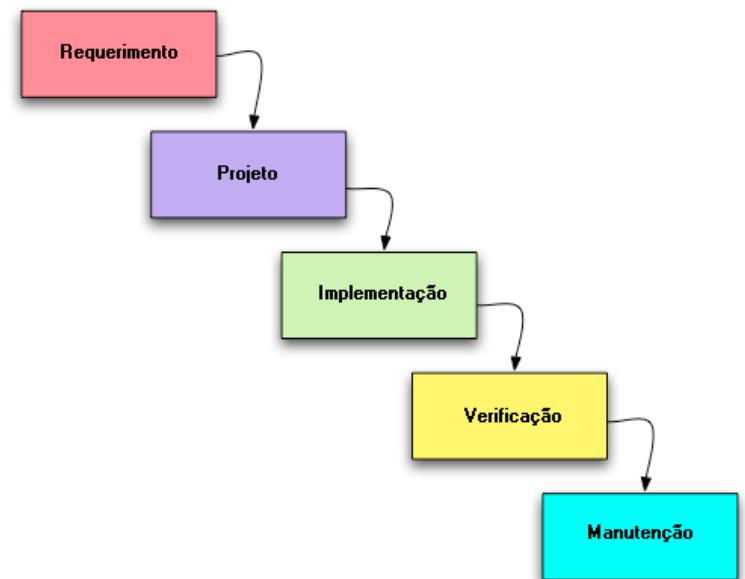


# ENGENHARIA DE SOFTWARE



## Modelo Cascata (Waterfall)

- O modelo cascata ou clássico também pode ser conhecido como “waterfall”, tendo sido criado na década de 1970 por Royce, sendo o modelo mais aceito até a metade da década de 1980.
- Esse modelo é oriundo de outros modelos de atividades de engenharia com a finalidade de determinar a ordem no desenvolvimento de grandes produtos de software.
- Comparado com os outros modelos de criação de software, o cascata se caracteriza por ser mais rígido e menos administrativo.
- O cascata é um dos modelos mais importantes e utilizados como referência para a criação de outros modelos de desenvolvimento de software, funcionando como base para os demais projetos modernos.

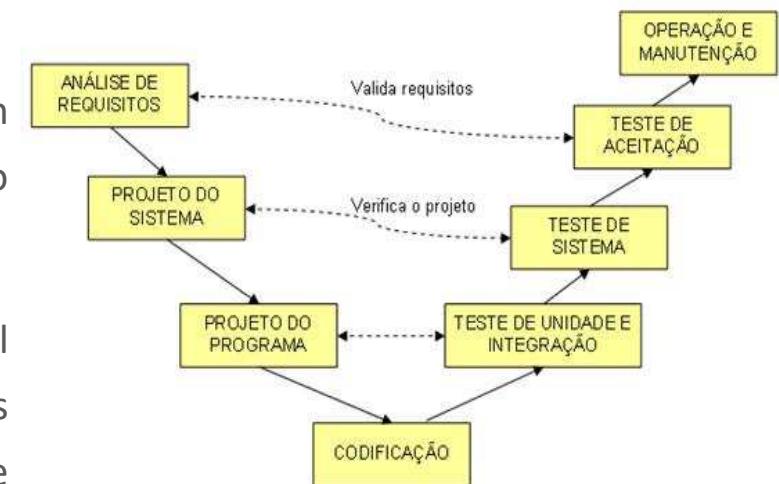


# ENGENHARIA DE SOFTWARE

## Modelo em “V”



- Neste modelo, do Ministério de Defesa da Alemanha, 1992, o modelo em cascata é colocado em forma de "V".
- Do lado esquerdo do V ficam da análise de requisitos até o projeto, a codificação fica no vértice e os testes, desenvolvimento, implantação e manutenção, à direita.
- A característica principal desse modelo, que o diferencia do modelo em cascata, é a ênfase dada à verificação e validação: cada fase do lado esquerdo gera um plano de teste a ser executado no lado direito.
- Mais tarde, o código fonte será testado, do mais baixo nível ao nível sistêmico para confirmar os resultados, seguindo os respectivos planos de teste: o teste de unidade valida o projeto do programa, o teste de sistema valida o projeto de sistema e o teste de aceitação do cliente valida a análise de requisitos.



# ENGENHARIA DE SOFTWARE



## Modelo de Prototipação

- O objetivo é entender os requisitos do usuário e assim, obter uma melhor definição dos requisitos do sistema. Possibilita que o desenvolvedor crie um modelo (protótipo) do software que deve ser construído.
- Apropriado para quando o cliente não definiu detalhadamente os requisitos.
- Idealmente, o protótipo serve como um mecanismo para identificação dos requisitos de software. Se um protótipo de trabalho for construído, o desenvolvedor tentará usar fragmentos de programas existentes ou aplicará ferramentas que possibilitem que programas de trabalho sejam gerados rapidamente

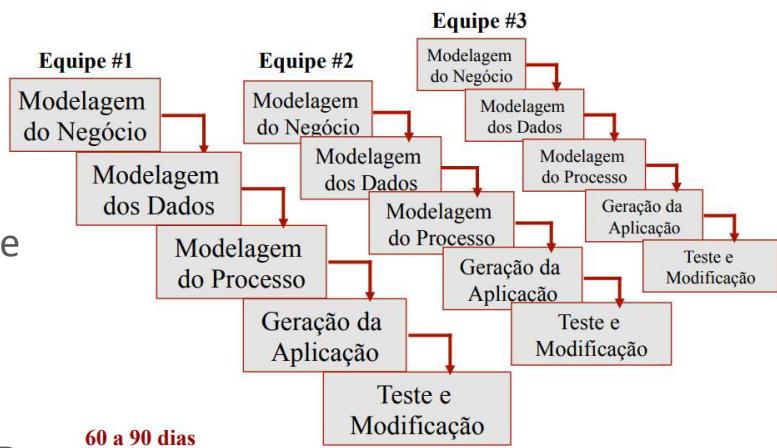


# ENGENHARIA DE SOFTWARE



## Modelo RAD (Rapid Application Development)

- É um modelo sequencial linear que enfatiza um ciclo de desenvolvimento extremamente curto. O desenvolvimento rápido é obtido usando uma abordagem de construção baseada em componentes.
- Os requisitos devem ser bem entendidos e o alcance do projeto restrito
- O modelo RAD é usado principalmente para aplicações de sistema de informação.
- Cada função principal pode ser direcionada para uma equipe RAD separada e então integrada para formar o todo.

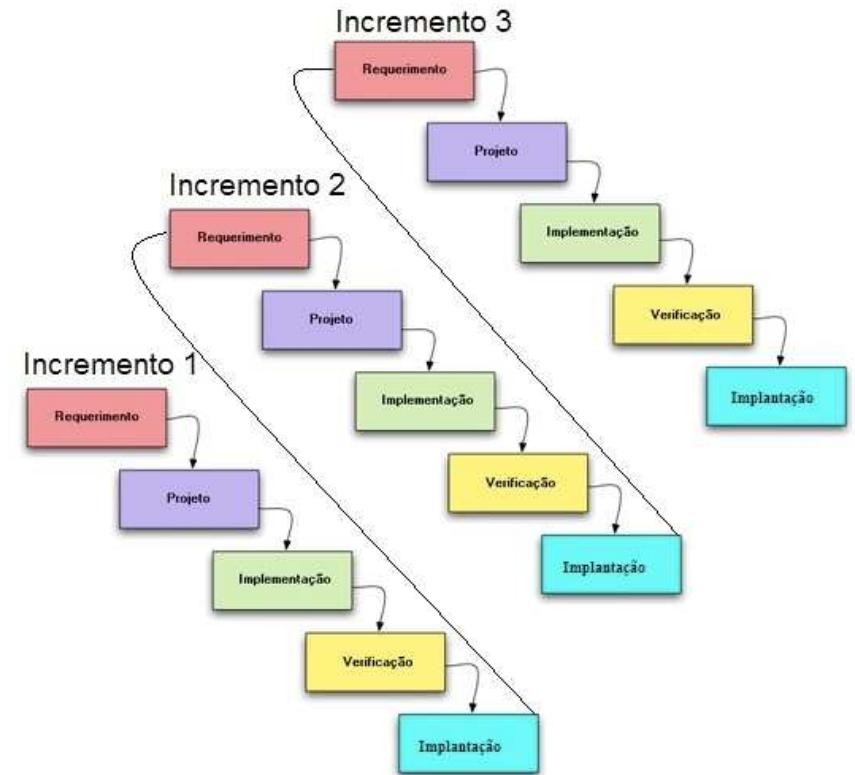


# ENGENHARIA DE SOFTWARE

## Modelo Iterativo e Incremental



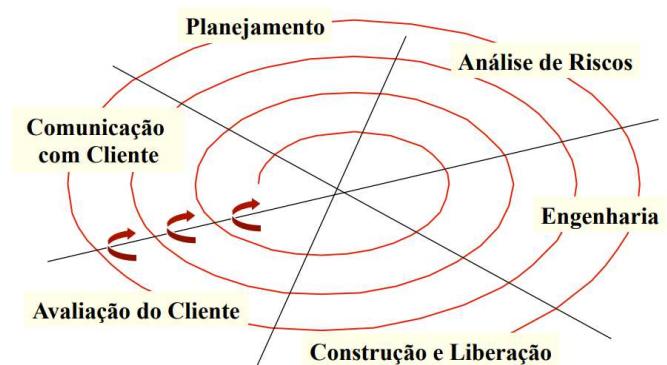
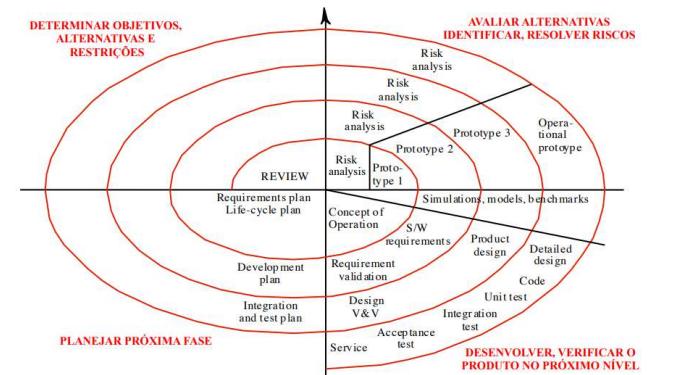
- Combina elementos do modelo cascata (aplicado repetidamente) com a filosofia iterativa da prototipação. O objetivo é trabalhar junto ao usuário para descobrir seus requisitos, de maneira incremental, até que o produto final seja obtido.
- A versão inicial é frequentemente o núcleo do produto (a parte mais importante). A evolução acontece quando novas características são adicionadas à medida que são sugeridas pelo usuário.
- Este modelo é importante quando é difícil estabelecer a priori uma especificação detalhada dos requisitos. O modelo incremental é mais apropriado para sistemas pequenos
- As novas versões podem ser planejadas de modo que os riscos técnicos possam ser administrados (ex: disponibilidade de determinado hardware)



# ENGENHARIA DE SOFTWARE

## Modelo Espiral (Evolutivo)

- Acopla a natureza iterativa da prototipação com os aspectos controlados e sistemáticos do modelo cascata.
- É dividido em uma série de atividades de trabalho ou regiões de tarefa, onde existem tipicamente de 3 a 6 regiões de tarefa.
- São definidos objetivos específicos para a fase do projeto.
- São identificadas restrições sobre o processo e o produto, projetando um plano de gerenciamento detalhado.
- São identificados riscos do projeto dependendo dos riscos, estratégias alternativas podem ser planejadas.
- É, atualmente, a abordagem mais realística para o desenvolvimento de software em grande escala.
- Usa uma abordagem que capacita o desenvolvedor e o cliente a entender e reagir aos riscos em cada etapa evolutiva.

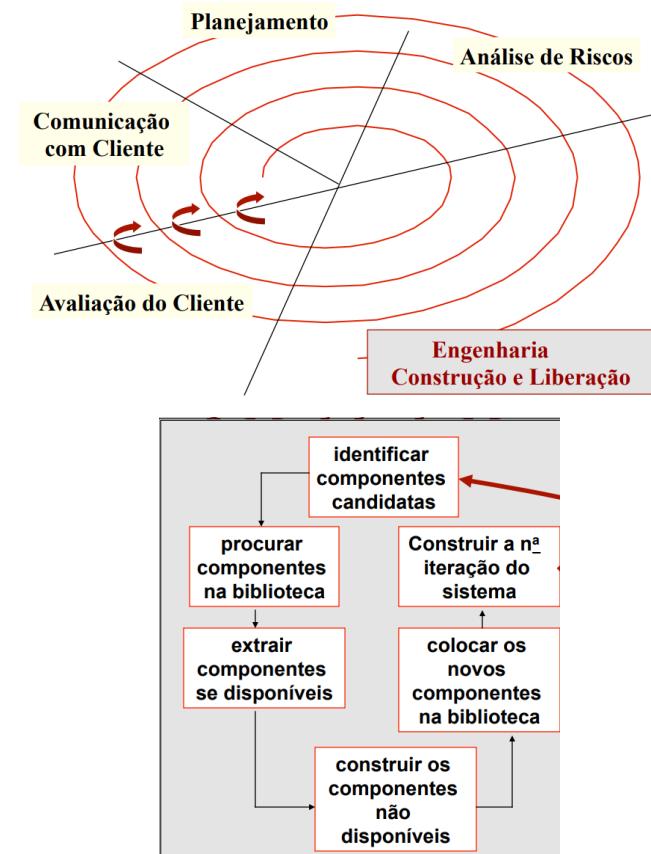


# ENGENHARIA DE SOFTWARE

## Modelo de Montagem de Componentes



- Utiliza tecnologias orientadas a objeto.
- Quando projetadas e implementadas apropriadamente as classes orientadas a objeto são reutilizáveis em diferentes aplicações e arquitetura de sistema.
- O modelo de montagem de computadores incorpora muitas das características do modelo espiral
- Conduz ao reuso do software, onde a reusabilidade fornecer uma série de benefícios
  - Redução de 70% no tempo de desenvolvimento
  - Redução de 84% no custo de projetos
- Esses resultados dependem da robustez da biblioteca de componentes

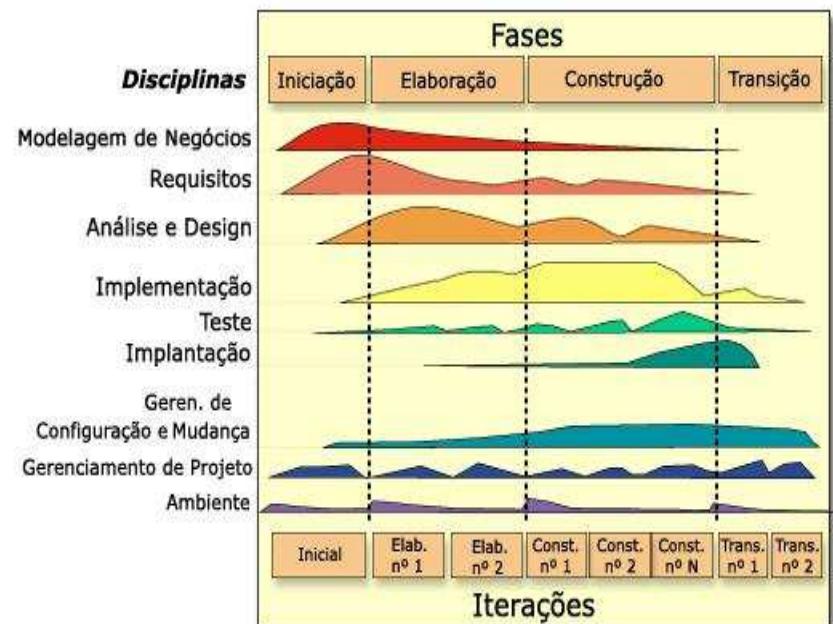


# ENGENHARIA DE SOFTWARE



## RUP - Rational Unified Process

- É uma metodologia (ou processo) iterativa, onde cada iteração representa um ciclo completo de desenvolvimento, desde a captação de requisitos na análise até a implementação e a realização de testes, resultando numa versão executável do sistema.
- Por sua vez, as iterações são agrupadas em fases. Uma fase é o período de tempo entre dois marcos de progresso, onde objetivos são alcançados, artefatos são concluídos e decisões são tomadas em relação à passagem para a fase seguinte.
- Criado pela Rational Software Corporation, fornece-nos técnicas a serem seguidas pelos membros da equipe de desenvolvimento de software com o objetivo de aumentar a sua produtividade.

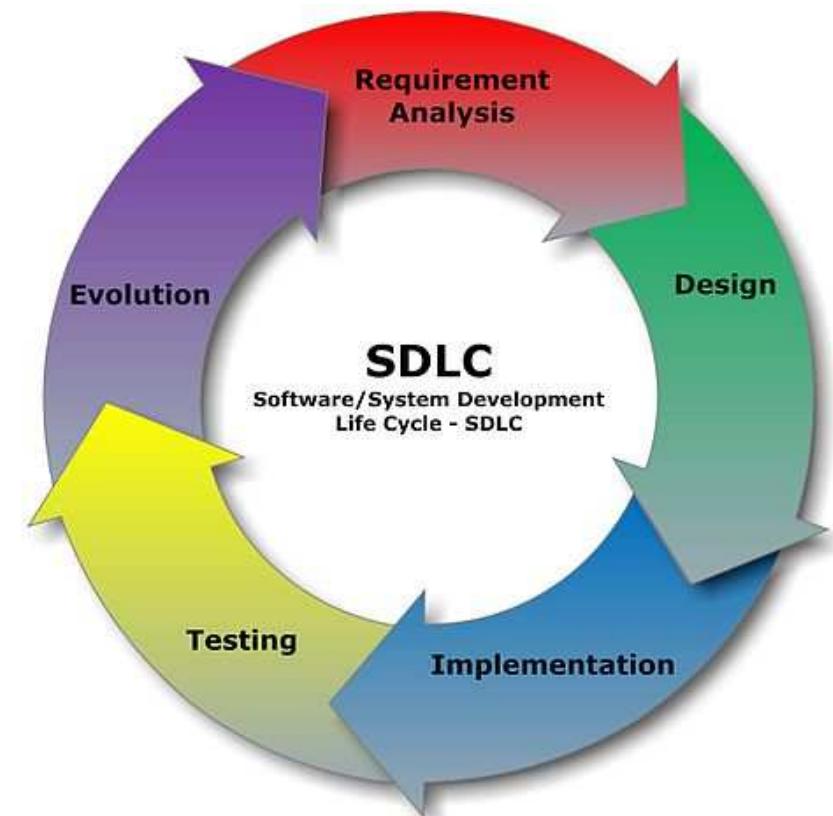


# ENGENHARIA DE SOFTWARE



## Ciclo de Vida de Software (SDLC)

- 1- Fase de requisitos : levanta os requisitos mínimos, estuda a viabilidade e define o modelo a ser usado.
- 2- Fase de projeto : Envolve atividades de concepção, especificação, design da interface, prototipação, design da arquitetura;
- 3- Fase de implementação : tradução para uma linguagem de programação das funcionalidades definidas durante as fases anteriores.
- 4- Fase de testes : realização de testes no que foi desenvolvido de acordo com os requisitos.
- 5- Fase de produção : implantação em produção do produto final.

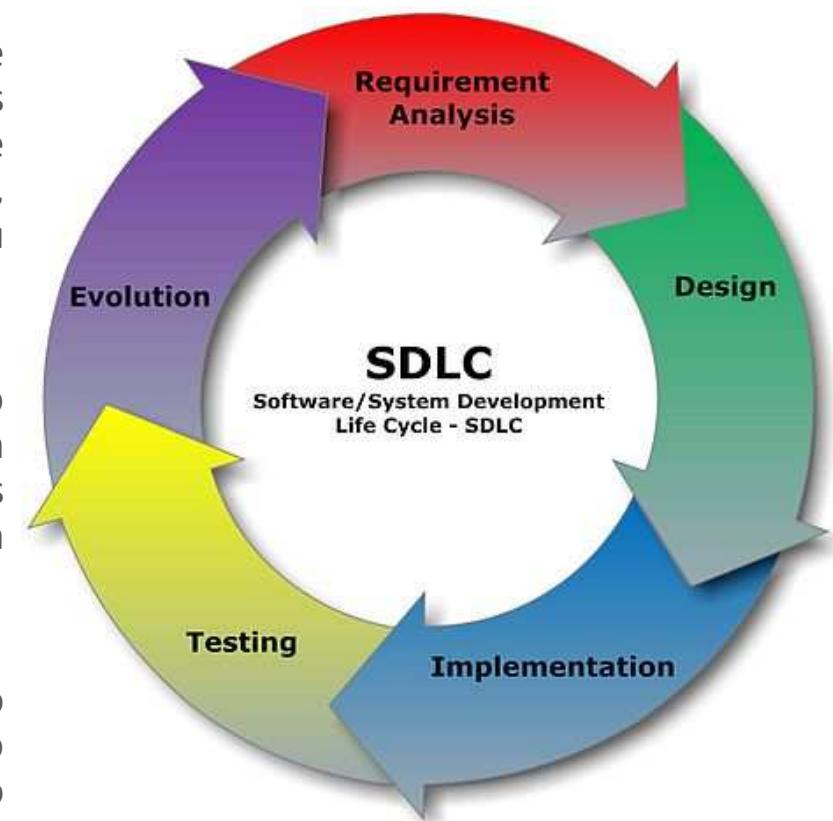


# ENGENHARIA DE SOFTWARE

## Ciclo de Vida de Software (SLDC)



- De acordo com a NBR ISO/IEC 12207:1998, o ciclo de vida é a “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.
- O modelo de ciclo de vida é a primeira escolha a ser feita no processo de software. A partir desta escolha definir-se-á desde a maneira mais adequada de obter as necessidades do cliente, até quando e como o cliente receberá sua primeira versão operacional do sistema.
- Não existe um modelo ideal. O perfil e complexidade do negócio do cliente, o tempo disponível, o custo, a equipe, o ambiente operacional são fatores que influenciarão diretamente na escolha do ciclo de vida de software a ser adotado.

