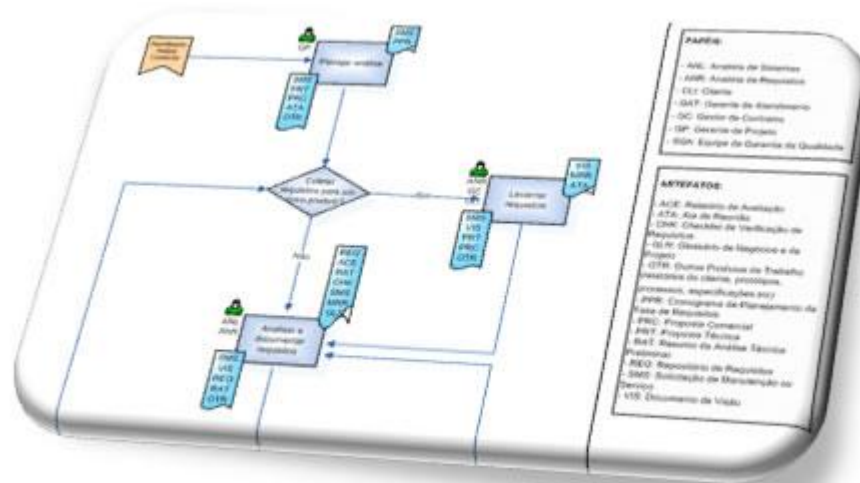


# Engenharia de Software

Prof. M.Sc. Sílvio Bacalá Júnior



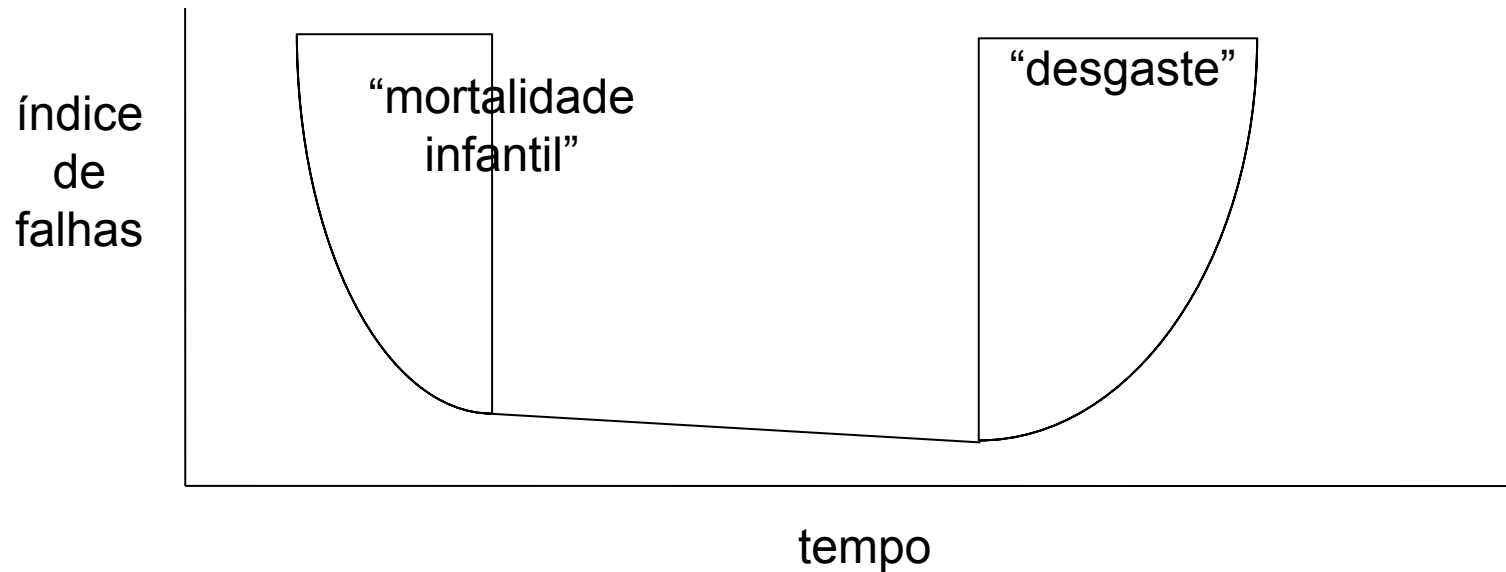
# O que é software?

- Programas de computador e documentação associada
- Produtos de software podem ser desenvolvidos para um cliente particular ou podem ser desenvolvidos para um mercado geral