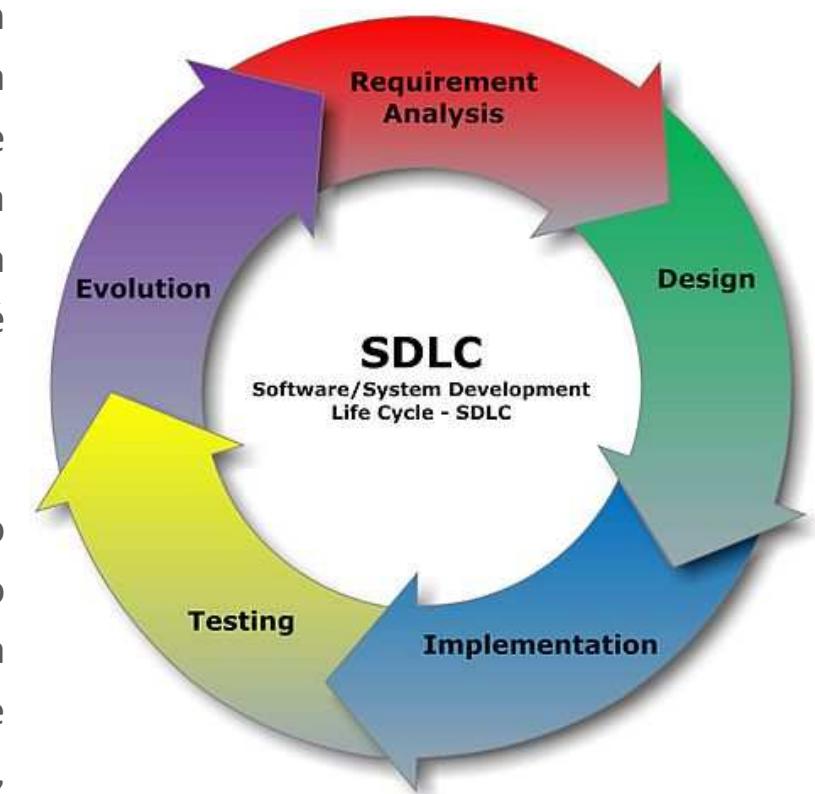


# ENGENHARIA DE SOFTWARE

## Ciclo de Vida de Software (SLDC)



- Da mesma forma, também é difícil uma empresa adotar um único ciclo de vida. Na maior parte dos casos, vê-se a presença de mais de um ciclo de vida no processo. Os ciclos de vida se comportam de maneira sequencial (fases seguem determinada ordem) e/ou incremental (divisão de escopo) e/ou iterativa (retroalimentação de fases) e/ou evolutiva (software é aprimorado).
- De maneira menos formal o SLDC é um processo reconhecido pela indústria como uma série de atividades ou etapas para o desenvolvimento de um novo produto software ou para modificar um software existente. A principal função do ciclo de vida do desenvolvimento de software é indicar as fases, atividades, entregas e responsabilidades de cada envolvido no processo de desenvolvimento de software.



# ENGENHARIA DE SOFTWARE



## Gerenciamento do Ciclo de Vida de Aplicações (ALM) Application Lifecycle Management

- Refere-se à capacidade de integrar, coordenar e controlar as diversas fases de desenvolvimento de um software até a entrega. Gerenciar o ciclo de vida das aplicações é fazer a integração entre a necessidade de atender o negócio e a engenharia de software.



# ENGENHARIA DE SOFTWARE



## Gerenciamento do Ciclo de Vida de Aplicações (ALM) Application Lifecycle Management

- Representa um único processo que abrange todos elementos envolvidos no processo de desenvolvimento de software, oferecendo um trabalho contínuo e sempre com novos recursos. Atua no acompanhamento de todo o processo de vida útil de uma aplicação, seja a construção de novos softwares ou alterações nos softwares já existentes, integrando todas as fases da engenharia de software.



# ENGENHARIA DE SOFTWARE



## Gerenciamento do Ciclo de Vida de Aplicações (ALM) Application Lifecycle Management

- Podemos dividir o ALM em 3 áreas distintas: Governança, Desenvolvimento e Operações
  - Governança** - A governança engloba toda a tomada de decisões e gerenciamento de projetos em toda a empresa.
  - Desenvolvimento** - O desenvolvimento é definido como o processo de criação da aplicação real.
  - Operações** - Uma operação é o trabalho necessário para executar e gerenciar o aplicativo. Começa pouco antes implantação e depois é executado continuamente.

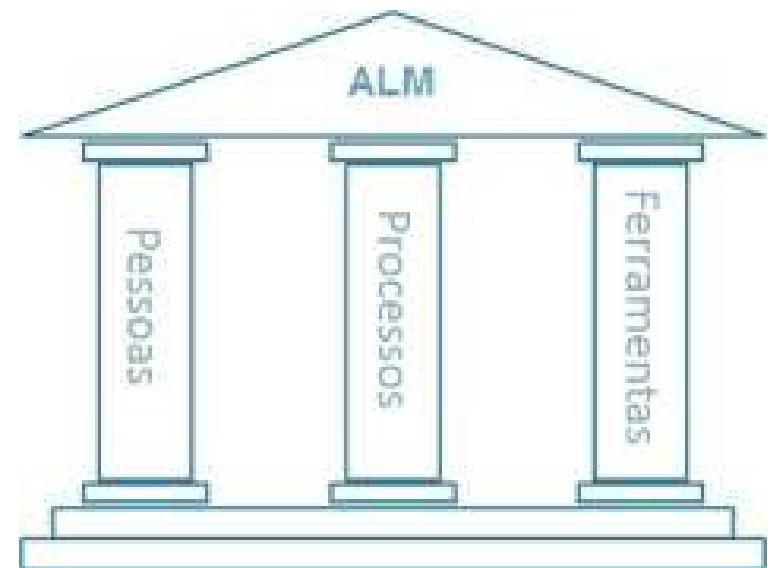


# ENGENHARIA DE SOFTWARE



## **Gerenciamento do Ciclo de Vida de Aplicações (ALM)** **Application Lifecycle Management**

- Os pilares que estruturam o ALM e trabalham de forma complementar para prover o gerenciamento do ciclo de vida de aplicações são: Pessoas, Processos e Ferramentas
- **Pessoas:** o time envolvido na gestão do ciclo de vida da aplicação inclui atribuições desde o alinhamento estratégico até a execução do desenvolvimento de sistemas.
- **Processos:** conjunto de boas práticas, artefatos, guias e manuais que orientam o desenvolvimento e manutenção de uma aplicação
- **Ferramentas:** engloba meios/equipamentos/tecnologias que automatizam e facilitam a condução dos processos pelas pessoas

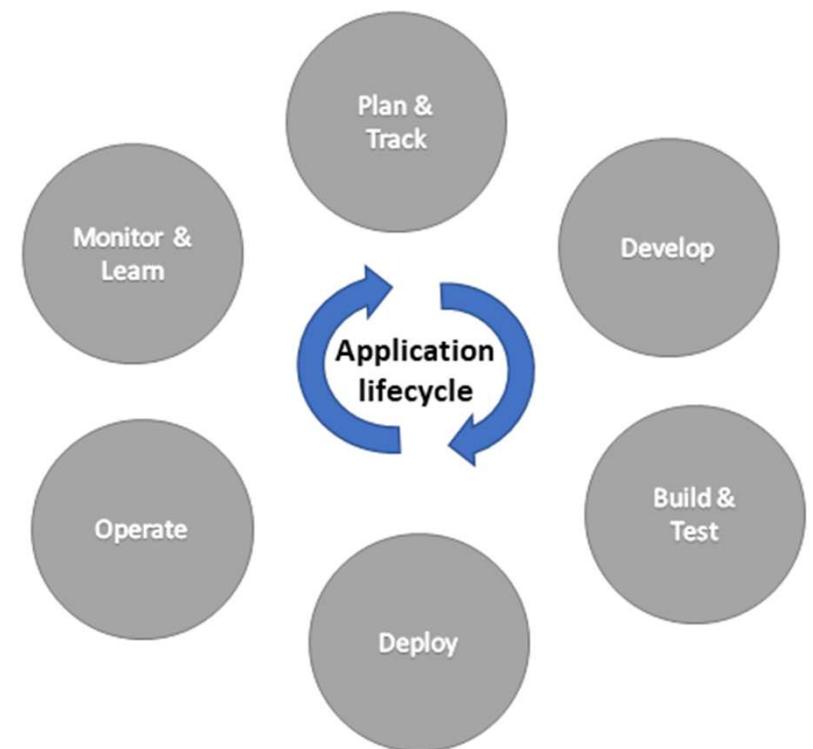


# ENGENHARIA DE SOFTWARE

## Diferença entre SLDC e ALM



- **SDLC (Software Development Lifecycle Management)** inclui os processos relacionados ao **DESENVOLVIMENTO** da aplicação como requisitos, arquitetura, codificação, testes e gestão da configuração/projeto.
- **ALM (Application Lifecycle Management)** inclui todo o ciclo de vida da **APLICAÇÃO**, desde a ideia do negócio, identificação da necessidade, riscos e desafios. Este ciclo geralmente é encerrado quando a aplicação não é mais utilizada pelo negócio, potencialmente alguns anos após o desenvolvimento inicial.
- No contexto empresarial (grandes, médias, pequenas empresas, startups, etc.), considerando a dificuldade de construir aplicações com a qualidade e valor desejados pelo usuário, o ALM pode ser então definido como o processo que guia a vida útil de uma aplicação desde a sua concepção, passando pela construção, operação e manutenção evolutiva.



# ENGENHARIA DE SOFTWARE



- Segundo Pressman (2006), um software é um conjunto composto por instruções de computador, estruturas de dados e documentos.
- O software é o conjunto de vários artefatos e não apenas o código fonte (SOMMERVILLE, 2003).
- O software é caro porque torna-se uma atividade difícil e trabalhosa de ser realizado pelo engenheiro de software (JALOTE, 2005).

## Produtos de Software pode ser:

- **Genéricos:** desenvolvidos para serem vendidos em uma grande variedade de clientes, por exemplo, softwares para PC, tais como Excel e Word.
- **Personalizados:** desenvolvidos para um único cliente de acordo com as suas especificações.



# ENGENHARIA DE SOFTWARE



- Segundo Pressman (2006), os softwares estão categorizados nos seguintes tipos:
- **Software de sistema:** são programas que apoiam outros programas, como o software que realiza a comunicação entre hardware e software (sistema operacional) ou que ajuda na construção de outro software (compiladores).
- **Software de aplicação:** são programas que são desenvolvidos para executar no negócio de uma determinada empresa.
- **Software científico e de engenharia:** são algoritmos que processam números.
- **Software embutido:** são programas construídos para executarem dentro de um produto específico como as teclas digitais de um forno micro-ondas



# ENGENHARIA DE SOFTWARE



- **Software para linhas de produtos:** são conhecidos como software de prateleiras.
- **Software de web:** são aplicativos executados via Internet.
- **Software de inteligência artificial:** são os que fazem uso de algoritmos não numéricos e que se encaixam na robótica.
- **Software aberto:** são os que disponibiliza a visualização do código fonte da aplicação para a modificação da maneira como deseja.
- **Software legado:** nome que é dado a um programa de computador que foi desenvolvido há muito e permanece até hoje em uma empresa.
- **Computação ubíqua:** são softwares que realiza a verdadeira computação distribuída.



# ENGENHARIA DE SOFTWARE



- Outra forma de classificar os produtos de software é como ele resolve o problema e coloca o usuário no centro, sendo um ponto de vista com três tipos de produto de software.
- **Para o consumidor final:** que está disposto a pagar uma taxa para usar os serviços.
- **Alguns exemplos são o Netflix, LinkedIn e Jogos.**
- Podemos pensar também em produtos web para consumidores finais que pagam a taxa pelo uso de forma indireta, ou seja, a taxa é paga por um serviço maior e o produto web está embutido no serviço.
- **Alguns exemplos são:** Internet Banking, Intranets de Universidades, acesso a resultado de Exames Laboratoriais, Sites de Comércio Eletrônico que são também dessa categoria.



# ENGENHARIA DE SOFTWARE



- **Para as empresas:** esse tipo de produto de software resolve o problema de empresas que normalmente tem mais disposição de pagar que o consumidor final. Alguns exemplos são produtos da Locaweb, Google, AdWords, SAP, AutoCAD e Microsoft Office.
- **Mistos:** nesse caso, o produto de software resolve tanto um problema par ao consumidor final quanto para as empresas.
- Normalmente esse tipo de produto de software não tem nenhum custo para o consumidor final e quem paga a conta são as empresas.
- O modelo mais comum de geração de receita desse tipo de produto de software é o anúncio pago pela empresa quando o consumidor final vê o anúncio, clica nele ou compra algo a partir dele



# ENGENHARIA DE SOFTWARE

## Produtos ou Plataforma?



- Atualmente vemos mais produtos de softwares serem classificados como plataformas.
- Exemplos não faltam, principalmente grandes empresas de tecnologia ou não:
- **Google:** criou uma plataforma conectando pessoas que buscam informações na internet com pessoas que querem anunciar coisas na internet.
- **Facebook:** que começou como uma plataforma em que amigos se encontravam e trocavam informações, e tornou- se uma plataforma na qual anunciantes podem falar com pessoas por meio de anúncios e páginas de empresas. **LinkedIn:** uma plataforma para profissionais, empresas e, mais recentemente, anunciantes.



# ENGENHARIA DE SOFTWARE

## Produtos ou Plataforma?

- **Apple:** que conecta desenvolvedores de software com usuários de sua AppStore.
- **iTunes:** conectando produtores de mídia com pessoas interessadas em música, filmes, séries e livros.
- **Amazon Kindle:** plataforma para que editoras ou autores possam publicar livros para pessoas interessadas nesses conteúdos.
- **Uber:** conectando motoristas com pessoas querendo transporte.
- **Airbnb:** conectando quem tem imóvel para alugar por períodos curtos com quem quer alugar um imóvel nessas condições.
- **Bitcoin:** quanto mais usuários tiver e quanto mais empresas aceitarem melhor.



# ENGENHARIA DE SOFTWARE

## Ferramentas CASE (Computer-Aided Software Engineering)



- Os engenheiros de software usam um conjunto de ferramentas, que podem trabalhar de maneira simples e pontual sob um aspecto particular para suportar várias atividades de uma vez.
- É importante ressaltar que ferramentas CASE estão disponíveis para várias áreas da engenharia de software e não apenas aos projetos de software.



# ENGENHARIA DE SOFTWARE

## Ferramentas CASE (Computer-Aided Software Engineering)



- Dessa forma, temos ferramentas CASE para sistemas de informação, gerenciamento de projetos, requisitos, análise e projeto, implementação, construção de protótipos e simulação, integração e testes, gerência de configuração, mapeamento de processos, processos de negócio, manutenção e framework.
- Geralmente, as ferramentas CASE funcionam bem para aquelas áreas mais maduras, na qual os processos possuem melhores definições. Em processos cujas atividades são mais variadas, ainda dependentes da organização, as ferramentas CASE não costumam funcionar muito bem.



# ENGENHARIA DE SOFTWARE

## Ferramentas CASE (Computer-Aided Software Engineering)



- Pressman (2006) preconiza ser difícil classificar as ferramentas CASE.
- A confusão é grande e, enquanto um determinado tipo de ferramenta pode ser categorizada de uma determinada forma, outros autores podem classificá-la de forma diferente, levando em consideração outros aspectos da ferramenta.
- Segundo ele, a classificação simplista tende a ser pobre e incompleta.



# ENGENHARIA DE SOFTWARE

## Ferramentas CASE (Computer-Aided Software Engineering)



- Mesmo assim, Pressman diz ser necessário classificar as ferramentas CASE e afirma ter vários assuntos para categorizá-las, como, por exemplo, função, papel para gerentes ou técnicos, utilização em vários passos do processo de engenharia de software, arquitetura, origem ou custo.

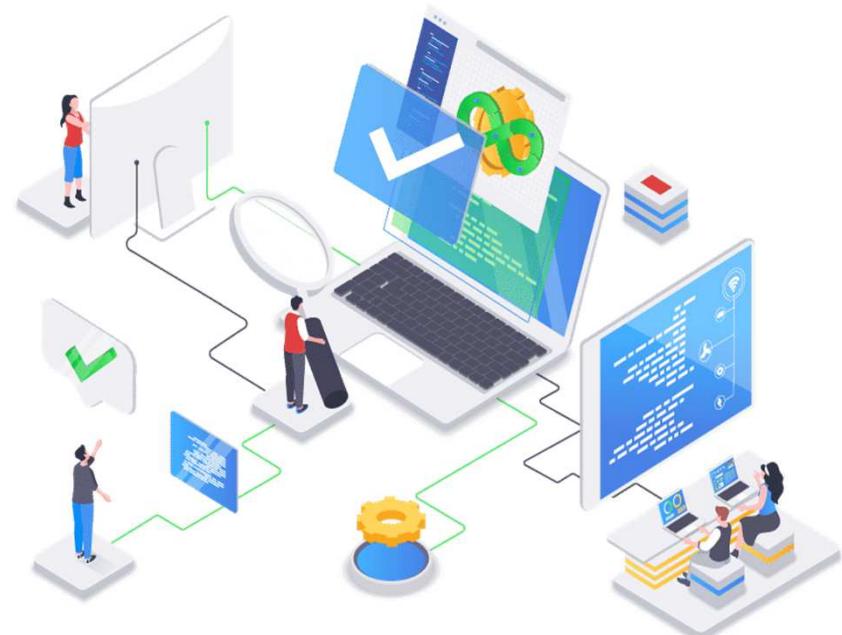
- **Ferramentas de PLANEJAMENTO DE SISTEMAS DE INFORMAÇÃO**

Ao modelar os requisitos estratégicos de informação de uma organização, as essas ferramentas proporcionam um “meta-modelo” a partir do qual sistemas de informações específicos são derivados.



# ENGENHARIA DE SOFTWARE

- Elas respondem a perguntas simples, mas importantes:
  - **Onde** se originam os dados críticos do negócio?
  - **Para onde** vão tais informações? Como elas são usadas?
  - **Como** elas são transformadas à medida que realizam as funções do negócio? Quais novas informações são adicionadas?
- As ferramentas de planejamento de sistemas de informação ajudam os desenvolvedores de software a criar sistemas de informações que encaminham dados àqueles que precisam de informação e que resistem a enganar os membros do pessoal com informações irrelevantes, melhorando a análise final e as tomadas de decisão são agilizadas.



# ENGENHARIA DE SOFTWARE



- **Ferramentas de GERENCIAMENTO DE PROJETOS**
- São ferramentas focadas em estimar o custo e o esforço a ser gasto no software, bem como a realização de cronogramas de projeto. Existem diversas ferramentas que tratam do planejamento, gestão e acompanhamento de projetos.
- A ferramenta proprietária mais conhecida é a MSProject da Microsoft. Entre as gratuitas, podemos citar aquelas que são utilizadas em desktop como GattProject, Open Workbench, Faces, TaskJuggler, OpenProj e outra categoria que são instaláveis em servidores web que rodam no browser Achievo, DotProject.



# ENGENHARIA DE SOFTWARE



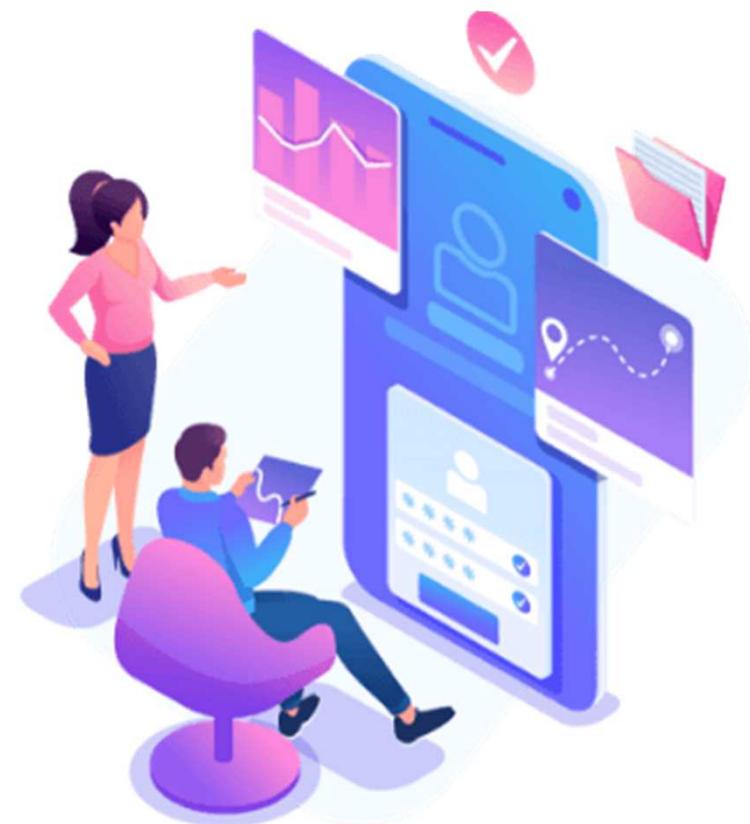
- Usando essas ferramentas de gerenciamento de projetos pode gerar estimativas úteis de um projeto de software, definindo uma estrutura de divisão de trabalho, baseline para a produtividade no desenvolvimento de software e qualidade do produto de software.
- **Ferramentas de ENGENHARIA DE PROCESSO DE NEGÓCIO**
- Ferramentas desse tipo modelam os requisitos de informação estratégica de uma determinada organização. Consistem em identificar objetos de dados de negócio, seus relacionamentos e como esses objetos fluem entre as diferentes áreas de negócio da organização.



# ENGENHARIA DE SOFTWARE



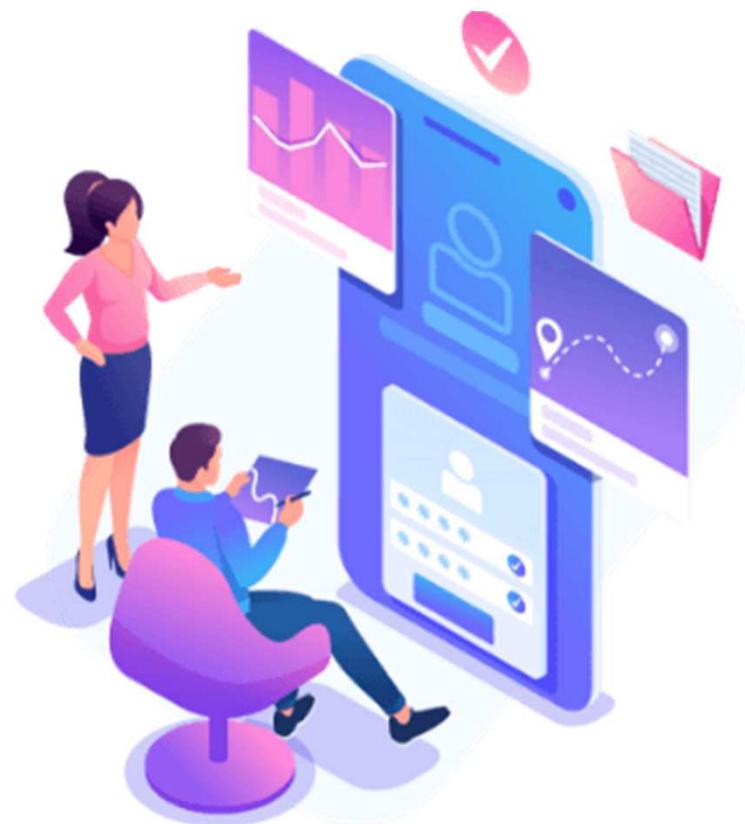
- Para processos de negócio, podemos adotar qualquer ferramenta de modelagem **BPMN – Business Process Modeling Notation**.
- A BPMN, assim como a UML (Unified Modeling Language), é uma conhecida notação gráfica, mas seu intuito é totalmente diferente. A BPMN consiste basicamente em modelar processos de negócio organizacionais.
- **Ferramentas de ENGENHARIA DE PROCESSO DE NEGÓCIO**
- Ferramentas desse tipo modelam os requisitos de informação estratégica de uma determinada organização. Consistem em identificar objetos de dados de negócio, seus relacionamentos e como esses objetos fluem entre as diferentes áreas de negócio da organização.



# ENGENHARIA DE SOFTWARE



- Para processos de negócio, podemos adotar qualquer ferramenta de modelagem **BPMN – Business Process Modeling Notation**.
- A BPMN, assim como a UML (Unified Modeling Language), é uma conhecida notação gráfica, mas seu intuito é totalmente diferente. A BPMN consiste basicamente em modelar processos de negócio organizacionais.
- Ferramentas de **GESTÃO DE PROCESSOS DE NEGÓCIOS (BPM)** Identificam os principais modelos de processos da organização. O objetivo é entender melhor os processos para depois melhorá-los.
- BPM é a abreviação de **Business Process Management**, que traduzido para o português significa Gerenciamento de Processos de Negócio.



# ENGENHARIA DE SOFTWARE

- Ferramentas de **RASTREAMENTO DE REQUISITOS** Com essas ferramentas, conseguimos montar um mecanismo para isolar os requisitos, desde a definição inicial do cliente. Uma das preferidas do mercado é a Rational RequisitePro.
- Ferramentas de **MÉTRICAS E GESTÃO** Essas ferramentas focam as peculiaridades do projeto, a fim de obter uma referência globalizada da produtividade ou da qualidade do projeto. Podemos citar como exemplo de ferramentas para métricas, COCOMO e APF entre outras.
- Ferramentas de **DOCUMENTAÇÃO** Focalizam a melhoria da produtividade ao permitirem a realização da documentação do software. Nesse caso, os mais comuns e conhecidos são os editores de texto e editoração eletrônica, como os existentes em pacotes de software para escritório.



# ENGENHARIA DE SOFTWARE



- Ferramentas de **SOFTWARE BÁSICO**: É a base tecnológica para o correto funcionamento do ambiente em que está localizada a ferramenta CASE. Representam softwares comuns, básicos, como os softwares de rede, correio eletrônico, etc.
- Ferramentas de **GARANTIA DA QUALIDADE**: Ferramentas de medição que realizam auditoria no código-fonte do projeto, para averiguação das normas da linguagem de programação.
- Ferramentas de **GESTÃO DE BASE DE DADOS**: Essas ferramentas podem ser utilizadas para a geração dos modelos que compõem um Projeto de Banco de Dados, sua documentação, a criação efetiva dos objetos do banco de dados, e suporte à linguagem de manipulação de dados em banco de dados.



# ENGENHARIA DE SOFTWARE



- Ferramentas de **GESTÃO DE CONFIGURAÇÃO DE SOFTWARE**

Ferramentas que ajudam a localizar e relacionar os itens de configuração do ambiente, além da identificação, do controle de versão, controle de modificação, auditoria e listagem de categorias. Podemos citar como exemplos de ferramentas gratuitas: o CVS, Subversion, SVN e o GitHub.

- Ferramentas de **ANÁLISE E PROJETO** Essas ferramentas são aquelas que dão suporte à criação de modelos conceituais e arquiteturais das classes e entidades do projeto.

- No contexto da orientação a objetos, temos as ferramentas de modelagem que utilizam a notação UML como, por exemplo, ArgoUML, Rational Rose, Power Designer e Enterprise Architect, entre inúmeras outras.



# ENGENHARIA DE SOFTWARE



- Ferramentas de **PROTOTIPAÇÃO/SIMULAÇÃO**: São ferramentas de prototipação e simulação. Elas têm a capacidade de prever o comportamento do software em tempo real, antes de sua construção, como por exemplo FIGMA, IN VISION entre outras
- Ferramentas de **PROJETO** e desenvolvimento de **INTERFACES**: Essas ferramentas contêm um grande número de componentes visuais que ajudam a construir uma interface de maneira fácil e rápida.
- Ferramentas de **PROGRAMAÇÃO (IMPLEMENTAÇÃO)**: São ferramentas disponíveis no mercado para a escrita do código-fonte. Possuem editores de texto, compiladores, depuradores, entre outras ferramentas para melhorar a produtividade dos desenvolvedores.



# ENGENHARIA DE SOFTWARE



- Existem diversas ferramentas CASE para programação. Podemos destacar: o Eclipse, NetBeans e o próprio Visual Studio .NET, entre várias outras IDEs (Integrated Development Environment)) ou Ambiente de Desenvolvimento Integrado, sendo ferramentas usada no desenvolvimento de aplicações, que combina diferentes funcionalidades em uma única interface gráfica.
- Ferramentas de **DESENVOLVIMENTO WEB**: São ferramentas que possibilitam a construção de texto, gráficos, formulários e demais elementos disponíveis em uma página web.
- Android Studio Desenvolvimento WEB abrange uma grande gama de ferramentas, se nos concentrarmos única e exclusivamente nas interfaces WEB e MOBILE.



# ENGENHARIA DE SOFTWARE



- Ferramentas de **INTEGRAÇÃO** e **TESTE**: Ferramentas responsáveis por colher dados, realizar medições estáticas e/ou dinâmicas do código-fonte, simulações de dispositivos, gestão e cruzamento de fronteiras, para atingir o objetivo de testar e automatizar os sistemas. Selenium, Cucumber, Appium
- As ferramentas de **INTEGRAÇÃO, ENTREGA E IMPLEMENTAÇÃO CONTÍNUA (CI/CD - DEVOPS)** permite que seja configurado o seu sistema, o ambiente de desenvolvimento (exemplo: jdk), permite que seja configurado o sistema de build automatizado, integrar com o repositório de controle de versão, e permite ainda o envio de e-mails de notificação e monitoramento das aplicações até a gestão de mudanças de software.



# ENGENHARIA DE SOFTWARE

## Projeto de Software



- Normalmente, o desenvolvimento de um novo software é feito dentro de um projeto.
- Todo projeto tem uma data de início, uma data de fim, uma equipe (da qual faz parte um responsável, que chamaremos de gerente do projeto) e outros recursos. Um projeto representa a execução de um processo de software.
- Quando um processo é bem definido, ele tem subdivisões que permitam avaliar o progresso de um projeto, e corrigir seus rumos quando acontecerem problemas. Estas subdivisões são chamadas de fases, atividades ou iterações.



# ENGENHARIA DE SOFTWARE

## Projeto de Software



- As subdivisões devem ser terminadas por marcos, isto é, pontos que representam estados significativos do projeto.
- Geralmente os marcos são associados a resultados concretos: documentos, modelos ou porções do produto, que podem fazer parte do conjunto prometido aos clientes, ou ter apenas utilização interna ao projeto.
- O próprio produto é um resultado associado ao marco de conclusão do projeto.



# ENGENHARIA DE SOFTWARE

## Projeto de Software



- Por que tantos sistemas informatizados são entregues com atraso e custam mais do que o previsto? Estourar cronogramas e orçamentos é parte da rotina da maioria dos profissionais de software.
- Clientes e gerentes se desesperam com os atrasos dos projetos de software, e às vezes sofrem enormes prejuízos com eles. **Entretanto, no próximo contrato, eles provavelmente escolherão que prometer menor prazo e/ou menor custo.**
- Se for um projeto interno da organização, farão todo tipo de pressões para conseguir que os desenvolvedores prometam prazos politicamente agradáveis, embora irreais.



# ENGENHARIA DE SOFTWARE

## Projeto de Software



- Estimar prazos e custos faz parte da rotina de qualquer ramo da engenharia.
- Para um produto ser viável, não basta que atenda aos requisitos desejados; tem de ser produzido dentro de certos parâmetros de prazo e custo.
- Se isto não for possível, o produto pode não ser viável do ponto de vista de mercado, ou pode ser preferível adquirir outro produto, ainda que sacrificando alguns dos requisitos. Ter estimativas de prazos e custos, portanto, é uma expectativa mais que razoável de clientes e gerentes.



# ENGENHARIA DE SOFTWARE

- O problema é que existem alguns desenvolvedores pouco escrupulosos. E existem muitos que, mesmo sendo honestos, não conhecem métodos técnicos de estimativa de prazos e custos do desenvolvimento de software.
- E existem ainda os que, mesmo sabendo fazer boas estimativas, trabalham em organizações onde não existe clima para que os desenvolvedores possam apresentar avaliações francas das perspectivas dos projetos.
- Nestas organizações, existe a política de "matar os mensageiros de más notícias". Esta política foi usada por muitos reis da antiguidade, com resultados geralmente desastrosos.



# ENGENHARIA DE SOFTWARE



- Requisitos, prazos e custos formam os vértices de um triângulo crítico da Engenharia de Software.
- Aumentos de requisitos levam a aumentos de prazos ou custos, ou ambos. Reduções de requisitos podem levar a reduções de prazos ou custos (mas nem sempre).
- Uma coisa é exigir dos engenheiros de software estimativas de prazos, e cobrar o cumprimento dos prazos prometidos.
- Clientes e gerentes podem e devem fazê-lo. Outra coisa é pressioná-los para que façam promessas que não podem ser cumpridas



*Um triângulo crítico da Engenharia de Software*

# ENGENHARIA DE SOFTWARE



- Uma frase comum desta cultura é: "Não me interessa como você vai fazer, desde que entregue no prazo!".
- Na realidade, o cliente ou gerente deve, no seu próprio interesse, ter algum meio de checar se o cronograma e orçamento propostos são realistas; se preciso, recorrendo aos serviços uma terceira parte.
- A cultura do prazo político é ruim para todos. Para os desenvolvedores, ela significa estresse e má qualidade de vida.
- Para os gerentes, perda de credibilidade e prejuízos. E para os clientes, produtos de má qualidade e mais caros do que deveriam. Ainda por cima, entregues fora do prazo.



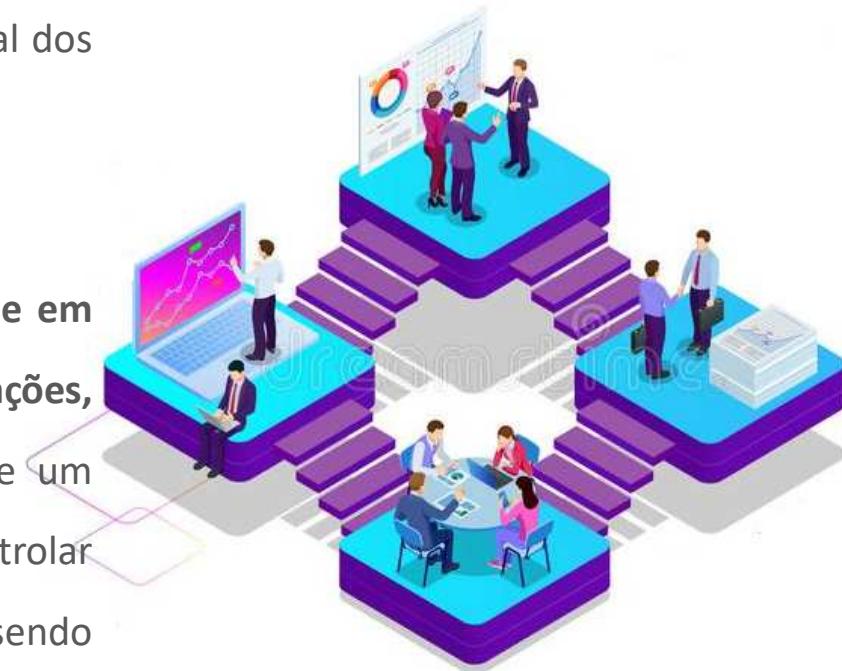
# ENGENHARIA DE SOFTWARE

- Para cumprir compromissos de prazos e custos, estes compromissos têm de ser assumidos com base em requisitos bem levantados, analisados e documentados. E os planos dos projetos têm de ser feitos com boas técnicas de estimativa e análise de tamanho, esforços, prazos e riscos.
- **Estas técnicas pertencem à disciplina de planejamento de projetos, que faz parte da Engenharia de Software. A disciplina complementar do planejamento de projetos é o controle dos projetos. Ele compreende:**
  - O acompanhamento do progresso dos projetos, comparando-se o planejado com o realizado;
  - A busca de alternativas para contornar problemas surgidos na execução dos projetos;
  - O replanejamento dos projetos, quando não é possível manter os planos anteriores dentro de um grau razoável de variação;
  - A renegociação dos compromissos assumidos, envolvendo todas as partes interessadas.

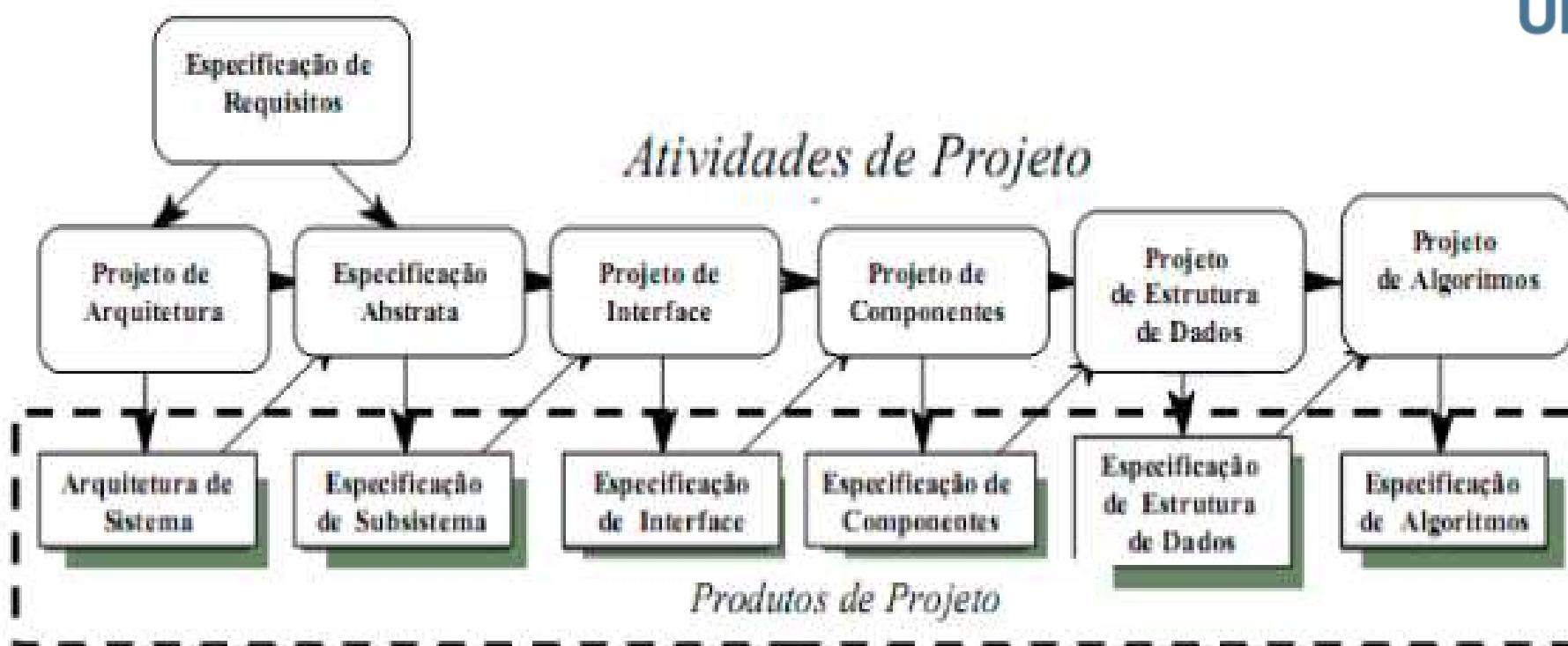


# ENGENHARIA DE SOFTWARE

- Todo plano comporta incertezas. Por exemplo, o tamanho de certas partes do produto pode ser estimado grosseiramente a partir dos requisitos, mas o desenho detalhado das partes do produto permite refinar as estimativas, e o tamanho correto só é exatamente conhecido no final dos projetos. E riscos previstos e não previstos podem se materializar.
- **A produtividade dos desenvolvedores pode ser estimada com base em projetos anteriores da organização, mas é afetada por muitas variações, que dependem de pessoas, processos e tecnologia.** Ao longo de um projeto, os gerentes têm de enfrentar problemas e tentar controlar variáveis que afetem o cumprimento de seus compromissos, sendo necessário replanejar atualizando as estimativas.



# ENGENHARIA DE SOFTWARE



**Figura 7: Projeto de software.**

# ENGENHARIA DE SOFTWARE

## Requisitos de Software



- Requisitos são as bases para todo projeto de software, definindo o que as partes interessadas de um novo sistema necessitam e o que o sistema deve fazer para satisfazer as suas necessidades. São tão importantes que normalmente eles possuem um grande impacto nas falhas dos projetos de software.
- O Standish Group, por meio de seu *Chaos Report de 2000*, informa que 74% dos projetos de software desenvolvidos falharam. Os motivos em grande parte com objetivos não esclarecidos, ausência de planejamento, requisitos e especificações incompletos, ou falta de controle na mudança destes.



<https://www.standishgroup.com/about>

# ENGENHARIA DE SOFTWARE

## Requisitos de Software



- O Standish Group ainda fez uma análise sobre os fatores críticos para sucesso dos projetos de software.
- Podemos perceber que três dos principais fatores estão relacionados às atividades de requisitos:
  - (1) Requisitos Incompletos
  - (2) Falta de Envolvimento do Usuário
  - (3) Mudança de Requisitos e Especificações



# ENGENHARIA DE SOFTWARE

## Requisitos de Software



- Um requisito é uma característica do sistema ou a descrição de algo que o sistema é capaz de realizar para atingir os seus objetivos.
- Um requisito é uma propriedade que o software deve exibir para resolver algum problema no mundo real.
- Uma condição ou uma capacidade que deve ser alcançada ou estar presente em um sistema para satisfazer um contrato, padrão, especificação ou outro documento formalmente imposto.



# ENGENHARIA DE SOFTWARE

## Requisitos de Software



- Podemos entender requisitos como sendo o conjunto de necessidades explicitadas pelo cliente que deverão ser atendidas para solucionar um determinado problema do negócio no qual o cliente faz parte.
- Embora o requisito seja definido pelo cliente, nem sempre o que o cliente quer é o que o negócio precisa, sendo necessário identificar a real necessidade do negócio.
- O entendimento dos requisitos de um problema está entre as tarefas mais difíceis enfrentadas pelos profissionais de desenvolvimento de sistemas.



# ENGENHARIA DE SOFTWARE

## Requisitos de Software



Os requisitos são importantes para:

- Estabelecer uma base de concordância entre o cliente e o fornecedor sobre o que o software fará.
- Fornecer uma referência para a validação do produto final.
- Reduzir o custo de desenvolvimento, ou seja, requisitos mal definidos causam retrabalho.



# ENGENHARIA DE SOFTWARE

## Requisitos Funcionais

- São diretamente ligados a funcionalidade do software, descrevem as funções que o software deve executar, ou seja, representa o que o software faz, em termos de tarefas e serviços.
- Os requisitos funcionais podem ser cálculos, detalhes técnicos, manipulação de dados e de processamento e outras funcionalidades específicas que definem o que um sistema, idealmente, será capaz de realizar.
- Exemplos:
  - O software deve permitir o cadastro de clientes;
  - O software deve permitir a geração de relatórios sobre o desempenho de vendas no semestre;
  - O software deve permitir o pagamento das compras através de cartão de crédito



# ENGENHARIA DE SOFTWARE

## Requisitos Não Funcionais



- Expressam condições que o software deve atender ou qualidades específicas que o software deve ter. Em vez de informar o que o sistema fará, os requisitos não-funcionais colocam restrições no sistema.
- São os requisitos relacionados ao uso da aplicação em termos de desempenho, usabilidade, confiabilidade, segurança, disponibilidade, manutenção e tecnologias envolvidas.
- Exemplos:
  - O software deve ser compatível com os browsers IE (versão 5.0 ou superior) e Firefox (1.0 ou superior)
  - O software deve garantir que o tempo de retorno das consultas não seja maior do que 5 segundos;



# ENGENHARIA DE SOFTWARE

## Requisitos de Domínio



- São requisitos derivados do domínio da aplicação e descrevem características do sistema e qualidades que refletem o domínio (podem ser requisitos funcionais e não funcionais).
- Podem ser requisitos funcionais novos, restrições sobre requisitos existentes ou computações específicas.
- Exemplos:
  - O cálculo da média final de cada aluno é dado pela fórmula:  $(Nota1 * 2 + Nota2 * 3)/5$
  - Um aluno pode se matricular em uma disciplina desde que ele tenha sido aprovado nas disciplinas consideradas pré-requisitos;



# ENGENHARIA DE SOFTWARE

## Regras de Negócio

- As regras de negócio são restrições/premissas necessárias para o negócio “acontecer”. Diferença de “requisito funcional” e “regra de negócio” é algo comum na cabeça de vários analistas de sistemas.