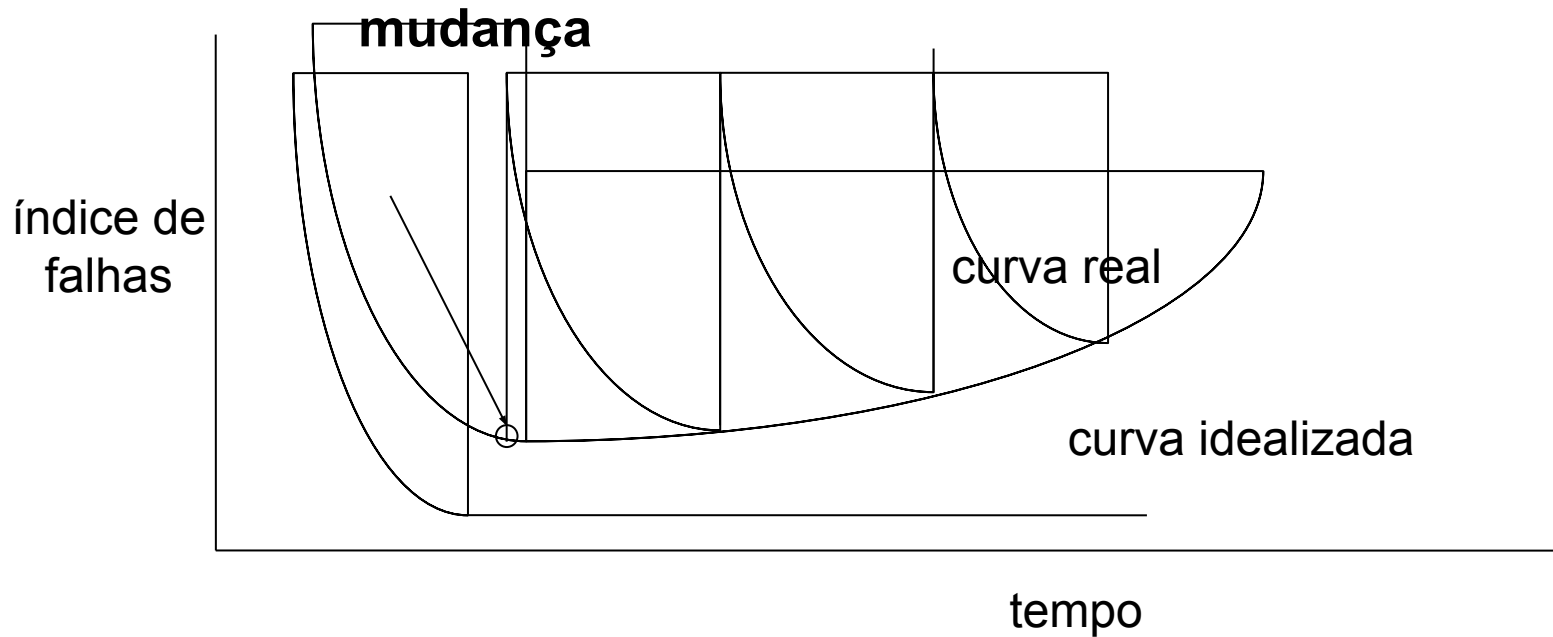
# Características do Software

1. desenvolvido ou projetado por engenharia, não manufaturado no sentido clássico
2. não se desgasta mas se deteriora
3. a maioria é feita sob medida em vez de ser montada a partir de componentes existentes

# Curva de falhas para o hardware



# Curva de falhas do software



# Custos do Software

- Os custos do software geralmente dominam os custos do sistema total.
  - Os custos de software em um PC são geralmente maiores que o custo do hardware
- Softwares são mais caros para manter do que para desenvolver.
  - Para sistemas com uma vida longa, os custos com manutenção podem ser muitas vezes maiores que os custos de desenvolvimento
- Engenharia de software preocupa-se com o desenvolvimento de softwares rentáveis

# Engenharia de Software

- As economias de TODAS as nações desenvolvidas são dependentes de software
- Cada vez mais os sistemas são controlados por software
- A engenharia de software se preocupa com teorias, métodos e ferramentas para o desenvolvimento profissional de software

# Aplicações do software

- BÁSICO coleção de programas escritos para dar apoio a outros programas
- DE TEMPO REAL software que monitora, analisa e controla eventos do mundo real
- COMERCIAL sistemas de operações comerciais e tomadas de decisões administrativas
- CIENTÍFICO E DE ENGENHARIA caracterizado por algoritmos de processamento de números



# Aplicações do software

- EMBUTIDO usado para controlar produtos e sistemas para os mercados industriais e de consumo
- DE COMPUTADOR PESSOAL envolve processamento de textos, planilhas eletrônicas, diversões, etc.
- DE INTELIGÊNCIA ARTIFICIAL faz uso de algoritmos não numéricos para resolver problemas que não sejam favoráveis à computação ou à análise direta

# Alguns problemas na construção de software

- Questões que caracterizaram as preocupações com o processo de desenvolvimento de software:
  - por que o software demora tanto para ser concluído?
  - por que os custos de produção têm sido tão elevados?
  - por que não é possível detectar todos os erros antes que o software seja entregue ao cliente?
  - por que é tão difícil medir o progresso durante o processo de desenvolvimento de software?

# problema de comunicação entre cliente e fornecedor

- Insatisfação do cliente com o sistema "concluído" ocorre devido aos projetos de desenvolvimento serem baseados em informações vagas sobre as necessidades e desejos do cliente;

# Falta de teste

- Qualidade do software é quase sempre suspeita, problema resultante da pouca atenção que foi dada, historicamente, às técnicas de teste de software (até porque o conceito de **qualidade de software é algo relativamente recente**);

# Programação sem controles

- a “cultura de programação” que ainda é difundida e facilmente aceita por estudantes e profissionais de Ciências da Computação;

# Como reduzir ou resolver estes problemas?

- Não existe uma abordagem **mágica** que seja a melhor para a solução destes problemas
- Os métodos devem ser suportados por um conjunto de ferramentas que permita automatizar o desenrolar das etapas do projeto
- Definição clara de critérios de qualidade e produtividade de software
- Estes aspectos caracterizam a **ENGENHARIA DE SOFTWARE**

# Definição

- Na literatura, pode-se encontrar diversas definições da Engenharia de Software:
  - *"O estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais" [NAU 69].*
  - *"Conjunto de métodos, técnicas e ferramentas necessárias à produção de software de qualidade para todas as etapas do ciclo de vida do produto."* [Krakowiak, 85]

# Evolução do software

- (1950 - 1965)
  - O hardware sofreu contínuas mudanças
  - O software era uma arte "secundária" para a qual havia poucos métodos sistemáticos
  - O hardware era de propósito geral
  - O software era específico para cada aplicação
  - Não havia documentação



# Evolução do software

- (1965 - 1975)
  - Multiprogramação e sistemas multiusuários
  - Técnicas interativas
  - Sistemas de tempo real
  - 1a. geração de SGBD's
  - Produto de software - software houses
  - Bibliotecas de Software
  - Cresce nro. de sistemas baseado em computador
  - Manutenção quase impossível

# Evolução do software

- ( a partir 1975)
  - Sistemas distribuídos
  - Redes locais e globais
  - Uso generalizado de microprocessadores - produtos inteligentes
  - Hardware de baixo custo
  - Impacto de consumo

# Evolução do software

- (Quarta era do software de computador)
  - Tecnologias orientadas o objetos
  - Sistemas especialistas e software de inteligência artificial usados na prática
  - Software de rede neural artificial
  - Computação Paralela

# Crise de software

- Refere-se a um conjunto de problemas encontrados no desenvolvimento de software:
- Estimativas de prazo e de custo imprecisas
  - “Não dedicamos tempo para coletar dados sobre o processo de desenvolvimento de software”
  - “Sem nenhuma indicação sólida de produtividade, não podemos avaliar com precisão a eficácia de novas ferramentas, métodos ou padrões”

# Crise de software

- A produtividade das pessoas da área de software não tem acompanhado a demanda por seus serviços
- “Os projetos de desenvolvimento de software normalmente são efetuados apenas com um vago indício das exigências do cliente”

# Crise de software

- A qualidade de software às vezes é menos que adequada
  - Só recentemente começam a surgir conceitos quantitativos sólidos de garantia de qualidade de software
- O software existente é muito difícil de manter
  - A tarefa de manutenção devora o orçamento destinado ao software
  - A facilidade de manutenção não foi enfatizada como um critério importante

# Motivação

- Mars Climate Orbiter
- Objetivo
  - Enviar sinais a partir de Marte, após seu pouso no planeta
- Desastre
  - Chocou-se com o planeta
- Motivo
  - Bug no software responsável pela conversão de medidas
- Prejuízo
  - 165 milhões de dólares

# Motivação

- Airbus Airbus 320
- Desastre
  - USS Vincennes derrubou um Airbus 320 em 1988
- Motivo
  - Bug no software no software de reconhecimento confundindo o avião com um F14
- Prejuízo
  - 290 mortes



# Motivação

- Máquina de Terapia Radiotiva
- Desastre
  - Overdose em pacientes sob tratamento
- Motivo
  - Inabilidade em gerenciar certas condições de disputa
- Prejuízo
  - Morte de 2 pessoas
  - 6 outras lesionadas

# Motivação

- London Ambulance Service
- Desastre
  - Serviço auxiliado por computador falhou nos dias 26 e 27 de novembro de 1992, gerando várias falhas, como o envio de 2 ambulâncias para o mesmo destino, envio de uma ambulância para um local estando outras mais próximas, etc
- Motivo
  - Tudo indica que o problema estava relacionado a alta carga de emergências durante o período.
- Prejuízo
  - 20 mortes

# Motivação

- Airbus A300 China Air Lines
- Desastre
  - Avião caiu em 1994
- Motivo
  - Foi feita uma investigação e, dentre as recomendações, aconselharam mudanças nos softwares de controle
- Prejuízo
  - 264 mortes

# Crise de software

- estimativas de prazo e de custo ↑
- produtividade das pessoas ↓
- qualidade de software ↓
- software difícil de manter ↑

# Causas dos problemas associados à crise de software

- PRÓPRIO CARÁTER DO SOFTWARE
  - Elemento de sistema lógico e não físico.
  - Sucesso medido pela qualidade de uma única entidade e não pela qualidade de muitas entidades manufaturadas
- O software não se desgasta, mas se deteriora

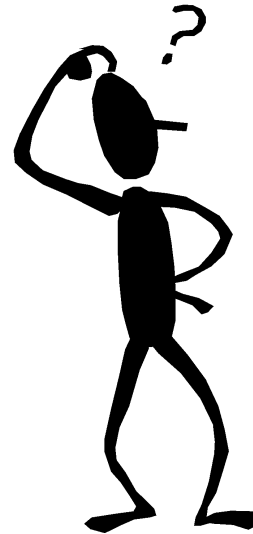
# Causas dos problemas associados à crise de software

- FALHAS DAS PESSOAS RESPONSÁVEIS PELO DESENVOLVIMENTO DE SOFTWARE
  - Gerentes com poucos fundamentos em software
  - Os profissionais têm recebido pouco treinamento formal em novas técnicas para o desenvolvimento de software
  - Resistência a mudanças.

# Causas dos problemas associados à crise de software

- MITOS DO SOFTWARE

- Propagaram desinformação e confusão
  - administrativos
  - cliente
  - profissional



# Mitos do software (ADMINISTRATIVOS)

- Mito: Já temos um manual repleto de padrões e procedimentos para a construção de software. Isso não oferecerá ao meu pessoal tudo o que eles precisam saber?
  - Realidade: Será que o manual é usado?
  - Os profissionais sabem que ele existe?
  - Ele reflete a prática moderna de desenvolvimento de software? Ele é completo?



# Mitos do software (ADMINISTRATIVOS)

- Mito: Meu pessoal tem ferramentas de desenvolvimento de software de última geração; afinal lhes compramos os mais novos computadores.
- Realidade: É preciso muito mais do que os mais recentes computadores para se fazer um desenvolvimento de software de alta qualidade.

# Mitos do software (ADMINISTRATIVOS)

- Mito: Se nós estamos atrasados nos prazos, podemos adicionar mais programadores e tirar o atraso.
- Realidade: O desenvolvimento de software não é um processo mecânico igual à manufatura. Acrescentar pessoas em um projeto torna-o ainda mais atrasado.
- Pessoas podem ser acrescentadas, mas somente de uma forma planejada.

# Mitos do software (CLIENTE)

- Mito: Uma declaração geral dos objetivos é suficiente para se começar a escrever programas - podemos preencher os detalhes mais tarde.
- Realidade: Uma definição inicial ruim é a principal causa de fracassos dos esforços de desenvolvimento de software. É fundamental uma descrição formal e detalhada do domínio da informação, função, desempenho, interfaces, restrições de projeto e critérios de validação.