Poderíamos elencar dezenas de regras de negócio, mas é fundamental que seja claro que regras de negócio existem sem sistema, e que uma empresa não existe sem regras de negócio.

- A diferença entre requisito funcional e regra de negócio, conceitualmente falando, é que o requisito funcional se refere a “**o que o sistema deverá fazer**”, enquanto a regra de negócio refere-se a “**como o sistema deverá fazer**”. Do ponto de vista do negócio ambos são necessidades mas cada uma com um foco diferente.



# ENGENHARIA DE SOFTWARE

## Engenharia de Requisitos



- Portanto, a engenharia de requisitos é o processo pelo qual os requisitos de um produto de software são coletados, analisados, documentados e gerenciados ao longo de todo o ciclo de vida do software.
- **Termo usado para descrever as atividades relacionadas à investigação e definição de escopo de um sistema de software.**
- Processo sistemático de desenvolvimento de requisitos através de um processo cooperativo de análise onde os resultados das observações são codificados em uma variedade de formatos e a acurácia das observações é constantemente verificada.



# ENGENHARIA DE SOFTWARE

## Engenharia de Requisitos



- Processo de descobrir, analisar, documentar e verificar as funções e restrições do sistema, registrar e acompanhar requisitos ao longo de todo o processo de desenvolvimento.
- Estabelecer uma visão comum entre o cliente e a equipe de projeto em relação aos requisitos que serão atendidos pelo projeto de software.
- Documentar e controlar os requisitos alocados para estabelecer uma baseline para uso gerencial e da engenharia de software, manter planos, artefatos e atividades de software consistentes com os requisitos alocados.



# ENGENHARIA DE SOFTWARE

## Engenharia de Requisitos

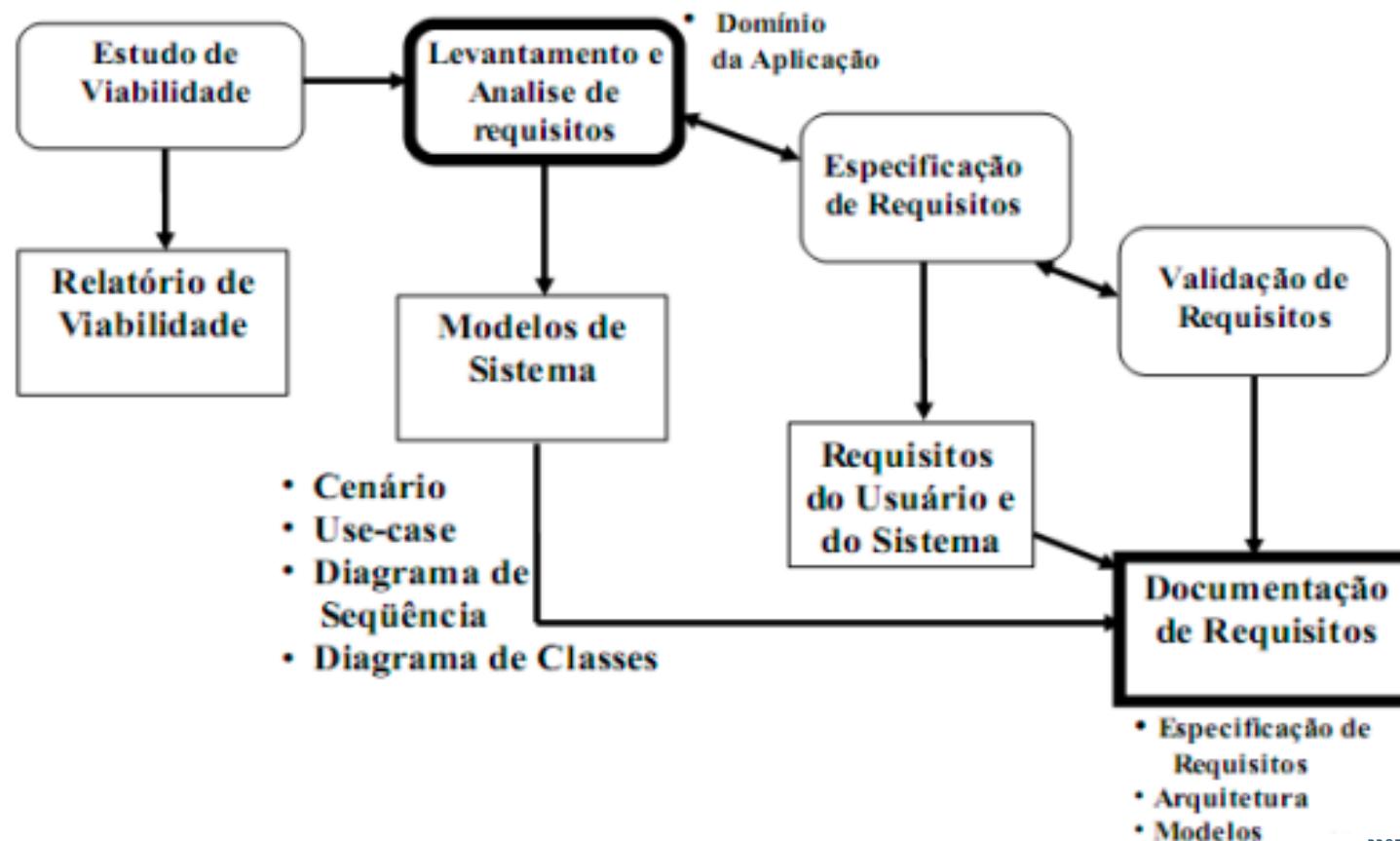


- De uma forma geral os requisitos possuem um papel fundamental para o desenvolvimento de software.
- São uma das principais medidas de sucesso de um software, visto que se eles atendem aos objetivos para os quais foi construído o software e se está totalmente de acordo com as necessidades dos clientes.
- Requisitos são a base para estimativas, modelagem, projeto, implementação, testes e até mesmo para a manutenção. Assim, os requisitos estão presentes ao longo de todo o ciclo de vida de um software



# ENGENHARIA DE SOFTWARE

## Processo de Engenharia de Requisitos



# ENGENHARIA DE SOFTWARE

## Processo de Engenharia de Requisitos



Fonte: <https://www.devmedia.com.br/introducao-a-engenharia-de-requisitos/8034>

# ENGENHARIA DE SOFTWARE

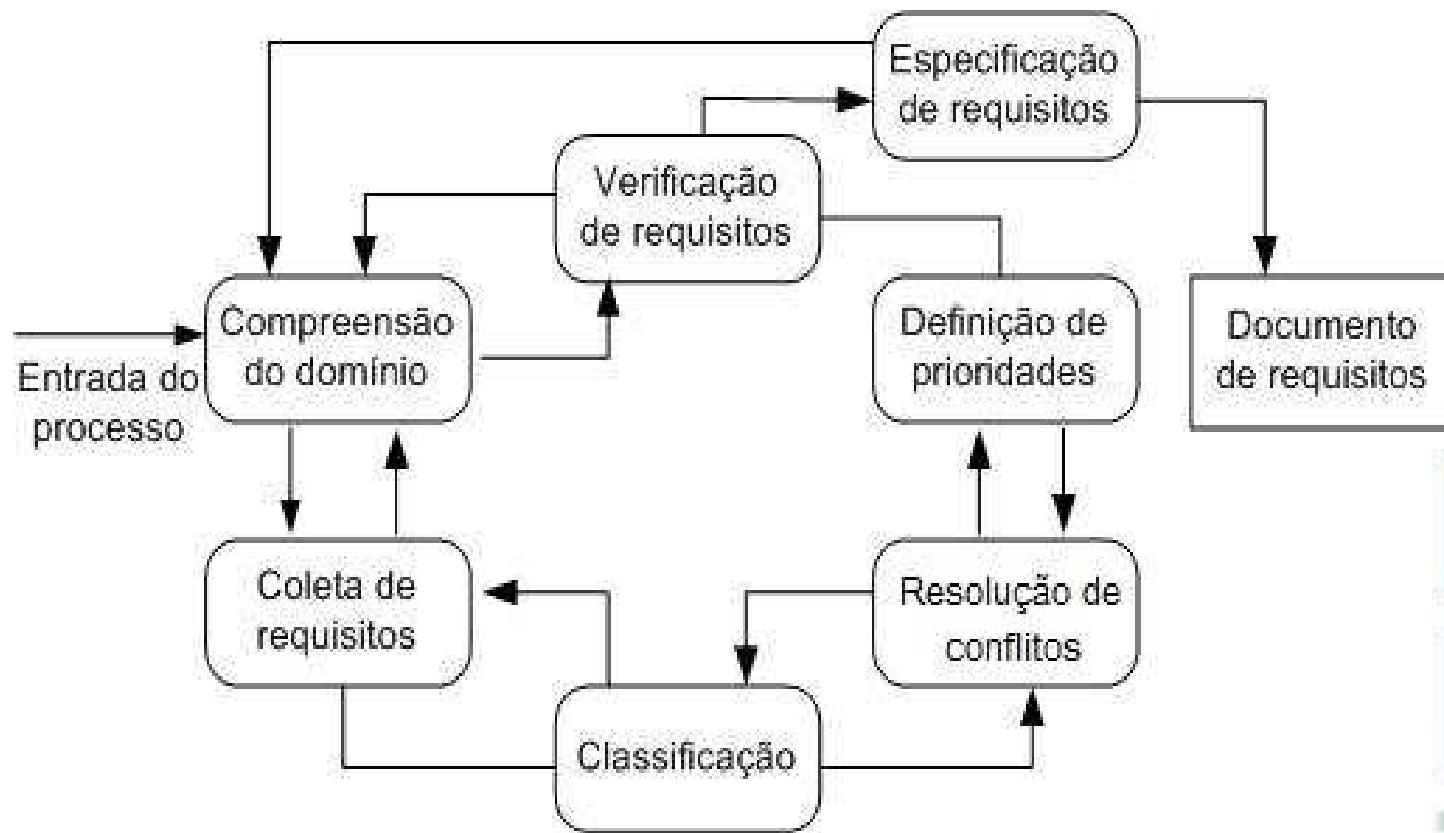
## Engenharia de Requisitos



- Sommerville (2003) propõe um processo genérico de levantamento e análise que contém as seguintes atividades:
  - **Compreensão do domínio:** Os analistas devem desenvolver sua compreensão do domínio da aplicação;
  - **Coleta de requisitos:** É o processo de interagir com os stakeholders do sistema para descobrir seus requisitos. A compreensão do domínio se desenvolve mais durante essa atividade;
  - **Classificação:** Essa atividade considera o conjunto não estruturado dos requisitos e os organiza em grupos coerentes;
  - **Resolução de conflitos:** Quando múltiplos stakeholders estão envolvidos, os requisitos apresentarão conflitos. Essa atividade tem por objetivo solucionar esses conflitos;
  - **Definição das prioridades:** Em qualquer conjunto de requisitos, alguns serão mais importantes do que outros. Esse estágio envolve interação com os stakeholders para a definição dos requisitos mais importantes;
  - **Verificação de requisitos:** Os requisitos são verificados para descobrir se estão completos e consistentes e se estão em concordância com o que os stakeholders desejam do sistema.



# ENGENHARIA DE SOFTWARE



Processo de levantamento e análise de requisitos (SOMMERVILLE, 2003).

Fonte: <https://educapes.capes.gov.br/bitstream/capes/177122/2/Material%20Didatico-Engenharia%20de%20Software.pdf>

# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



- É um fruto da constatação feita, de forma independente, por 17 profissionais renomados na área de engenharia de software, de que, apesar de terem aprendido segundo a cartilha tradicional, só conseguiam minimizar os riscos associados ao desenvolvimento de software, pensando e agindo de forma muito diferente do que tradicionalmente está nos livros.
- Estes profissionais, a maioria veteranos consultores para projetos de softwares, decidiram reunir-se no início de 2001 durante um workshop realizado em Snowbird, Utah, EUA.



# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



- Embora cada envolvido tivesse suas próprias práticas e teorias preferidas, todos concordavam que, em suas experiências prévias, os projetos de sucesso tinham em comum um pequeno conjunto de princípios.
- Com base nisso eles criaram o **Manifesto para o Desenvolvimento Ágil de Software**, ou apenas de Manifesto Ágil.
- O termo Desenvolvimento Ágil identifica metodologias de desenvolvimento que adotam os valores e princípios do manifesto ágil.



# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



- Os 4 valores são:
  - Indivíduos e interação entre eles mais que processos e ferramentas;
  - Software em funcionamento mais que documentação abrangente;
  - Colaboração com o cliente mais que negociação de contratos;
  - Responder a mudanças mais que seguir um plano.
- O manifesto reconhece o valor dos itens à direita, mas diz que devemos valorizar bem mais os itens à esquerda.



# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



- Indivíduos e interação entre eles mais que processos e ferramentas
  - Devemos entender que o desenvolvimento de software é uma atividade humana e que a qualidade da interação entre as pessoas pode resolver problemas crônicos de comunicação.
  - Processos e ferramentas são importantes, mas devem ser simples e úteis.
- Software em funcionamento mais que documentação abrangente
  - O maior indicador de que sua equipe realmente construiu algo é software funcionando. Clientes querem é resultado e isso pode ser com software funcionando.
  - Documentação também é importante, mas que seja somente o necessário e que agregue valor.



# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



- Colaboração com o cliente mais que negociação de contratos

- Devemos atuar em conjunto com o cliente e não “contra” ele ou ele “contra” a gente.
- O que deve acontecer é colaboração, tomada de decisões em conjunto e trabalho em equipe, fazendo que todos sejam um só em busca de um objetivo.

- Responder a mudanças mais que seguir um plano.

- Desenvolver software e produtos é um ambiente de alta incerteza e por isso não podemos nos debruçar em planos enormes e cheio de premissas.
- O que deve ser feito é aprender com as informações e feedbacks e adaptar o plano a todo momento.



# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



- Os 12 princípios são:

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

2) Mudanças nos requisitos são bem vindas, mesmo em estágios tardios do desenvolvimento. Processos ágeis devem admitir mudanças que trazem vantagens competitivas para o cliente.

3) Entregar software funcionando com frequência, na escala de semanas até meses, de preferência na menor escala de tempo possível.

4) Conhecedores do negócio e desenvolvedores devem trabalhar juntos diariamente durante o projeto.



# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



- Os 12 princípios são:

5) Construir projetos com indivíduos motivados. Dê a eles o espaço de trabalho e o apoio de que necessitam, e confie neles para ter o trabalho pronto.

6) O método mais eficiente e eficaz de transmitir informações na equipe é a conversa cara-a-cara.

7) Software funcionando é a principal medida de progresso.

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



# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



- Os 12 princípios são:

9) Atenção contínua à excelência técnica e bom design aumentam a agilidade.

10) Simplicidade – a arte de maximizar a quantidade de trabalho não feito – é essencial.

11) As melhores arquiteturas, requisitos e design emergem de equipes auto-organizadas.

12) Em intervalos regulares, a equipe reflete sobre como se tornar mais eficaz, depois aprimora e ajusta seu comportamento em conformidade.



# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



- No desenvolvimento ágil, os projetos adotam o modelo iterativo e em espiral.
- Neste processo, todas as fases descritas no modelo em cascata são executadas diversas vezes ao longo do projeto, produzindo ciclos curtos que se repetem ao longo de todo o desenvolvimento, sendo que, ao final de cada ciclo, sempre se tem um software funcional, testado e aprovado.
- Os ciclos são chamados de iterações e crescem em número de funcionalidades a cada repetição, sendo que, no último ciclo, todas as funcionalidades desejadas estarão implementadas, testadas e aprovadas.
- O cliente aprende ao longo do desenvolvimento, à medida que é capaz de manipular o sistema.



# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



- Um dos problemas mais complexos que afetam o desenvolvimento de software é a enorme quantidade de detalhes que precisam ser considerados.
- Normalmente, ao especificar um sistema, o cliente tem o conhecimento de alguns aspectos do software que deseja. Entretanto, muitos outros só ficam claros quando ele tem a oportunidade de utilizar o sistema.
- Portanto, o cliente não especifica estes detalhes no início do projeto por uma razão muito simples: ele não os conhece.
- Além do mais, mesmo que tais detalhes fossem conhecidos previamente, seria muito difícil especificá-los através de documentos, devido à grande quantidade de elementos que precisariam ser descritos.



# **ENGENHARIA DE SOFTWARE**

## **Desenvolvimento Ágil de Software**



- Perceber que o cliente aprende ao longo do desenvolvimento é a chave para se compreender o grande desafio existente no desenvolvimento de software.
  - O cliente aprende à medida que tem acesso ao sistema e se envolve em discussões sobre ele. Este aprendizado pode ser utilizado para realimentar o processo de desenvolvimento ou pode ser ignorado.
  - Esta última alternativa é o caminho adotado normalmente no desenvolvimento tradicional, pois ele parte da premissa que o cliente já sabe o que quer no início do projeto e a ele cabe apenas descrever os requisitos.
  - Depois, ao final, ele receberá o sistema e não haverá ciclos de realimentação entre a especificação e a entrega do sistema.



# **ENGENHARIA DE SOFTWARE**

## **Desenvolvimento Ágil de Software**



- Ao contrário do desenvolvimento tradicional que acredita que o custo de uma mudança cresce exponencialmente a medida que o tempo de desenvolvimento avança, no desenvolvimento ágil acredita-se que o custo de mudança do software ao longo do tempo tende a se tornar constante.
  - **Se fundamenta nos seguintes fatores:**
    - Avanços ocorridos nas tecnologias;
    - Adoção da orientação a objetos;
    - Uso da refatoração para aprimorar e simplificar o design;
    - Adoção de testes automatizados;
    - Melhores linguagens e ambientes de desenvolvimento.



# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



- No desenvolvimento ágil, modificações no projeto são naturais, pois não existe nenhum custo exponencial associado às alterações, ou seja, não existe nenhuma necessidade de tentar especificar detalhadamente tudo que ocorrerá durante a implementação do sistema, para tentar minimizar possíveis alterações, até porque, isto dificilmente traz os resultados esperados.
- No desenvolvimento ágil os envolvidos no desenvolvimento são tratados como trabalhadores do conhecimento e consequentemente são estimulados a aprender durante o desenrolar do próprio trabalho e tomar decisões melhores com base neste aprendizado.
- Não existe um processo rígido que impõe o que pode ou não ser feito e desestimula a criatividade.



# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



- No desenvolvimento tradicional, os especialistas do negócio conversam com os analistas, que digerem, abstraem e passam as informações mastigadas aos programadores, que não pensam nada além do necessário para escrever o código do software.
- Esta especialização de atividades é péssima porque é completamente desprovida de feedback. O analista tem toda a responsabilidade de criar o modelo de domínio, baseado somente nas informações fornecidas pelos especialistas do negócio.
- Eles não têm a oportunidade de aprender com o desenvolvimento ou ganhar experiência com as versões iniciais do software.