# Mitos do software (CLIENTE)

- Mito: Os requisitos de projeto modificam-se continuamente, mas as mudanças podem ser facilmente acomodadas, porque o software é flexível.
- Realidade: Uma mudança, quando solicitada tardiamente num projeto, pode ser maior do que a ordem de magnitude mais dispendiosa da mesma mudança solicitada nas fases iniciais.

# Magnitude das mudanças

FASES	CUSTO DE MANUTENÇÃO
DEFINIÇÃO	1 x
DESENVOLVIMENTO	1.5 - 6x
MANUTENÇÃO	60 - 100x

# Mitos do software (PROFISSIONAL)

- Mito: Assim que escrevermos o programa e o colocarmos em funcionamento nosso trabalho estará completo.
- Realidade: Os dados da indústria indicam que entre 50 e 70% de todo esforço gasto num programa serão despendidos depois que ele for entregue pela primeira vez ao cliente.

# Mitos do software (PROFISSIONAL)

- Mito: Enquanto não tiver o programa "funcionando", eu não terei realmente nenhuma maneira de avaliar sua qualidade.
- Realidade: Um programa funcionando é somente uma parte de uma Configuração de Software que inclui todos os itens de informação produzidos durante a construção e manutenção do software.

# Relembrando a aula passada...



# Engenharia de Software

- Abrange três elementos fundamentais:
  - Métodos,
  - Ferramentas e
  - Procedimentos
- MÉTODOS: proporcionam os detalhes de como fazer para construir o software

# Engenharia de Software

- Planejamento e estimativa de projeto
- Análise de requisitos de software e de sistemas
- Projeto da estrutura de dados
- Algoritmo de processamento
- Codificação
- Teste
- Manutenção

# Engenharia de Software

- FERRAMENTAS: dão suporte automatizado aos métodos.
  - Existem atualmente ferramentas para sustentar cada um dos métodos
  - Quando as ferramentas são integradas é estabelecido um sistema de suporte ao desenvolvimento de software chamado CASE - *Computer Aided Software Engineering*

# Engenharia de Software

- PROCEDIMENTOS: constituem o elo de ligação entre os métodos e ferramentas
  - Sequência em que os métodos serão aplicados
  - Produtos que devem ser entregues
  - Controles que ajudam assegurar a qualidade e coordenar as alterações
  - Marcos de referência que possibilitam administrar o progresso do software.

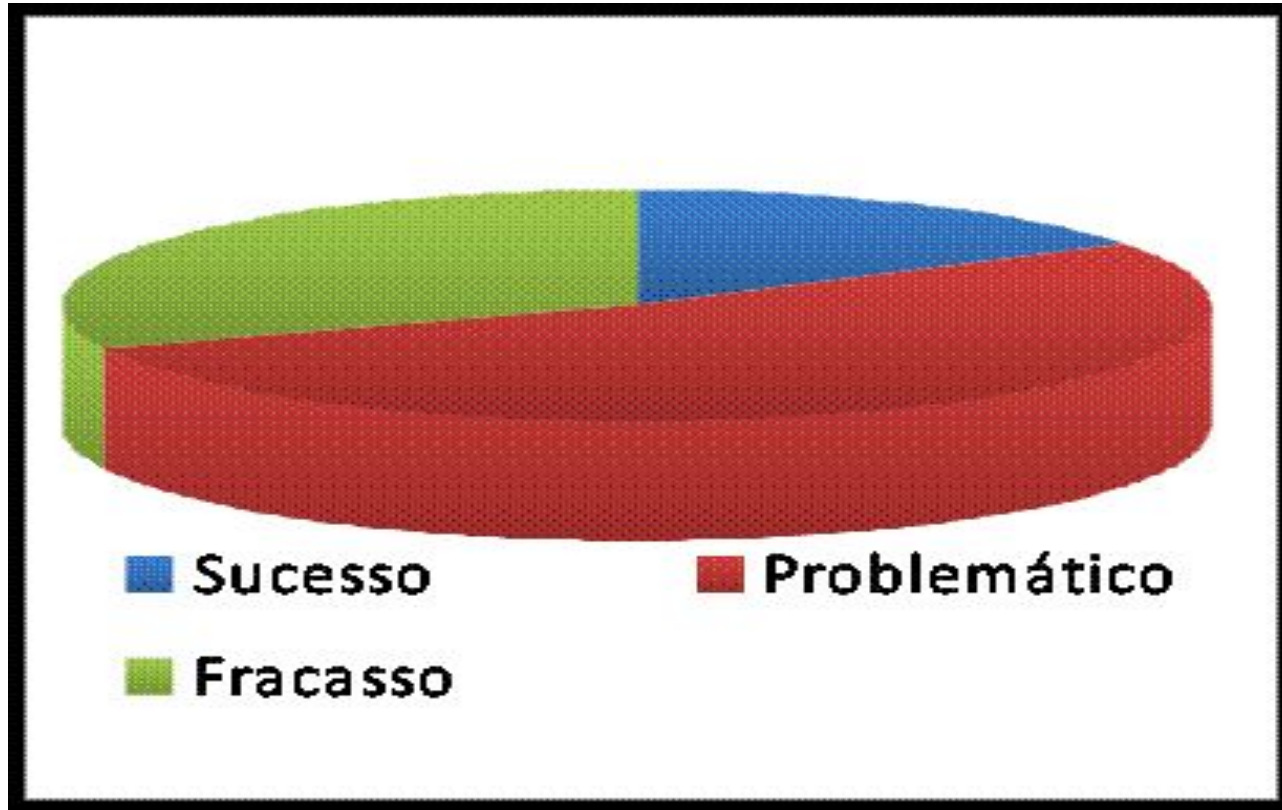
# Desenvolvimento de software

- Cenário atual e o idealizado estão distantes
- Fatores:
  - **não** uso dos fundamentos da engenharia de software para apoiar as atividades do desenvolvimento;
  - **mau** uso dos fundamentos da engenharia de software para apoiar as atividades do desenvolvimento.

# Cenário atual de desenvolvimento

	% do Custo de Desenvolvimento	% dos erros introduzidos	% dos erros encontrados	Custo relativo de correção
Análise de Requisitos	5	55	18	1
Projeto	25	30	10	1 - 1.5
Códificação e teste de unidade	50			
Teste	10	10	50	1 - 5
Validação e Documentação	10			
Manutenção		5	22	10 - 100

# Distribuição da conclusão dos projetos de software



Estudo realizado pelo Standish Group, considerando 350 companhias e 8.000 projetos de software, em 1995




# Fatores críticos para sucesso dos projetos de software

Fatores Críticos	%
1. Requisitos Incompletos	13,1%
2. Falta de Envolvimento do Usuário	12,4%
3. Falta de Recursos	10,6%
4. Expectativas Irreais	9,9%
5. Falta de Apoio Executivo	9,3%
6. Mudança de Requisitos e Especificações	8,7%
7. Falta de Planejamento	8,1%
8. Sistema não mais necessário	7,5%



# Quais os atributos de um bom software?

- O software deve fornecer as funcionalidades e performance requeridas para o usuário e deve ser fácil de manter, confiável e utilizável
- Manutenibilidade
  - O software deve evoluir para atender às necessidades de mudança
- Confiabilidade
  - O software deve ser confiável
- Eficiência
  - O software não deve fazer uso desnecessário de recursos do sistema
- Usabilidade
  - O software deve ser utilizável pelos usuários para os quais ele foi projetado

# Quais os principais desafios enfrentados pela engenharia de software?

- Lidando com sistemas legados, lidando com a diversidade crescente e lidando com a demanda de tempos para entrega reduzidos
- Sistemas legados
  - Sistemas antigos e de valor devem ser mantidos e atualizados
- Heterogeneidade
  - Os sistemas são distribuídos e incluem um misto de hardware e software
- Entrega
  - Existe uma pressão crescente para agilizar a entrega de software

# Características da Engenharia de Software

- A Engenharia de Software se refere a software (sistemas) desenvolvidos
  - por grupos ao invés de indivíduos
  - usa princípios de engenharia ao invés de arte, e
  - inclui tanto aspectos técnicos quanto não técnicos

# O que é um software de qualidade?

- O software que satisfaz os requisitos solicitados pelo usuário. Deve ser fácil de manter, ter boa performance, ser confiável e fácil de usar
- Alguns atributos de qualidade
  - Manutenibilidade
    - O software deve evoluir para atender os requisitos que mudam
  - Eficiência
    - O software não deve desperdiçar os recursos do sistema
  - Usabilidade
    - O software deve ser fácil de usar pelos usuários para os quais ele foi projetado

# Qualidade de Software (um exemplo para o Varejo)

- Correto
  - A loja não pode deixar de cobrar por produtos comprados pelo consumidor
- Robusto e altamente disponível
  - A loja não pode parar de vender
- Eficiente
  - O consumidor não pode esperar
  - A empresa quer investir pouco em recursos computacionais (CPU, memória, rede)

# Produtividade

- Custo de desenvolvimento reduzido
  - A empresa consumidora quer investir pouco em software
  - A empresa produtora tem que oferecer "software barato"
- Tempo de desenvolvimento reduzido
  - Suporte rápido às necessidades do mercado

# “Software Barato”

- Nem tanto resultado de baixos custos de desenvolvimento, mas principalmente da **distribuição dos custos entre vários clientes**.
- Reuso, extensibilidade e adaptabilidade são **essenciais** para viabilizar tal distribuição.

# Importância da Engenharia de Software

- Qualidade de software e produtividade garantem:
  - Disponibilidade de serviços essenciais
  - Segurança de pessoas
  - Competitividade das empresas
    - Produtores
    - Consumidores



# Mas, na realidade, temos a Crise de Software...

- 25% dos projetos são cancelados
- o tempo de desenvolvimento é bem maior do que o estimado
- 75% dos sistemas não funcionam como planejado
- a manutenção e reutilização são difíceis e custosas
- os problemas são proporcionais a complexidade dos sistemas

# Causas da Crise de Software

- Essências

- Complexidade dos sistemas
- Dificuldade de formalização

- Acidentes

- Má qualidade dos métodos, linguagens, ferramentas, processos, e modelos de ciclo de vida
- Falta de qualificação técnica

# Ciclo de Vida e Processo de Desenvolvimento de Software

# Ciclo de vida de um software

- Etapas pelas quais a construção de um software passa, desde a escolha e concepção do sw até a liberação para uso pelo usuários.



# CICLOS DE VIDA DE SOFTWARE

- Conjunto de etapas que envolve MÉTODOS, FERRAMENTAS e PROCEDIMENTOS.
- Alguns ciclos de vida mais conhecidos são:
  - Ciclo de Vida Clássico,
  - Prototipação,
  - Modelo Espiral e
  - Técnicas de 4a Geração

# Escolha de um Ciclo de Vida de software

Depende:

- da natureza do projeto e da aplicação
- dos métodos e ferramentas a serem usados
- dos controles e produtos que precisam ser entregues

# O que é um modelo de ciclo de vida de processo de software?

- Representação abstrata e simplificada do processo de desenvolvimento software
  - principais atividades e
  - dados
- usados na produção e manutenção de software

# Engenharia de Software uma visão genérica

- O processo de desenvolvimento de software contém 3 fases genéricas, independentes do modelo de engenharia de software escolhido:
  - DEFINIÇÃO,
  - DESENVOLVIMENTO e
  - MANUTENÇÃO.



# Engenharia de Software uma visão genérica

- FASE DE DEFINIÇÃO: “o que” será desenvolvido.
  - Análise do Sistema:
    - define o papel de cada elemento num sistema baseado em computador, atribuindo em última análise, o papel que o software desempenhará.
  - Planejamento do Projeto de Software:
    - assim que o escopo do software é estabelecido, os riscos são analisados, os recursos são alocados, os custos são estimados e, tarefas e programação de trabalho definidas.
  - Análise de Requisitos:
    - o escopo definido para o software proporciona uma direção, mas uma definição detalhada do domínio da informação e da função do software é necessária antes que o trabalho inicie.