# ENGENHARIA DE SOFTWARE



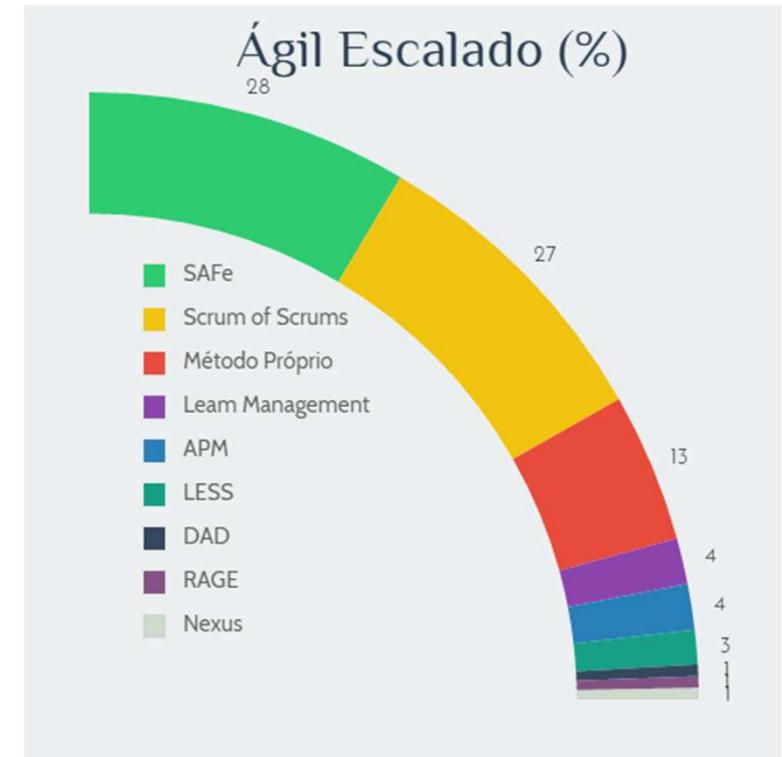
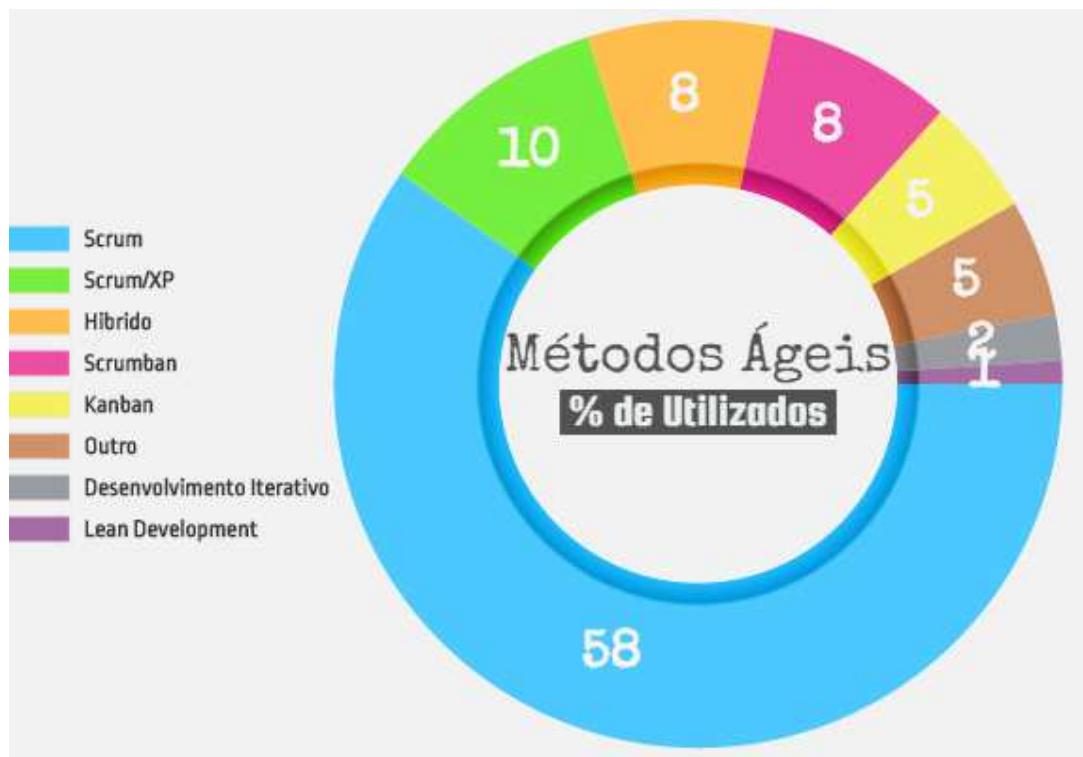
## Desenvolvimento Ágil de Software

- O desenvolvimento ágil sabe que as pessoas que tem o conhecimento rendem melhor em ambientes que estimulam o uso intensivo da criatividade e do conhecimento.
- Escrever softwares é um trabalho tão ou mais criativo que escrever um livro, um artigo ou uma monografia.
- É uma atividade tipicamente intelectual, onde é muito comum ocorrerem idas e vindas, já que o aprendizado adquirido com o seu desenrolar torna possível aos envolvidos perceberem maneiras cada vez melhores de fazerem as coisas.
- Este tipo de trabalho não linear funciona muito melhor de forma iterativa e incremental (em espiral).



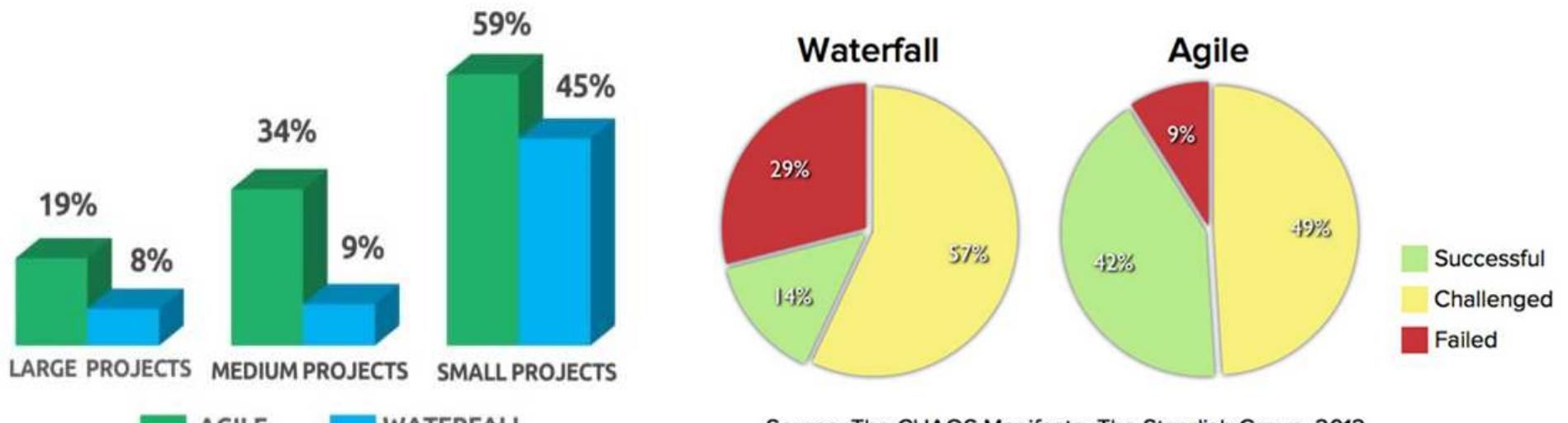
# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



Source: The CHAOS Manifesto, The Standish Group, 2012.

<https://standishgroup.myshopify.com/>

# ENGENHARIA DE SOFTWARE

## Desenvolvimento Ágil de Software



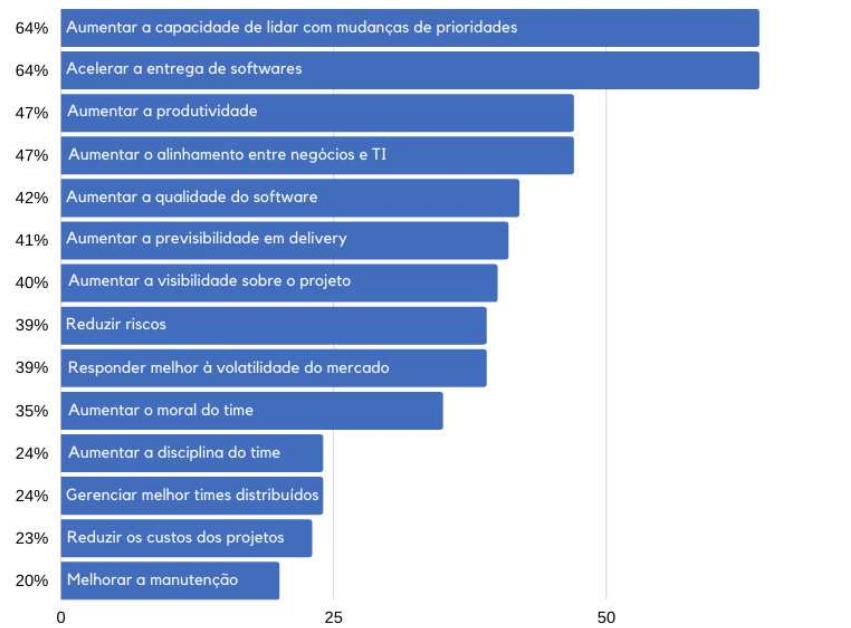
### PRINCIPAIS DESAFIOS NA ADOÇÃO DE METODOLOGIAS ÁGEIS

Fonte: 15th State of Agile Report



### PRINCIPAIS MOTIVOS PARA ADOTAR O AGILE NA ORGANIZAÇÃO

Fonte: 15th State of Agile Report



<https://digital.ai/resource-center/analyst-reports/state-of-agile-report#>

PROFº-- ROMMEL GABRIEL GONÇALVES RAMOS

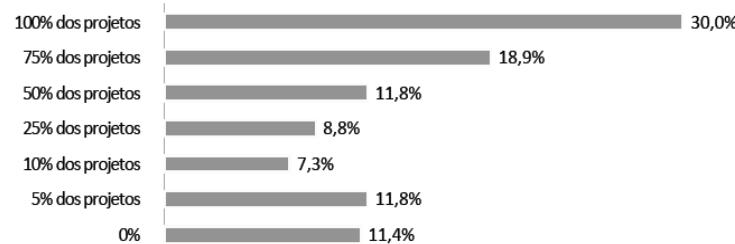
0104

# ENGENHARIA DE SOFTWARE

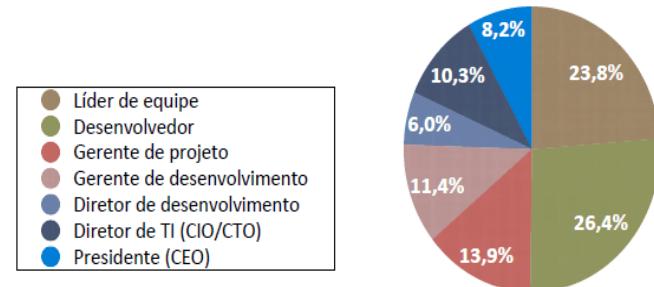


## Desenvolvimento Ágil de Software

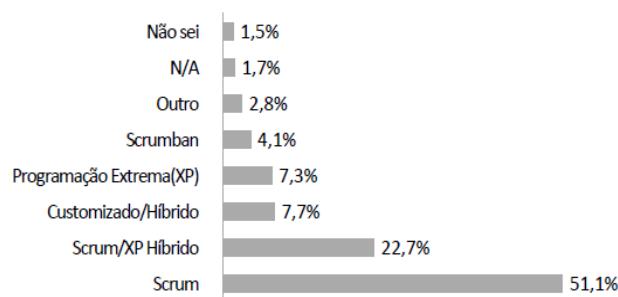
PERCENTUAL DE PROJETOS DA ORGANIZAÇÃO QUE ADOTAM MÉTODOS ÁGEIS



PRINCIPAL DEFENSOR INICIAL DE MÉTODOS ÁGEIS NA ORGANIZAÇÃO



QUAL O MÉTODO ÁGIL QUE VOCÊ MAIS SSEGUE

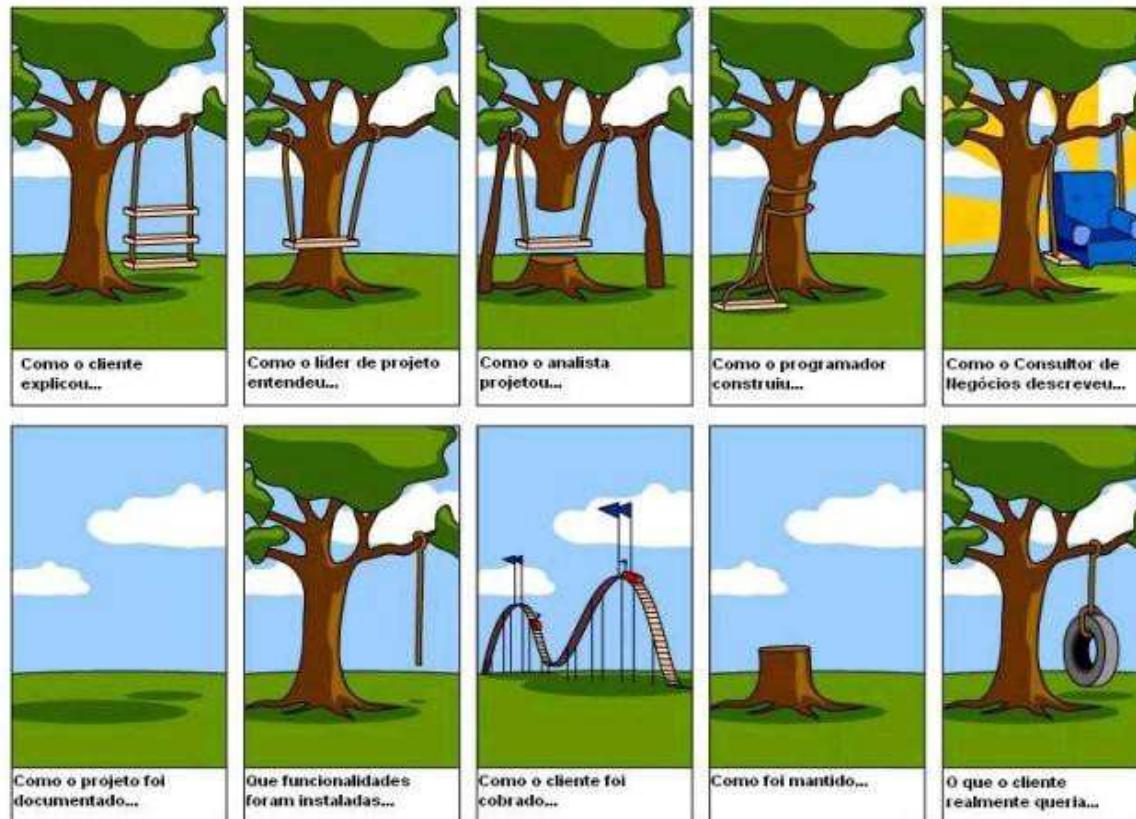


RAZÃO MAIS IMPORTANTE PARA A ADOÇÃO DE MÉTODOS ÁGEIS NA EQUIPE/ORGANIZAÇÃO



# ENGENHARIA DE SOFTWARE

## Especificação de Software



# ENGENHARIA DE SOFTWARE

## Especificação de Software



- Quanto mais a aplicação vai crescendo mais difícil fica de sabermos que pontos do sistema impactam em quais requisitos e operações do sistema. Dessa forma é sempre importante termos uma rastreabilidade sobre quais impactos temos entre sistema e requisitos.
- Assim, quando precisamos saber quem solicitou determinada funcionalidade podemos saber de imediato o nome do cliente. Isso muitas vezes é importante até mesmo quando precisamos solicitar maiores detalhes para o cliente ou então fazer alguma negociação.
- Para que a especificação de requisitos de software seja documentada de maneira correta, é preciso ficar atento à classificação dos mesmos. Isso é importante para não perder o foco durante o processo.



# ENGENHARIA DE SOFTWARE

## Especificação de Software



- As falhas em requisitos estão entre as principais razões para o fracasso de um software. Entre as principais razões destacam-se os requisitos mal organizados, requisitos mal expressos, requisitos desnecessários para os clientes e a dificuldade para lidar com requisitos frequentemente mutáveis.
- Falta de conhecimento sobre o domínio levando a entendimentos errados ou mal compreendidos. Além disso, quando temos poucos ou nenhum modelo e anotações sofremos com esquecimentos no futuro, inclusive com alguns conceitos iniciais ou mais básicos que quando não tomamos nota ou modelamos temos mais chances de esquecer.
- Quando temos diferentes pontos de vista entre o Usuário e o Analista podemos ter requisitos mal feitos ou falhos. O interessante nesse caso é fazermos um vocabulário da aplicação anotando conceitos importantes e procurando fazer um vocabulário comum junto ao cliente.



# ENGENHARIA DE SOFTWARE

## Especificação de Software



- É a etapa do desenvolvimento que determina o sucesso de um software. É o que define os objetivos e funções que um software precisa executar, bem como as que ele não pode ter (restrições).
- A etapa de especificação de requisitos de software exige a descrição do passo a passo do que irá ocorrer a cada ação do usuário. Dessa forma, o desenvolvimento será mais assertivo e o resultado estará alinhado com as expectativas do cliente.
- É possível que o orçamento tenha que ser revisto e o prazo de entrega fique prejudicado. Sendo assim, quanto mais coesa e clara for a especificação de requisitos de software, melhor para o sucesso desse sistema.
- Geralmente acontece com a elaboração de um documento pelo analista de sistemas. O profissional verifica junto ao cliente e demais interessados quais são as funções necessárias no software.



# ENGENHARIA DE SOFTWARE

## Especificação de Software



- É altamente recomendado que exista algum registro dessa aprovação pelo cliente, por exemplo:
  - Documento de especificação dos requisitos de software assinado pelo cliente aprovando seus requisitos;
  - E-mail do cliente contendo um “de acordo” sobre os requisitos aprovados;
  - Aprovações informais, sem registros, podem gerar conflitos futuros sobre o que teria sido aprovado na época e o conteúdo dos requisitos.
- Embora não exista um modelo padrão consagrado para gerenciar requisitos podemos definir alguns passos para um processo de especificação de requisitos software:
  - Descoberta dos requisitos (consultas, documentos, pesquisas, entrevistas);
  - Análise dos requisitos identificados com o refinamento e detalhamento dos mesmos;
  - Modelagem, verificação e validação dos requisitos verificando sua consistência.



# ENGENHARIA DE SOFTWARE

## Modelagem de Software

- É uma das principais atividades que levam à implementação de um bom software.
- Construímos modelos para comunicar a estrutura e o comportamento desejados do sistema, visualizar e controlar a arquitetura do mesmo e compreender melhor o sistema que estamos elaborando.
- Utiliza vários modelos para projetar um determinado sistema.
- Um modelo é uma simplificação da realidade, criado para facilitar o entendimento de sistemas complexos.
- Estes modelos podem abranger planos detalhados, assim como planos mais gerais com uma visão panorâmica do sistema.



# ENGENHARIA DE SOFTWARE

## Modelagem de Software



- O objetivo da modelagem de sistemas é facilitar todo o desenvolvimento do sistema, quer seja diretamente como técnica de modelagem de sistema, quer seja na sua utilização em metodologias de desenvolvimento ou em ferramentas de apoio (Ferramentas Case).
- Visa representar de forma relevante o sistema a ser desenvolvido, para que este possa ser construído de acordo com os requisitos e mantido ao longo de sua vida útil.
- Na primeira metade da década de 90 surgiram várias propostas de técnicas para modelagem de sistemas. Houve uma grande proliferação de propostas para modelagem de sistemas segundo o paradigma orientado a objetos.



# ENGENHARIA DE SOFTWARE

## Modelagem de Software

- Existiam diferentes notações gráficas para modelar uma mesma perspectiva de um sistema.
- Percebeu-se a necessidade de um padrão para a modelagem de sistemas, que fosse aceito e utilizado amplamente.
- Surgiram esforços nesse sentido de padronização, sendo o principal liderado por James Rumbaugh, Grady Booch e Ivar Jacobson (Rational).
- Todos os sistemas podem ser descritos sob diferentes aspectos, com a utilização de modelos distintos, e cada modelo será, portanto, uma abstração semanticamente específica do sistema.



# ENGENHARIA DE SOFTWARE

## Modelagem de Software



- Os modelos podem ser estruturais, dando ênfase à organização do sistema, ou podem ser comportamentais, dando ênfase à dinâmica do sistema.
- De acordo com Booch, Rumbaugh e Jacobson há quatro objetivos principais para se criar modelos:
  - Eles ajudam a visualizar o sistema como ele é ou como desejamos que ele seja;
  - Eles permitem especificar a estrutura ou o comportamento de um sistema;
  - Eles proporcionam um guia para a construção do sistema;
  - Eles documentam as decisões tomadas no projeto.



# ENGENHARIA DE SOFTWARE

## Modelagem de Software



- **A modelagem não se restringe a grandes sistemas.** Porém é absolutamente verdadeiro que, quanto maior e mais complexo for o sistema, maior será a importância da modelagem, por uma razão muito simples;
- **Construímos modelos de sistemas complexos porque não é possível compreendê-los em sua totalidade.** A escolha dos modelos a serem criados tem profunda influência sobre a maneira como um determinado problema é atacado e como uma solução é definida.
- Cada modelo poderá se expresso em diferentes níveis de precisão. Os melhores modelos estão relacionados à realidade. **Nenhum modelo único é suficiente. Qualquer sistema não-trivial será melhor investigado por meio de um pequeno conjunto de modelos quase independentes.**



# ENGENHARIA DE SOFTWARE

## Modelagem de Software



- Consiste na utilização de notações gráficas e textuais com o objetivo de construir modelos que representam as partes essenciais de um sistema, considerando-se diversas perspectivas diferentes e complementares.
- Assim como em outras áreas, em uma abordagem de Engenharia de Software inicialmente, o problema a ser tratado deve ser analisado e decomposto em partes menores, “**dividir para conquistar**”.
- “Uma empresa de software de sucesso é aquela que consistentemente produz software de qualidade que vai ao encontro das necessidades dos seus usuários”.



# ENGENHARIA DE SOFTWARE

## Modelagem de Software



- Uma empresa que consegue desenvolver tal software, de forma previsível, cumprindo os prazos, com uma gestão de recursos, quer humanos, quer materiais, eficiente e eficaz, é uma empresa que tem um negócio sustentado". Grady Booch, James Rumbaugh, Ivar Jacobson (UML - Guia do Usuário).
- Deve ser feita de acordo com os processos e padrões adotados e estabelecidos para o projeto e estes, por suas vezes, devem estar alinhados com as diretrizes e conceitos da Engenharia de Software.
- A Engenharia de Software trata de aspectos relacionados ao estabelecimento de processos, métodos, técnicas, ferramentas e ambientes de suporte ao desenvolvimento de software.



# ENGENHARIA DE SOFTWARE

## UML - Unified Modeling Language



- Começou a ser definida a partir de uma tentativa de Jim Rumbaugh e Grady Booch de combinar dois métodos populares de modelagem orientada a objeto: Booch e OMT (Object Modeling Language).
- Mais tarde, Ivar Jacobson, o criador do método Objectory, uniu-se aos dois (formando os famosos três amigos), para a concepção da primeira versão da linguagem UML (Unified Modeling Language)
- Linguagem para elaboração da estrutura de projetos de software
- Indica como criar e ler modelos de software bem formados.