# Engenharia de Software uma visão genérica

- DESENVOLVIMENTO: “como” o software vai ser desenvolvido.
  - Projeto de Software:
    - traduz os requisitos do software num conjunto de representações (algumas gráficas, outras tabulares ou baseadas em linguagem) que descrevem a estrutura de dados, a arquitetura do software, os procedimentos algorítmicos e as características de interface.
  - Codificação:
    - as representações do projeto devem ser convertidas numa linguagem artificial (a linguagem pode ser uma linguagem de programação convencional ou uma linguagem não procedimental) que resulte em instruções que possam ser executadas pelo computador.
  - Realização de Testes do Software:
    - logo que o software é implementado numa forma executável por máquina, ele deve ser testado para que se possa descobrir defeitos de função, lógica e implementação.

# Engenharia de Software uma visão genérica

- FASE DE MANUTENÇÃO:
  - concentra-se nas “mudanças” que ocorrerão depois que o software for liberado para uso operacional
    - Correção
    - Adaptação
    - Melhoramento Funcional

# Engenharia de Software uma visão genérica

- Correção:
  - mesmo com as melhores atividades de garantia de qualidade de software, é provável que o cliente descubra defeitos no software. A manutenção corretiva muda o software para corrigir defeitos.
- Adaptação:
  - com o passar do tempo, o ambiente original (por exemplo a CPU, o sistema operacional e periféricos) para o qual o software foi desenvolvido provavelmente mudará. A manutenção adaptativa muda o software para acomodar mudanças em seu ambiente.

# Engenharia de Software uma visão genérica

- Melhoria Funcional:
  - a medida que o software é usado, o cliente/usuário reconhecerá funções adicionais que oferecerão benefícios. A manutenção perfectiva estende o software para além de suas exigências funcionais originais.

# Engenharia de Software uma visão genérica

- ATIVIDADES DE PROTEÇÃO
  - Revisões:
    - efetuadas para garantir que a qualidade seja mantida à medida que cada etapa é concluída.
  - Documentação:
    - desenvolvida e controlada para garantir que informações completas sobre o software estejam disponíveis para uso posterior.
  - Controle das Mudanças:
    - instituído de forma que as mudanças possam ser aprovadas e acompanhadas.

# Principais Modelos do Ciclo de Vida de Software

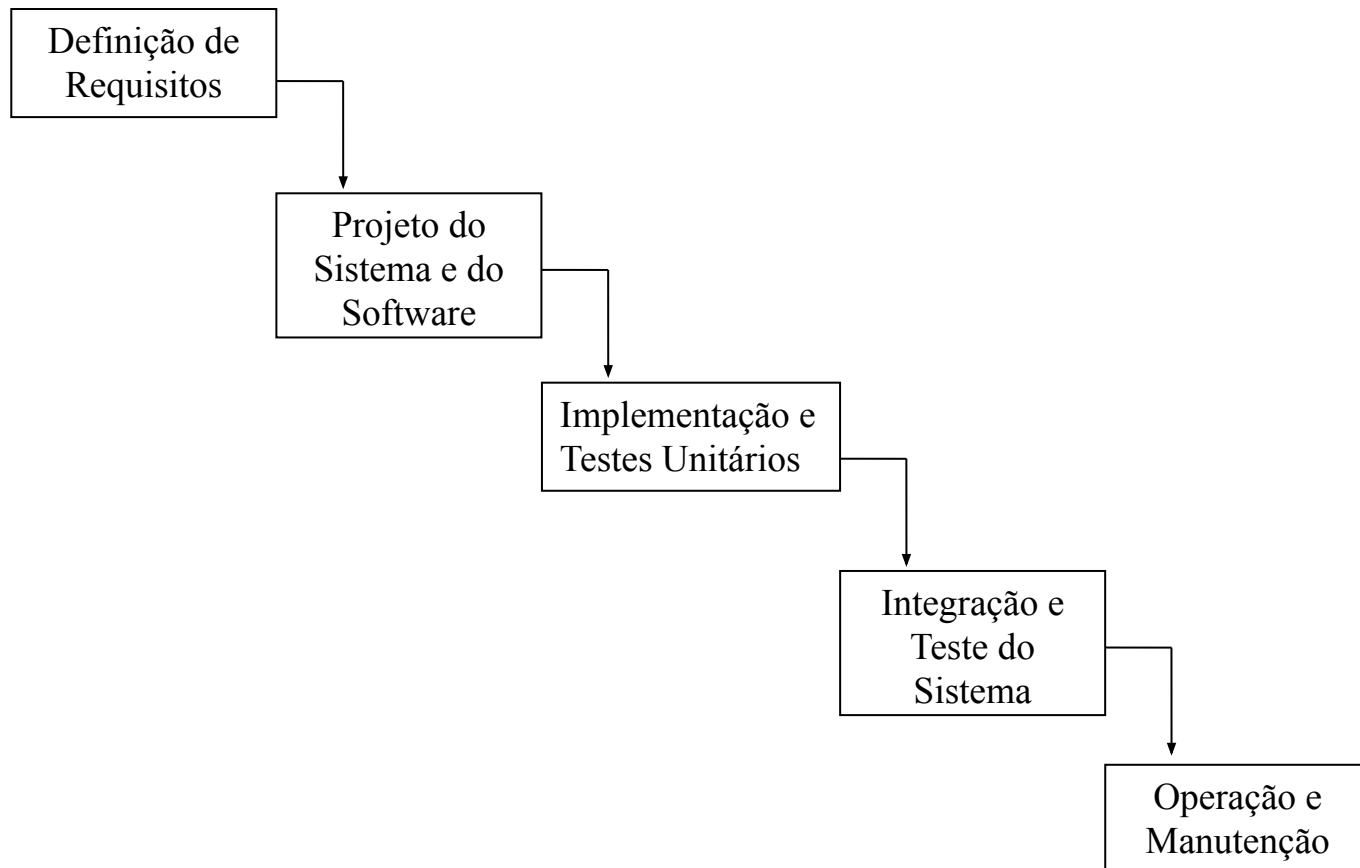
- Cascata
- Modelo de Desenvolvimento Evolucionário
  - Programação Exploratória
  - Prototipagem descartável
- Modelo de Transformação Formal
- Modelos Iterativos
  - Espiral
  - Incremental

# Modelo Cascata (ou clássico)

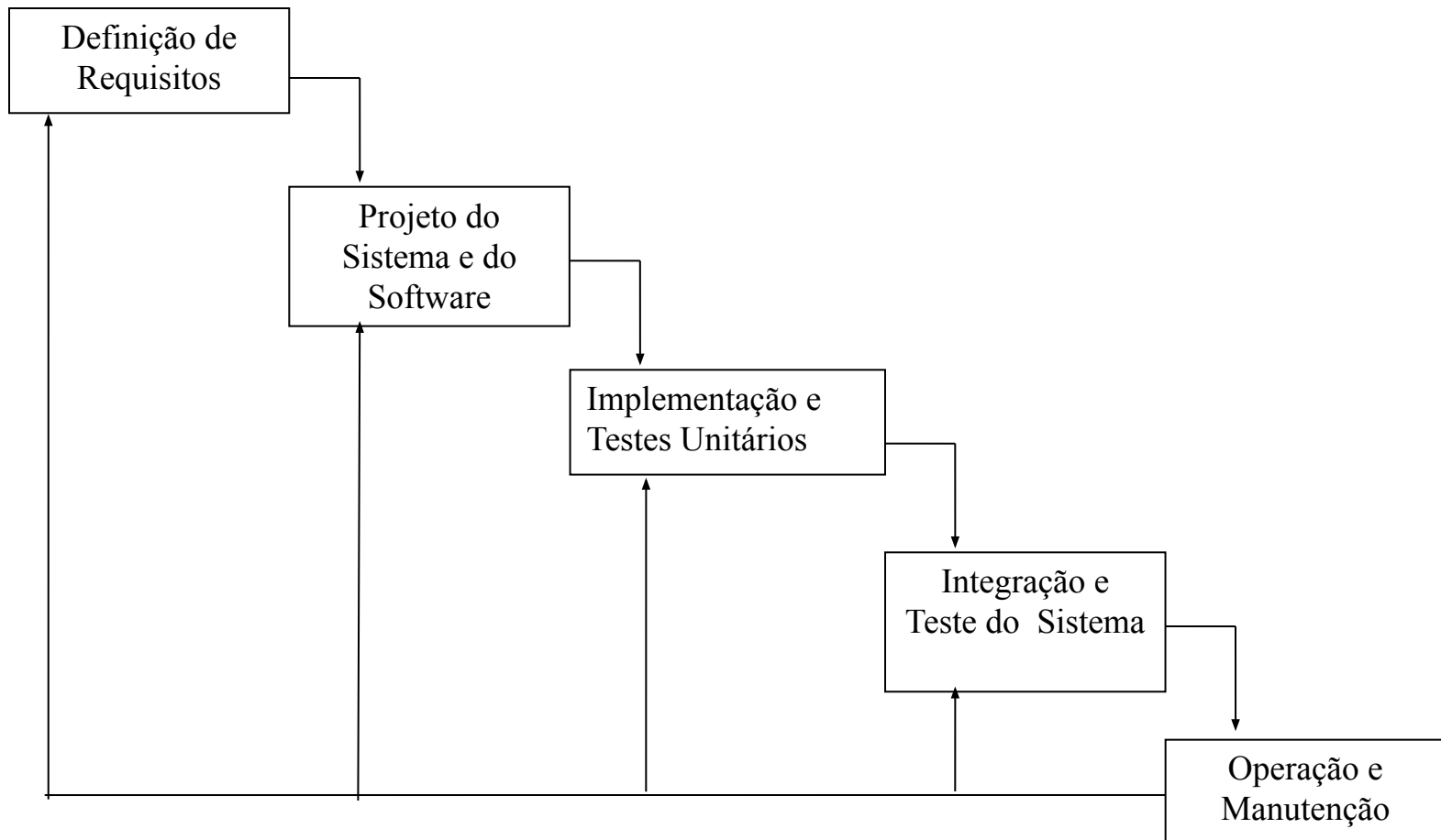
- Derivado de modelos existentes em outras engenharias
- Sua estrutura é composta por várias etapas que são executadas de forma sistemática e seqüencial
- Na prática, existe uma interação entre as etapas e cada etapa pode levar a modificações nas etapas anteriores