# ENGENHARIA DE SOFTWARE

## UML - Unified Modeling Language



- Não aponta quais modelos deverão ser criados, ou seja, eles são determinados pelo projeto.
- Não aponta quando os modelos deverão ser criados. Esta tarefa cabe ao Processo de Desenvolvimento de Software
- Linguagem para elaboração da estrutura de projetos de software
- É uma linguagem de modelagem utilizada ao longo de todo o ciclo de vida de Desenvolvimento de Software.
- Os modelos se aplicam desde a concepção do projeto até a entrega do mesmo



# ENGENHARIA DE SOFTWARE

## UML - Unified Modeling Language



- Possui diagramas para serem entregues utilizados para a modelagem de:
  - Negócios
  - Requisitos
  - Análise
  - Projeto
  - Implementação
  - Implantação
- Surge a UML 0.9 em 1996 como a melhor candidata para ser a linguagem unificadora de notações.
- Em 1997 a UML 1.0 é aprovada como padrão pela OMG (Object Management Group), Desde então tem tido grande aceitação, atualmente está na versão 2.5



# ENGENHARIA DE SOFTWARE

## UML - Unified Modeling Language



- É uma linguagem visual
- É independente de linguagem de programação
- É independente de processo de desenvolvimento
- **Não** é uma linguagem de programação
- **Não** é uma metodologia



# ENGENHARIA DE SOFTWARE

## UML - Unified Modeling Language



- É uma linguagem visual
- É independente de linguagem de programação
- É independente de processo de desenvolvimento
- **Não** é uma linguagem de programação
- **Não** é uma metodologia
  
- Pode ser usada para:
  - Mostrar os limites de um sistema e suas funções
  - Representa a estrutura estática de um sistema
  - Modelar o comportamento de objetos
  - Apresentar a implementação física e a arquitetura de um sistema

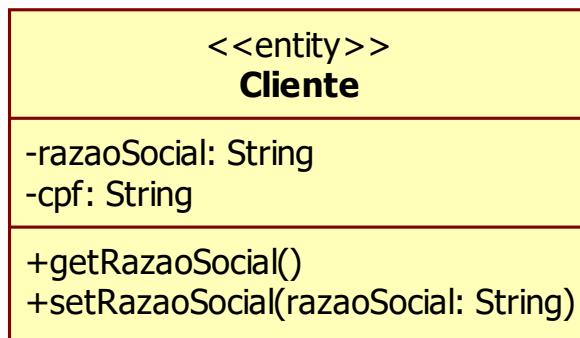


# ENGENHARIA DE SOFTWARE

## UML - Unified Modeling Language



Classe



Objeto



Interface

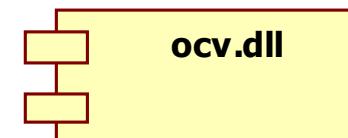


IImprimivel

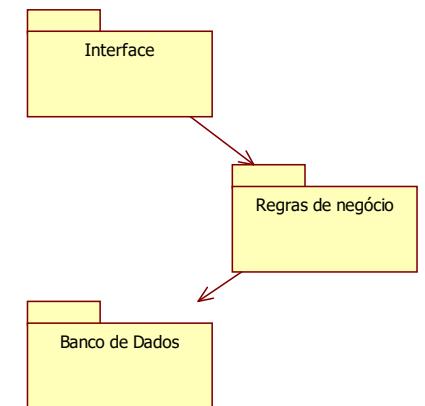
Nó



Componente



Pacotes



Relacionamentos



Notas

Notas servem para adicionar  
alguma informação nos diagramas

Tagged Values

{nome=João da Silva  
Cpf=12345678901}

# ENGENHARIA DE SOFTWARE



## Diagrama de Casos de Uso

⇒ O diagrama de **CASOS DE USO** procura, por meio de uma linguagem simples, possibilitar a compreensão do comportamento externo do sistema por qualquer pessoa, através da perspectiva do usuário

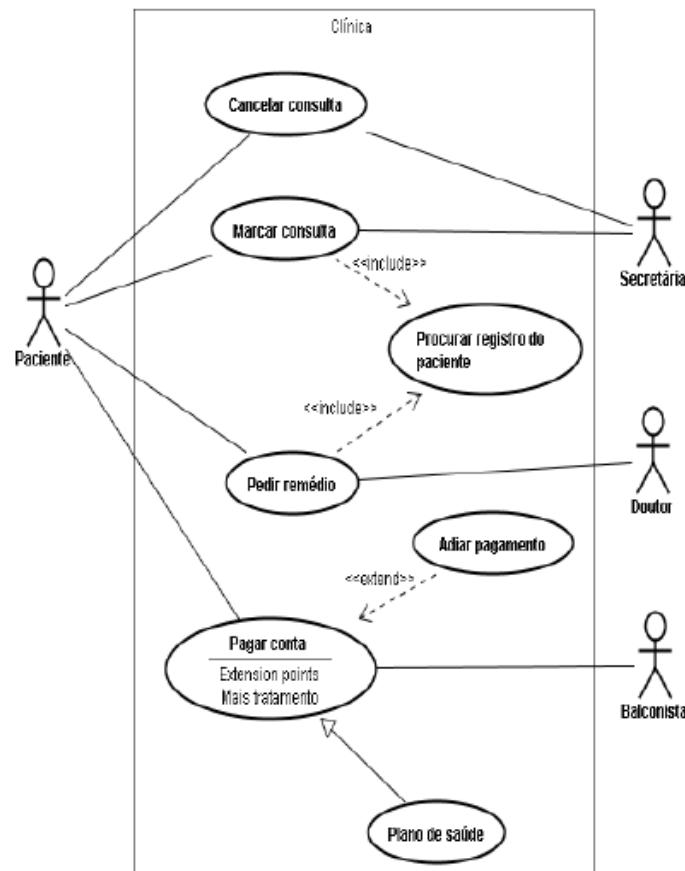
- Diagrama mais **ABSTRATO**
- Diagrama mais **FLEXÍVEL**
- Diagrama mais **INFORMAL**
- MAS extremamente importante ...
- Mapeamento dos REQUISITOS
- Base para os demais diagramas da UML

# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso



- ➊ O caso de uso é representado por uma elipse conectada a símbolos de atores.
- ➋ O retângulo (opcional) mostra os limites (fronteiras) do sistema, contendo em seu interior o conjunto de casos de uso e, ao lado externo, os atores interagindo com o sistema.
- ➌ Cada linha entre um ator e uma elipse de caso de uso mostra uma associação entre eles.
- ➍ Pode haver relacionamentos de inclusão e de extensão entre casos de uso.
- ➎ Uma linha tracejada com a palavra-chave <<include>> ou <<extend>> identifica estes relacionamentos.



# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso



### ATORES

- Representam os papéis desempenhados pelos diversos usuários que poderão utilizar de alguma maneira os serviços e funções do sistema.
- Normalmente → PESSOAS
- Eventualmente → HARDWARE – SOFTWARE que interajam com o sistema



Gerente



Cliente



Funcionário



Caixa Eletrônico



Sistema de Contas Pagar/Receber

# ENGENHARIA DE SOFTWARE



## Diagrama de Casos de Uso

### CASOS DE USO

- Referem-se aos serviços, tarefas ou funções que podem ser utilizados pelos usuários do sistema
- Utilizados para expressar/documentar os comportamentos pretendidos para as funções do sistema



# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso

### CASOS DE USO - DOCUMENTAÇÃO



- Descrever, através de uma linguagem simples:
  - A função em linhas gerais do caso de uso;
  - Quais atores interagem com o mesmo;
  - Quais etapas devem ser executadas pelo ator e pelo sistema;
  - Quais parâmetros devem ser fornecidos;
  - Quais as restrições/validações o caso de uso deve possuir
- UML não tem formato oficial/específico

# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso

### CASOS DE USO - DOCUMENTAÇÃO



Caso de Uso	Comprar Itens
Atores	Cliente, Caixa
Descrição	Um Cliente chega a um ponto de pagamento, com vários itens que deseja comprar. O Caixa registra os itens de compra, informa o total da compra ao cliente e recebe o pagamento..

- Os detalhes dos casos de uso não são formalizados pela UML e podem ser adequados para satisfazer as necessidades e o espírito de documentação necessária (clareza de comunicação).
- Os casos de uso podem ser detalhados através de uma Sequência Típica de Eventos

# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso

### CASOS DE USO - DOCUMENTAÇÃO



Seqüência Típica de Eventos	
Ação do Ator	Resposta do Sistema
1. Este caso de uso começa quando um Cliente chega a um ponto de pagamento com vários itens que deseja comprar.	
2. O Caixa registra o código de cada item Se houve mais de um exemplar do mesmo item, o Caixa também pode entrar a quantidade	3. Determina o preço do item e acrescenta a informação sobre o item à transação corrente de venda.
4. Ao término da entrada de itens, o Caixa indica ao POST que a entrada de itens está completa	5. Calcula e apresenta o total da venda
6. O Caixa informa ao Cliente o total da compra	
7. O Cliente dá um pagamento em dinheiro, possivelmente maior que o total da venda	
8. O Caixa registra o montante de dinheiro recebido	9. Exibe o valor do troco a ser devolvido ao Cliente
10. O Caixa deposita o dinheiro recebido e retira o troco	11. Registra a venda completada
12. O Cliente sai com os itens comprados	

# ENGENHARIA DE SOFTWARE



- ➲ Consequentemente há vários formatos de descrição propostos na literatura, assim como vários os graus de abstração utilizados.
- ➲ Alguns tipos de formato comumente utilizados são a **Descrição contínua**, a **Descrição numerada** e a **Descrição particionada**.
- ➲ O formato da **Descrição contínua** foi introduzido por Jacobson e seus colaboradores. Nesse formato, a narrativa é feita através de um texto livre.
- ➲ No formato da **Descrição numerada**, a narrativa é descrita através de uma série de passos numerados.
- ➲ O estilo da **Descrição particionada** tenta prover alguma estrutura à descrição de casos de uso.
- ➲ Nesse estilo, a seqüência de interações entre o ator e o sistema é particionada em duas colunas, uma para o ator e outra para o sistema.
- ➲ Essa forma de estruturação da narrativa tem o objetivo de separar as ações do ator e as reações do sistema.

# ENGENHARIA DE SOFTWARE



## Exemplo de Descrição Contínua:

O cliente chega ao caixa eletrônico e insere seu cartão. O sistema requisita a senha do cliente. Após o cliente fornecer a sua senha e esta ser validada, o sistema exibe as opções de operações possíveis. O cliente opta por realizar um saque. Então o sistema requisita o total a ser sacado. O sistema fornece a quantia desejada e imprime o recibo para o cliente.

## Exemplo de Descrição Numerada:

1. Cliente insere seu cartão no caixa eletrônico.
2. Sistema apresenta solicitação de senha.
3. Cliente digita senha.
4. Sistema exibe menu de operações disponíveis.
5. Cliente indica que deseja realizar um saque.
6. Sistema requisita quantia a ser sacada.
7. Cliente retira a quantia e o recibo.

# ENGENHARIA DE SOFTWARE



## Exemplo de Descrição Particionada

Cliente	Sistema
Insere seu cartão no caixa eletrônico.	
	Apresenta solicitação de senha.
Digita a senha.	
	Exibe menu de operações disponíveis.
Solicita a realização do saque.	
	Requisita quantia a ser sacada.
Retira a quantia e o recibo.	

# ENGENHARIA DE SOFTWARE



<b>Nome do Caso de Uso</b>	Abertura de Conta
<b>Caso de Uso Geral</b>	
<b>Autor Principal</b>	Cliente
<b>Atores Secundários</b>	Funcionário
<b>Resumo</b>	Este caso de uso descreve as etapas percorridas por um cliente para abrir uma conta-corrente
<b>Pré-Condições</b>	O pedido deve ser aprovado
<b>Pós-Condições</b>	É necessário realizar um depósito inicial
<b>Ações do Ator</b>	<b>Ações do Sistema</b>
1. Solicitar abertura de conta	
	2. Consultar o cliente por seu CPF
	3. Avaliar o pedido do cliente
	4. Aprovar o pedido
5. Escolher a senha da conta	
**** CONTINUA ****	
<b>Restrições/Validações</b>	Ser maior de idade/depósito inicial de R\$ 5.000,00

# ENGENHARIA DE SOFTWARE



## Diagrama de Casos de Uso ASSOCIAÇÕES

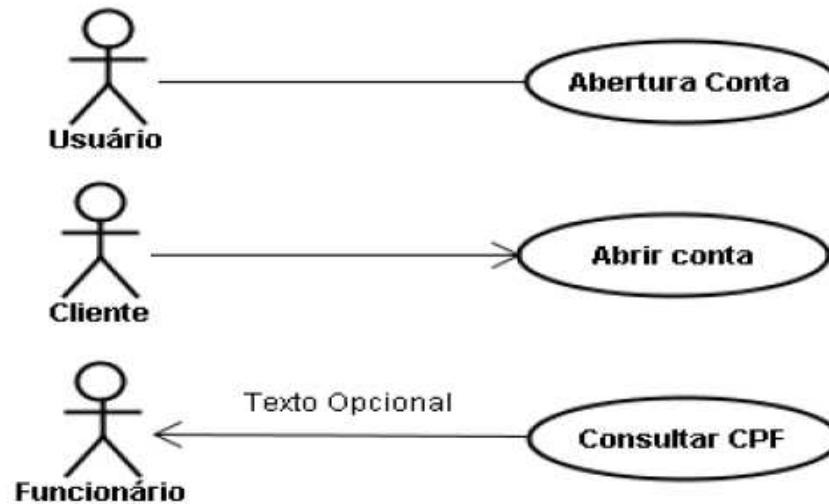
- Representam INTERAÇÕES/RELACIONAMENTOS entre:
  - ATORES
  - ATORES e CASOS DE USO
  - CASOS DE USO e CASOS DE USO
- Relacionamentos entre CASOS DE USO:
  - INCLUSÃO (include)
  - EXTENSÃO (extend)
  - GENERALIZAÇÃO

# ENGENHARIA DE SOFTWARE



## Diagrama de Casos de Uso ASSOCIAÇÕES

- ATOR ↔ CASO DE USO
- Demonstra que o ator utiliza-se da função do sistema representada pelo caso de uso – requisitando a execução, recebendo o resultado produzido.



# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso



### ASSOCIAÇÕES

- **ESPECIALIZAÇÃO/GENERALIZAÇÃO**
- Associação entre Casos de Uso com características semelhantes.
- A estrutura de um Caso de Uso generalizado é herdada pelos Casos de Usos especializados.



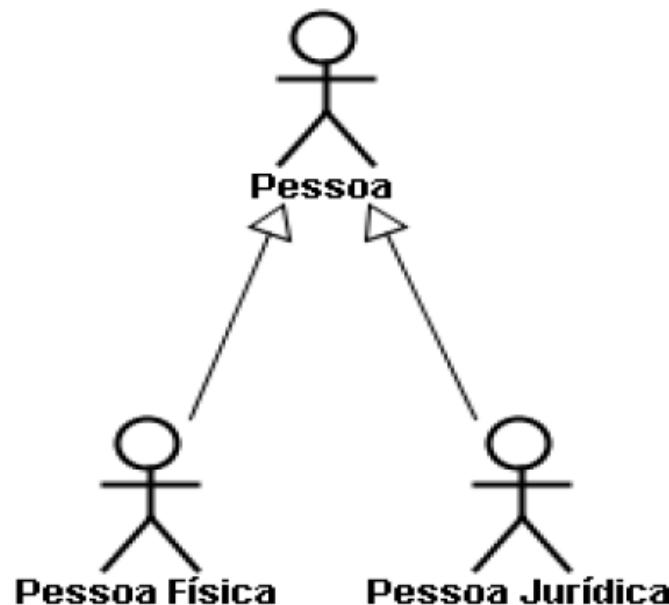
# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso



### ASSOCIAÇÕES

- ESPECIALIZAÇÃO/GENERALIZAÇÃO

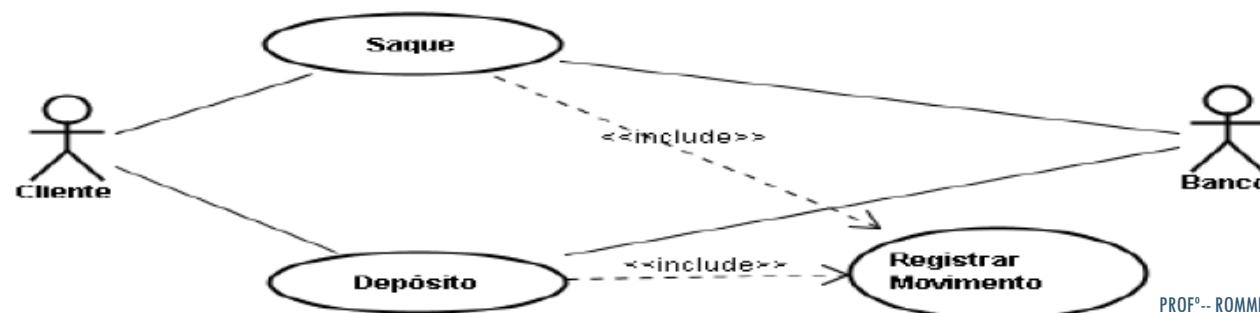


# ENGENHARIA DE SOFTWARE



## Diagrama de Casos de Uso ASSOCIAÇÕES

- **INCLUSÃO**
- Usada quando existe um serviço, situação ou rotina comum a mais de um Caso de Uso
- Outros Casos de Uso utilizam-se de um Caso de Uso “Chamada de Sub-Rotina”.
- Linha tracejada com texto “<<Include>>”.

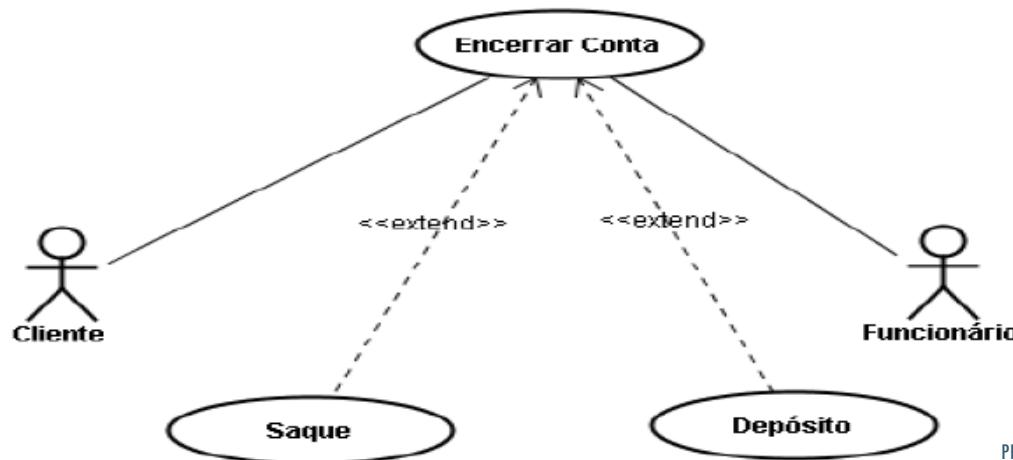


# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso ASSOCIAÇÕES



- **EXTENSÃO**
- Descrever cenários opcionais de um Caso de Uso
- Descrevem cenários que somente ocorrerão em uma situação específica – se uma determinada condição for satisfeita “<<Extend>>”.

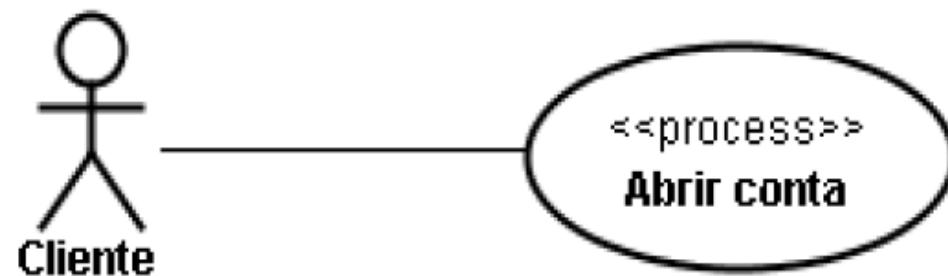


# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso



- **ESTEREÓTIPOS**
- Permitem a identificação de componentes, permitindo sua diferenciação dando maior destaque no diagrama.



# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso



- ➲ A UML não define uma estruturação específica a ser utilizada na descrição do formato expandido de um caso de uso.
- ➲ Há diversas propostas para descrever esse formato.
- ➲ No entanto, antes de começar a descrição deve-se atentar para o fato de que tais itens são uma sugestão.
- ➲ Pode ser que uma equipe de desenvolvimento não precise utilizar todos os itens.
- ➲ Pode até ser que mais detalhes sejam necessários.
- ➲ De qualquer modo, a equipe de desenvolvimento deve utilizar os itens de descrição que lhe forem realmente úteis.

# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso



### ➊ NOME

O mesmo nome que aparece no diagrama de casos de uso.  
Cada caso de uso deve ter um nome único.

### ➋ IDENTIFICADOR

Um código que permite fazer referência cruzada entre diversos documentos relacionados com o modelo de casos de uso ( por exemplo, a descrição de um cenário do caso de uso pode fazer referência a esse identificador).

Uma convenção que se recomenda é usar o prefixo CSU seguido de um número seqüencial Por exemplo: CSU01, CSU02.

### ➌ IMPORTÂNCIA

A definição da categoria de importância atribuída ao caso de uso.

# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso



### ➊ SUMÁRIO

Uma pequena descrição do caso de uso (no máximo de duas frases).

### ➋ ATOR PRIMÁRIO

O nome do ator que inicia o caso de uso.

### ➌ ATORES SECUNDÁRIOS

Os nomes dos demais atores participantes do caso de uso, se existirem.

### ➍ PRÉ-CONDIÇÕES

Pode haver alguns casos de uso cuja realização não faz sentido em qualquer momento, mas ao contrário, somente quando o sistema está em um determinado estado com certas propriedades.

Uma pré-condição de um caso de uso define que hipóteses são assumidas como verdadeiros para que o caso de uso tenha início. Por exemplo, “O cliente deve estar identificado no sistema”.

# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso



### ➊ FLUXO PRINCIPAL

Descreve o que normalmente acontece quando o caso de uso é realizado.

Toda descrição do caso de uso deve ter um item descrevendo o fluxo principal.

O texto descritivo desse fluxo deve ser claro e conciso.

Além disso, nessa descrição, o modelador deve se ater ao domínio do problema, e não à solução deste.

Portanto, o jargão computacional não deve ser utilizado na descrição de casos de uso, ao contrário, casos de uso devem ser escritos do ponto de vista do usuário e usando terminologia deste.

Essa dica vale igualmente para os **fluxos alternativos e de exceção**.

# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso



### ➊ FLUXO ALTERNATIVO

Esses fluxos podem ser utilizados para descrever o que acontece quando o ator faz uma escolha alternativa, diferente da descrita no fluxo principal, para alcançar o seu objetivo.

Fluxos alternativos também podem ser utilizados para descrever situações de escolhas exclusivas entre si (em que há diversas alternativas e somente uma deve ser realizada).

Uma dúvida que pode existir durante a descrição de um caso de uso é se um determinado comportamento deve ser descrito como um fluxo alternativo ou como um caso de uso de extensão.

Pode-se considerar o seguinte: um fluxo alternativo descreve um comportamento alternativo para a execução do fluxo principal. Ao contrário, uma extensão descreve um comportamento que funciona como uma interrupção em relação ao caso de uso principal.

De qualquer maneira, a decisão de utilizar um fluxo alternativo ou um caso de uso de extensão não terá tanta importância quanto o fato de ignorar a existência da seqüência alternativa.

# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso



### ⌚ FLUXO DE EXCEÇÃO

Correspondem à descrição das situações de exceção, que descrevem o que acontece quando algo inesperado ocorre na interação entre ator e caso de uso (por exemplo, quando um usuário realiza alguma ação inválida).

A importância de fluxos de exceção está no fato de podermos considerar situações não usuais, a partir das quais o sistema pode se recuperar (contornar a situação) ou pode cancelar a realizar da operação.

Um fluxo de exceção possui algumas características importantes, listadas a seguir:

- Representa um erro de operação durante o fluxo do caso de uso;
- Não tem sentido fora do contexto do caso de uso no qual ocorre;
- Deve indicar em que passo o caso de uso continua ou, conforme for, indicar explicitamente que o caso de uso termina.

Por exemplo considere o caso de uso REALIZAR PEDIDO. A seguir, são listados algumas situações não usuais que seriam tratadas em fluxos de exceção

- E se o cartão de crédito excede o limite?
- E se a loja não tem a quantidade requisitada do produto?
- E se o cliente já tem um débito anterior?

# ENGENHARIA DE SOFTWARE



## Diagrama de Casos de Uso

### ⇨ PÓS-CONDIÇÕES

Em alguns casos, em vez de gerar um resultado observável, o estado do sistema pode mudar após um caso de uso ser realizado.

Uma pós-condição é um estado que o sistema alcança após o caso de uso ter sido realizado.

A pós-condição não deve declarar como esse estado foi alcançado.

**Exemplos típicos de pós-condição:** uma (ou mais de uma) informação ter sido modificada, removida ou criada.

Pós-condições são normalmente descritas utilizando o tempo passado.

# ENGENHARIA DE SOFTWARE



## Diagrama de Casos de Uso

### ⌚ REGRAS DE NEGÓCIO

A descrição de um caso de uso também pode fazer referência a uma ou mais regras de negócio. São políticas, condições ou restrições que devem ser consideradas na execução de processos existentes em uma organização (Gottesdiener).

As regras de negócio constituem uma parte importante dos processos organizacionais porque descrevem a maneira como a organização funciona. A descrição do modelo de regras de negócio pode ser utilizando-se texto informal, ou alguma forma de estruturação.

Alguns exemplos de regras de negócio (não pertencentes a uma mesma organização):

- O valor total de um pedido é igual a soma dos totais dos itens do pedido acrescido de 10% de taxa de entrega;
- Um professor só pode estar lecionando disciplinas para as quais esteja habilitado;
- Um cliente do banco não pode retirar mais de R\$1.000 por dia de sua conta;
- Os pedidos para um cliente não-especial devem ser pagos antecipadamente;
- Para alugar um carro, o proponente deve estar com a carteira de motorista válida;
- O número máximo de alunos por turma é igual a 30;
- Um aluno deve ter a matrícula cancelada se obtiver dois conceitos D no curso;
- Uma vez que o professor confirma as notas de uma turma, estas não podem ser modificadas;
- Senhas devem ter, no mínimo, seis caracteres, entre números e letras, e devem ser atualizadas a cada três meses.

# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso



**Nome:** REALIZAR SAQUE

**Identificador:** CS01

**Importância:** O cliente realizará um saque no banco desejado a partir de um Caixa Eletrônico.

**Sumário:** O saque eletrônico é uma funcionalidade disponível no Sistema Bancário, que pode ser efetuada on-line através de transferência eletrônica ou nos terminais disponíveis em vários locais da cidade.

**Ator Primário:** Cliente

**Atores Secundários:** Sistema do Caixa Eletrônico

**Pré-Condições:** Para que o cliente possa realizar o saque ele deve ter o cartão bancário do banco constando a sua senha eletrônica e o código de validação para o saque ou o cartão de segurança.

**Fluxo Principal:**

- O cliente insere o cartão no terminal bancário.
- O sistema valida o cartão através da senha informada.
- O sistema disponibiliza as opções para o cliente realizar (saldo, extrato, depósitos, transferências).
- O cliente informa o valor desejado para o saque e insere o cartão para a confirmação da operação.

**Fluxos Alternativos:**

- O cliente poderá fazer a transferência do efetuar o saque sem o cartão desde que tenha o código de validação para o saque.
- O cliente pode solicitar pelo telefone do caixa eletrônico o saque automático na sua conta.

# ENGENHARIA DE SOFTWARE

## Diagrama de Casos de Uso



### Fluxos de Exceção:

- O cliente poderá habilitar novamente a sua senha em caso de digitação inadequada pelo telefone disponível no Caixa Eletrônico.
- O cliente poderá retirar o extrato e fazer depósitos sem o cartão bancário.

**Pós-condições:** O saque estipulado pelo cliente será descrito em um comprovante, após a entrega do valor no caixa eletrônico

### Regras de Negócio:

- Para que o cliente possa efetuar o saque deve ser validado inicialmente o seu cartão bancário com a senha.
- Caso seja digitado a senha por 3 vezes consecutivas de forma errada, o cartão deverá ser bloqueado

**Histórico:** O CASO DE USO FOI CRIADO EM 04/09/2013 às 19:00

**Para conectar uma regra de negócio a um caso de uso no qual ela é relevante, deve ser utilizado o identificador da regra de negócio que influenciar no caso de uso em questão.**

# ENGENHARIA DE SOFTWARE



## Diagrama de Classes

- ⇒ É o bloco mais importante dos sistemas orientados a objetos.
- ⇒ É uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica.
- ⇒ Implementa uma ou mais interfaces.
- ⇒ Utilizada para capturar o vocabulário do sistema que está em desenvolvimento.

# ENGENHARIA DE SOFTWARE



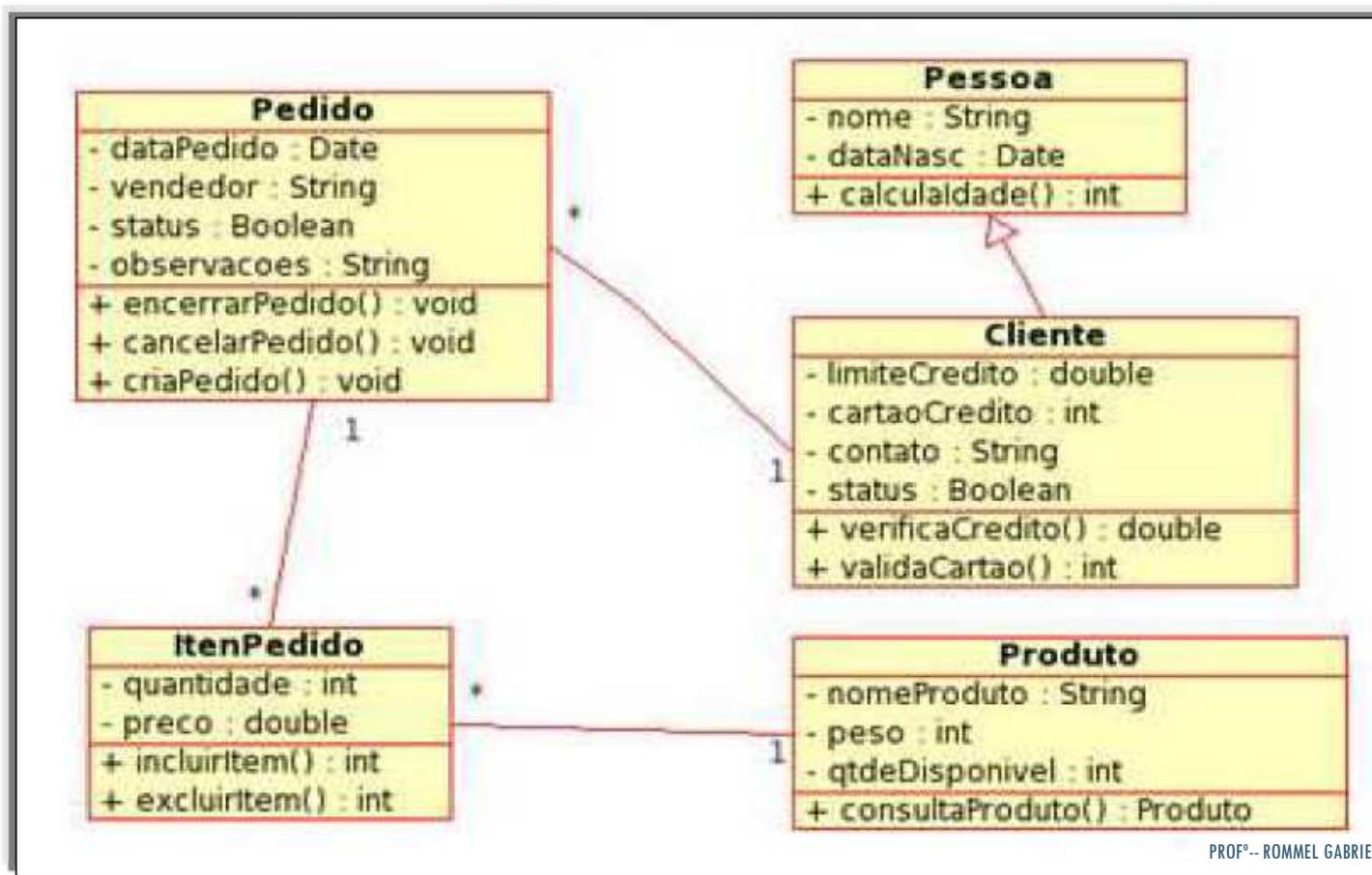
## Diagrama de Classes

- ➲ Pode incluir abstrações que são parte do domínio do problema.
- ➲ Representar item de software, item de hardware e até item puramente conceitual.
- ➲ Contém nome, atributos, operações, responsabilidade e visibilidades.
- ➲ Demonstra a estrutura estática das classes de um sistema onde estas representam as "coisas" que são gerenciadas pela aplicação modelada.

# ENGENHARIA DE SOFTWARE



## Diagrama de Classes



# ENGENHARIA DE SOFTWARE



## Diagrama de Classes

⇒ Quando esse diagrama é bem estruturado pode:

- Comunicar um único aspecto da visão estática do projeto do sistema.
- Conter somente os elementos essenciais à compreensão deste aspecto.
- Fornece detalhes consistentes com a abstração.

⇒ Quando criar esse diagrama:

- Dê-lhe um nome que comunique seu propósito;
- Minimize o cruzamento de linhas
- Use notas e cores para destacar características importantes
- Não exiba uma quantidade excessiva de tipos de relacionamentos

# ENGENHARIA DE SOFTWARE



## Diagrama de Classes Nome

- ➊ Deve ser um que diferencie das outras classes;
- ➋ É uma sequência de caracteres (não podendo ter o sinal de dois-pontos), podendo ter qualquer número de caracteres (mínimo um);
- ➌ Na prática, é um substantivo ou uma expressão breve, definido a partir do vocabulário do sistema

# ENGENHARIA DE SOFTWARE

## Diagrama de Classes Atributo



- ➲ É uma propriedade nomeada que descreve um intervalo de valores;
- ➲ Uma classe pode ter qualquer quantidade de atributos, até mesmo nenhum;
- ➲ Representa alguma propriedade do item que está sendo modelado;
- ➲ Uma abstração do tipo de dado ou estado que os objetos da classe podem abranger;
- ➲ Pode ser nomeado com um texto, assim como os nomes das classes.
- ➲ Na prática, é um substantivo ou uma expressão breve, que representa alguma propriedade da classe correspondente

# ENGENHARIA DE SOFTWARE



## Diagrama de Classes Operação

- ⌚ Implementa um serviço que pode ser solicitado por algum objetivo da classe para modificar o comportamento;
- ⌚ É uma abstração de algo que pode ser feito com um objeto e que é compartilhado por todos os objetos desta classe;
- ⌚ Uma classe pode ter qualquer quantidade de operações, até mesmo nenhuma.
- ⌚ Muitas vezes (mas nem sempre) a chamada a uma operação altera os dados ou o estado do objeto.
- ⌚ Pode ser nomeada com um texto, assim como os nomes das classes. Na prática, é um verbo ou uma locução verbal breve, representando algum comportamento da classe correspondente.
- ⌚ Pode receber parâmetros e retornar valores

# ENGENHARIA DE SOFTWARE



## Diagrama de Classes Responsabilidades

- ➊ Contrato ou obrigações de uma determinada classe;
- ➋ Todos os objetos desta classe têm o mesmo tipo de estado e o mesmo tipo de comportamento;
- ➌ É um texto de formato livre, praticamente uma expressão, uma oração ou um breve parágrafo

# ENGENHARIA DE SOFTWARE

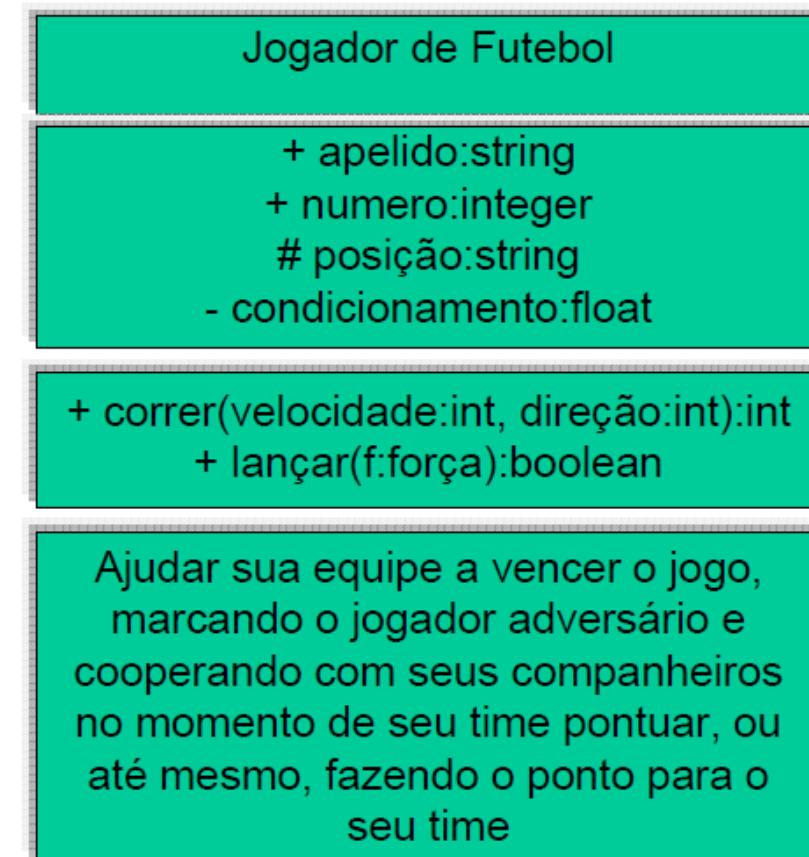
## Diagrama de Classes Visibilidade



- ➊ Um dos mais importantes detalhes que se pode especificar para os atributos e operações (métodos) de uma classe;
- ➋ Em UML podem ser determinados três níveis de visibilidade:
  - Público – visível 100% pelo mundo externo, especificado por ser atendido pelo símbolo de “+”.
  - Protegido – visível somente pelos dependentes da classe, especificado por ser atendido pelo símbolo de “#”.
  - Privado - visível somente pela classe, especificado por ser atendido pelo símbolo de “-”.

# ENGENHARIA DE SOFTWARE

## Diagrama de Classes



# ENGENHARIA DE SOFTWARE

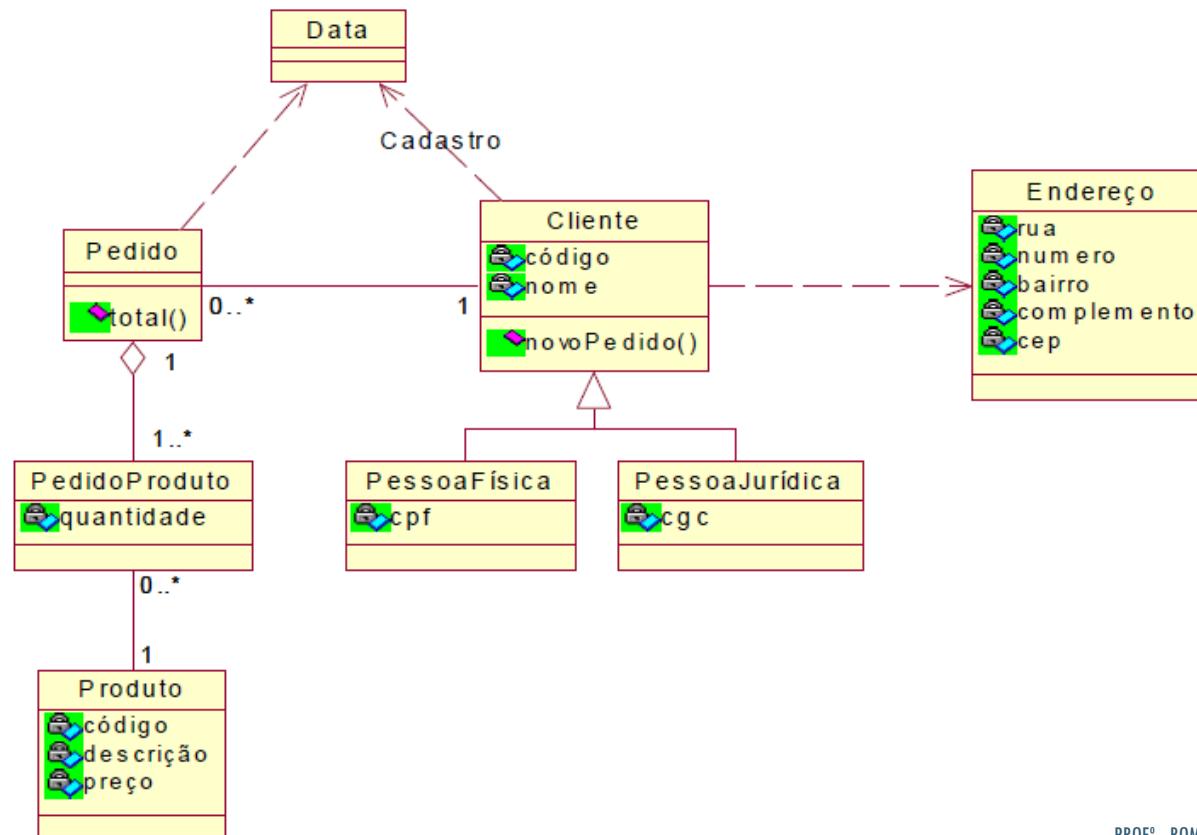


## Diagrama de Classes

- ➊ As classes podem se relacionar com outras através de diversas maneiras:
  - Associação (conectadas entre si);
  - Dependência (uma classe depende ou usa outra classe);
  - Especialização (uma classe é uma especialização de outra classe),
  - Pacotes (classes agrupadas por características similares).
- ➋ Todos estes relacionamentos são mostrados no diagrama de classes juntamente com as suas estruturas internas, que são os atributos e operações (métodos).

# ENGENHARIA DE SOFTWARE

## Diagrama de Classes



# ENGENHARIA DE SOFTWARE

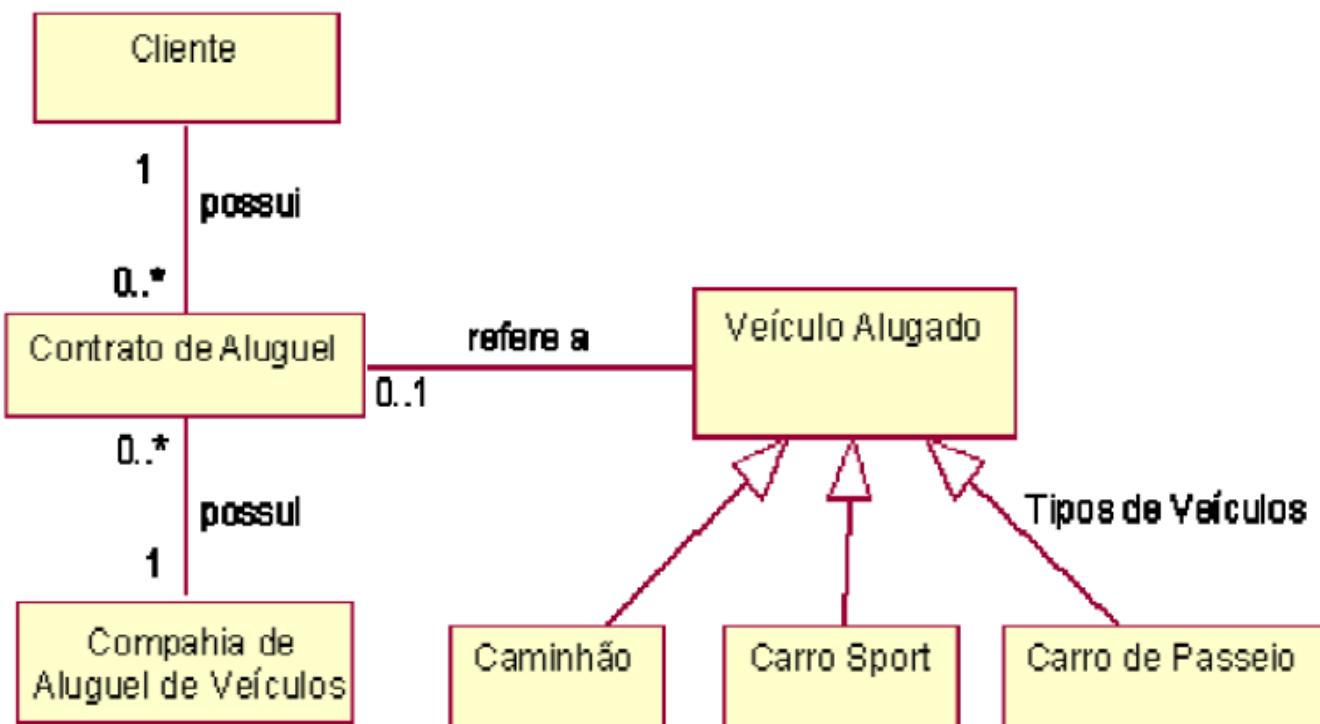
## Diagrama de Classes



- ⇒ É considerado estático já que a estrutura descrita é sempre válida em qualquer ponto do ciclo de vida do sistema.
- ⇒ Um sistema normalmente possui alguns diagramas de classes, já que não são todas as classes que estão inseridas em um único diagrama e certa classes pode participar de vários diagramas de classes.
- ⇒ Não servem só para visualização, especificação e documentação de modelos estruturais, mas também para a construção de sistemas executáveis por intermédio da engenharia de produção de software.
- ⇒ Uma classe num diagrama pode ser diretamente implementado utilizando-se uma linguagem de programação orientada a objetos que tenha suporte direto para construção de classes. Para criar um diagrama de classes, as classes têm que estar identificadas, descritas e relacionadas entre si.

# ENGENHARIA DE SOFTWARE

## Diagrama de Classes



# ENGENHARIA DE SOFTWARE



## Diagrama de Classes

⇒ Os elementos básicos de um diagrama de classes são:

- Classes
- Interfaces
- Relacionamentos:
  - ✓ Associação;
  - ✓ Dependência/Refinamento;
  - ✓ Generalização/Especialização
  - ✓ Composição/Agregação
  - ✓ Multiplicidade

# ENGENHARIA DE SOFTWARE

## Diagrama de Classes

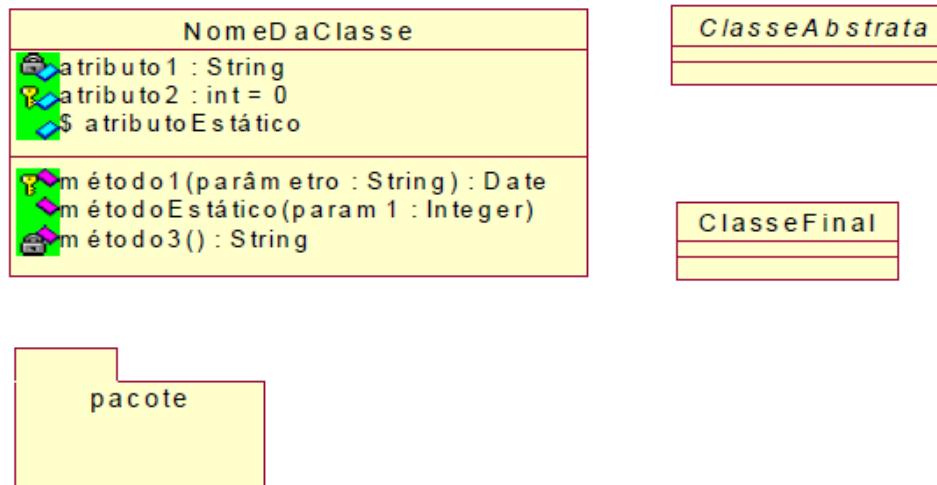


### ⌚ Classe:

- Tipo abstrato formado por atributos e métodos.

### ⌚ Pacote:

- Estrutura organizacional que agrupa elementos do diagrama de classes com o objetivo de melhor organizá-los.



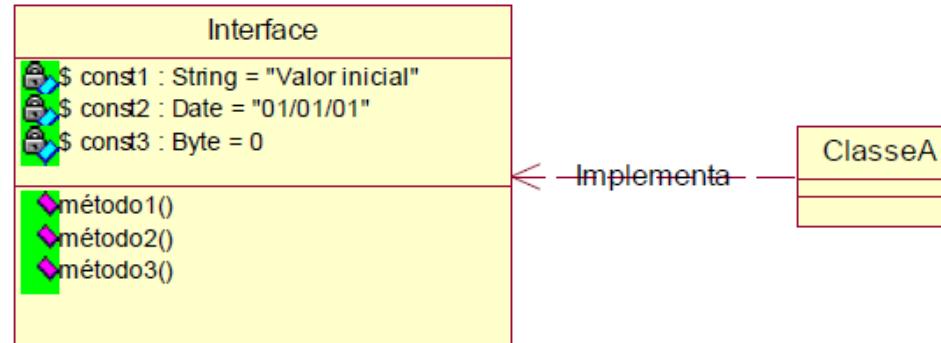
# ENGENHARIA DE SOFTWARE



## Diagrama de Classes

### ➊ Interfaces:

- Tipos especiais de classes possuindo apenas definições de métodos e/ou constantes.



# ENGENHARIA DE SOFTWARE



## Diagrama de Classes

### ⌚ Relacionamentos:

- Ao construir abstrações, descobre-se que há um número pequeno de classes que trabalham sozinhas;
- A maioria das classes colabora uma com as outras, de várias maneiras;
- Ao modelar o sistema é necessário identificar os itens que formam o vocabulário e como eles se relacionam;
- Em UML, o modo pelos quais os itens podem estar conectados a outros (lógica ou físico), são modelados os relacionamentos;
- Na modelagem orientada a objetos, existem três tipos de relacionamentos que são mais importantes: dependências, generalização e associações.

# ENGENHARIA DE SOFTWARE

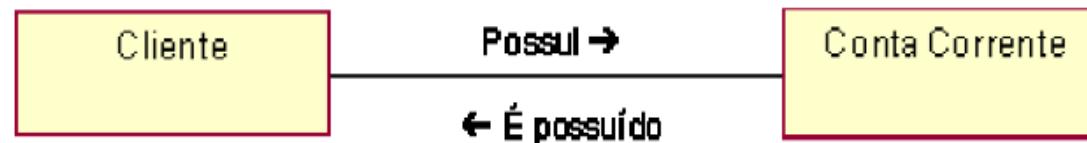


## Diagrama de Classes

### ⌚ Relacionamentos:

- **Associação**

- É um relacionamento estrutural, especificando que os objetos de uma classe estão conectados/vinculados a objetos de outra classe.
- Numa associação podem ser especificados os seguintes itens: um nome, o papel e multiplicidade em cada extremidade da associação, navegação e qualificação.
- Representa que duas classes possuem uma ligação (link) entre elas, significando, por exemplo, que elas "conhecem uma a outra", "estão conectadas com", "para cada X existe um Y" e assim por diante.



# ENGENHARIA DE SOFTWARE



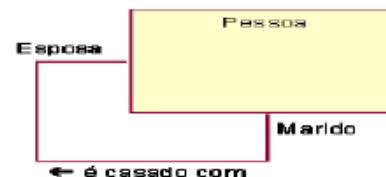
## Diagrama de Classes

### ⌚ Relacionamentos:

- Tipos de Associação:

- Recursiva

- É possível conectar uma classe a ela mesma através de uma associação e que ainda representa semanticamente a conexão entre dois objetos, mas os objetos conectados são da mesma classe.



- Qualificada

- São usadas com associações de um para vários (1..\*) ou vários para vários (\*). O "qualificador" (identificador da associação qualificada) especifica como um determinado objeto no final da associação "n" é identificado, e pode ser visto como um tipo de chave para separar todos os objetos na associação.
    - O identificador é desenhado como uma pequena caixa no final da associação junto à classe de onde a navegação deve ser feita.



# ENGENHARIA DE SOFTWARE

## Diagrama de Classes

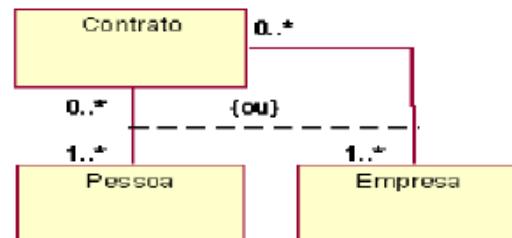


### ➊ Relacionamentos:

- Tipos de Associação:

- Exclusiva

- Ela especifica que objetos de uma classe podem participar de no máximo uma das associações em um dado momento. Uma associação exclusiva é representada por uma linha tracejada entre as associações que são parte da associação exclusiva, com a especificação "{ou}" sobre a linha tracejada.



- Ordenada

- As associações entre objetos podem ter uma ordem implícita. O padrão para uma associação é desordenada (ou sem nenhuma ordem específica). Mas uma ordem pode ser especificada através da associação ordenada.
    - Este tipo de associação pode ser muito útil em casos como este: janelas de um sistema têm que ser ordenadas na tela (uma está no topo, uma está no fundo e assim por diante).
    - A associação ordenada pode ser escrita apenas colocando "{ordenada}" junto à linha de associação entre as duas classes.

# ENGENHARIA DE SOFTWARE

## Diagrama de Classes

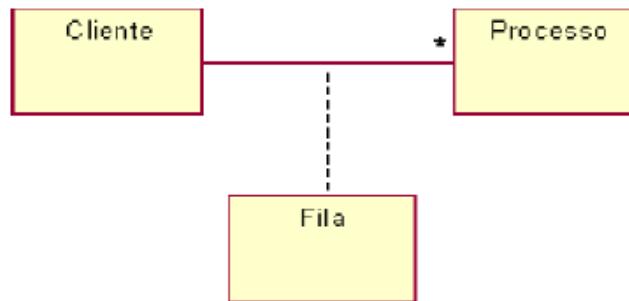


### ⌚ Relacionamentos:

- Tipos de Associação:

- De classe

- Uma classe pode ser associada a uma outra associação.
    - Este tipo de associação não é conectada a nenhuma das extremidades da associação já existente, mas na própria linha da associação.
    - Esta associação serve para se adicionar informações extra a associação já existente.



- A associação da classe Fila com a associação das classes Cliente e Processo pode ser estendida com operações de adicionar processos na fila, para ler e remover da fila e de ler o seu tamanho.
    - Se operações ou atributos são adicionados a associação, ela deve ser mostrada como uma classe.

# ENGENHARIA DE SOFTWARE



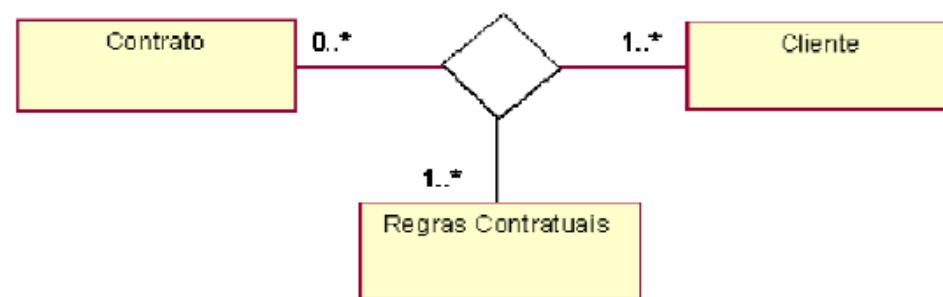
## Diagrama de Classes

### ⇨ Relacionamentos:

- Tipos de Associação:

- Ternária

- Mais de duas classes podem ser associadas entre si, a associação ternária associa três classes.
    - Ela é mostrada como um grande losango (diamante) e ainda suporta uma associação de classe ligada a ela, traçaria-se, então, uma linha tracejada a partir do losango para a classe onde seria feita a associação ternária.



- No exemplo acima a associação ternária especifica que um cliente poderá possuir 1 ou mais contratos e cada contrato será composto de 1 ou várias regras contratuais.

# ENGENHARIA DE SOFTWARE



## Diagrama de Classes

### ⇒ Tipos de Relacionamentos de Associação:

- Agregação:

- Associação utilizada para representar o relacionamento “todo/parte”.
- Relacionamento do tipo “tem”, o que significa que o objeto do todo tem os objetos das partes.
- Caso algum objeto das partes seja desativado, o objeto do todo se mantém ativo.

Exemplo: porta de fechadura



# ENGENHARIA DE SOFTWARE

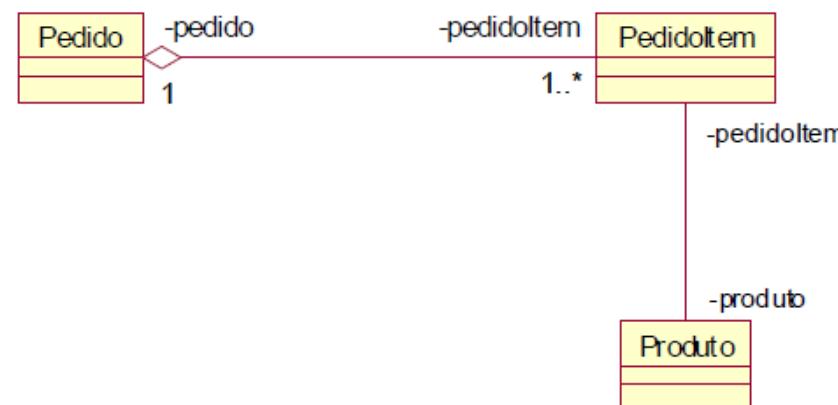


## Diagrama de Classes

### ⌚ Tipos de Relacionamentos de Associação:

- **Composição:**

- É uma associação do tipo agregação, cuja semântica é mais forte;
- Indicando que o objeto parte "é um atributo" do objeto todo, onde o ciclo de vida do objeto parte é limitado ao ciclo de vida do objeto todo.



# ENGENHARIA DE SOFTWARE

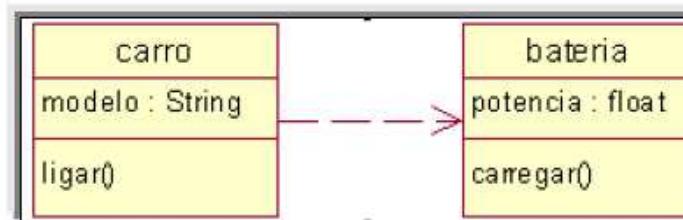
## Diagrama de Classes



### ⌚ Relacionamentos:

- **Dependência**

- Indica a ocorrência de um relacionamento semântico entre dois elementos do modelo, onde uma classe cliente é dependente de serviços de uma classe fornecedora, mas não existe uma dependência estrutural interna com esse fornecedor
- Indica que uma alteração na especificação de uma classe poderá afetar outra classe que a utilize, mas não necessariamente o inverso.
- A forma mais comum de dependência é quando uma classe utilizar parâmetros de uma outra classe em seus métodos.
- Uma relação de dependência é simbolizada por uma linha tracejada com uma seta no final de um dos lados do relacionamento.



# ENGENHARIA DE SOFTWARE



## Diagrama de Classes

### ➊ Relacionamentos:

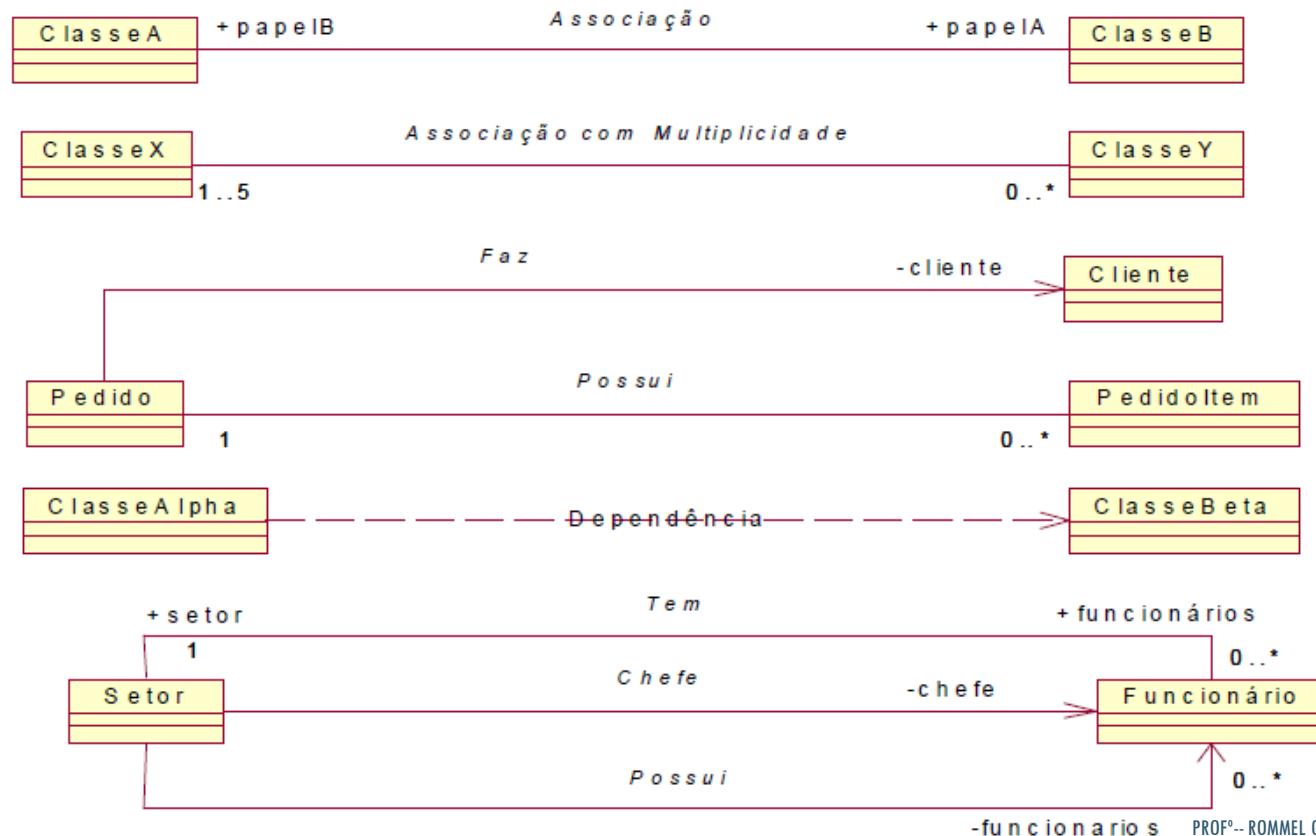
#### • Refinamentos

- Os refinamentos são um tipo de relacionamento entre duas descrições de uma mesma coisa, mas em níveis de abstração diferentes e podem ser usados para modelar diferentes implementações de uma mesma coisa (uma implementação simples e outra mais complexa, mas também mais eficiente).
- Os refinamentos são simbolizados por uma linha tracejada com um triângulo no final de um dos lados do relacionamento e são usados em modelos de coordenação.



# ENGENHARIA DE SOFTWARE

## Diagrama de Classes



# ENGENHARIA DE SOFTWARE

## Diagrama de Classes

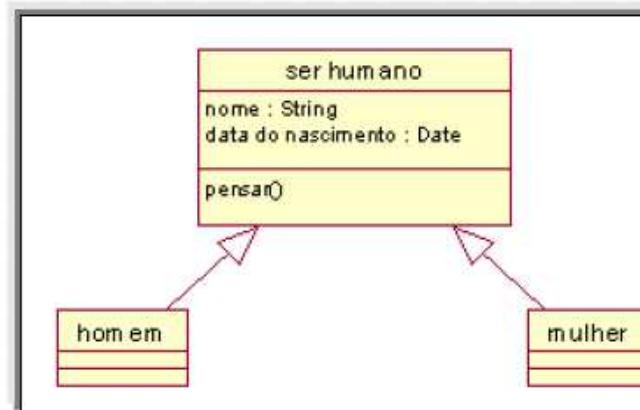


### ⌚ Relacionamentos:

- **Generalização/Especialização**

- Representa o conceito de herança da orientação a objetos e indica que uma classe estende outra.
- Conecta classe generalizada a outra classe especializada, o que é conhecido como relacionamento sub-classe/super-classe ou mãe/filha.

**Exemplo:** homens e mulheres são seres humanos



# ENGENHARIA DE SOFTWARE



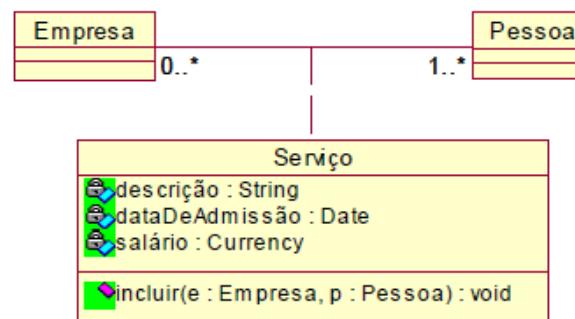
## Diagrama de Classes

### ⌚ Relacionamentos:

- **Multiplicidade**

- As associações permitem representar a informação dos limites inferior e superior da quantidade de objetos aos quais um outro objeto pode estar associado.
- Esses limites são chamados de multiplicidades. Na terminologia da UML.
- Cada associação em um diagrama de classes possui duas multiplicidades, uma em cada extremo da linha de associação.

Nome	Simbologia
Um para Um	1..1
Um para Muitos	1...*
Muitos para Muitos	1..*
Zero ou Um	0..1'





**ENGENHARIA DE SOFTWARE I**  
**SISTEMAS DE INFORMAÇÃO - 5º- PERÍODO - PASSOS/MG**