# Modelo Cascata



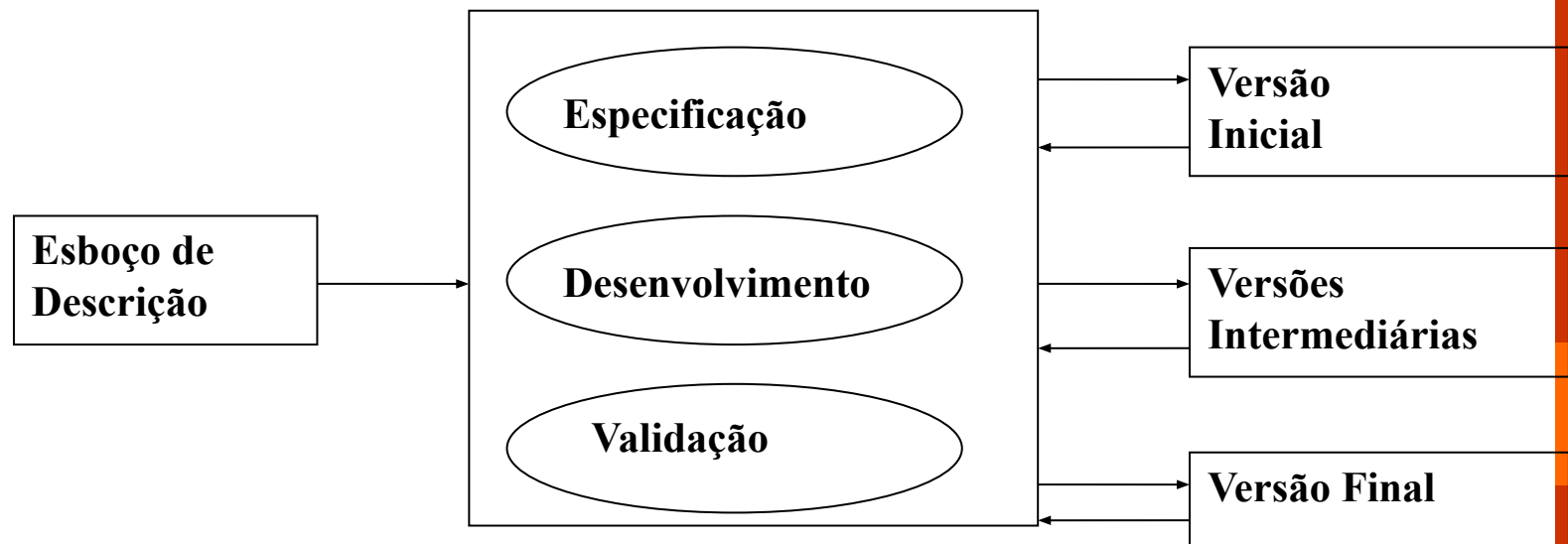
# Modelo Cascata na Prática



# Modelo de Desenvolvimento Evolucionário

- Programação Exploratória
- Prototipagem Descartável

Atividades Concorrentes



# Programação Exploratória

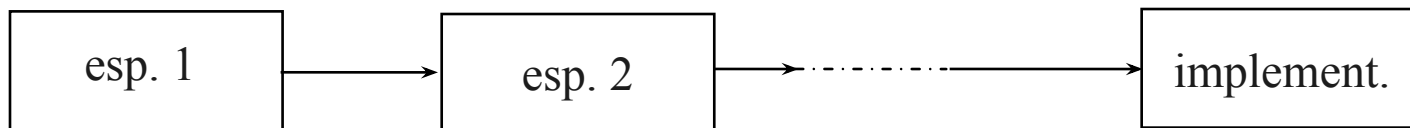
- Idéia geral:
  - Desenvolvimento da primeira versão do sistema o mais rápido possível
  - Modificações sucessivas até que o sistema seja considerado adequado
  - Após o desenvolvimento de cada uma das versões do sistema ele é mostrado aos usuários para comentários
- Adequado para o desenvolvimento de sistemas onde é difícil ou impossível se fazer uma especificação detalhada do sistema
- Principal diferença para os outros modelos é a ausência da noção de programa correto

# Prototipagem Descartável

- Como na programação exploratória, a primeira fase prevê o desenvolvimento de um programa para o usuário experimentar
  - No entanto, o objetivo aqui é estabelecer os requisitos do sistema
  - O software deve ser reimplementado na fase seguinte
- A construção de protótipos com os quais os usuários possam brincar é uma idéia bastante atrativa:
  - Para sistemas grandes e complicados
  - Quando não existe um sistema anterior ou um sistema manual que ajude a especificar os requisitos

# Transformação Formal

- Idéia geral:
  - Uma especificação formal (definição matemática, não ambígua) do software é desenvolvida e posteriormente “transformada” em um programa através de regras que preservam a corretude da especificação



# Modelo de Desenvolvimento Baseado em Reuso

- Baseado no reuso sistemático de componentes existentes ou sistemas COTS (Commercial-off-the-shelf)
- Etapas do processo
  - Especificação dos requisitos
  - Análise de componentes
  - Modificação dos requisitos
  - Projeto de sistema com reuso
  - Desenvolvimento e integração
  - Validação
- Esta abordagem está se tornando mais importante, mas há ainda pouca experiência com ela

# Modelos Iterativos

- Requisitos de sistema SEMPRE evoluem durante curso de um projeto. Assim a iteração do processo sempre faz parte do desenvolvimento de grandes sistemas
- Iterações podem ser aplicadas a quaisquer dos modelos de ciclo de vida
- Duas abordagens (relacionadas)
  - Desenvolvimento espiral
  - Desenvolvimento incremental



# Desenvolvimento Espiral

- Acrescenta aspectos gerenciais ao processo de desenvolvimento de software.
  - análise de riscos em intervalos regulares do processo de desenvolvimento de software
  - planejamento
  - controle
  - tomada de decisão
- O processo é representado como uma espiral em vez de uma sequência de atividades
- Cada volta na espiral representa uma fase no processo
- Não há fases fixas como especificação ou projeto - voltas na espiral são escolhidas dependendo do que é requerido
- Riscos são avaliados explicitamente e resolvidos ao longo do processo

# Desenvolvimento Incremental

- Em vez de entregar o sistema como um todo, o desenvolvimento e a entrega são divididos em incrementos, com cada incremento entregando parte da funcionalidade requerida
- Requisitos dos usuários são priorizados e os requisitos de mais alta prioridade são incluídos nas iterações iniciais
- Uma vez que o desenvolvimento de um incremento é iniciado, os requisitos são "congelados". Embora os requisitos possam continuar a evoluir para incrementos posteriores
- Em certos aspectos é similar à programação exploratória. No entanto, o escopo do sistema deve ser claramente entendido antes de se iniciar o desenvolvimento

# Processo

- Conjunto de atividades
  - bem definidas
  - com responsáveis
  - com artefatos de entrada e saída
  - com dependências entre as mesmas e ordem de execução
  - com modelo de ciclo de vida

# Processo versus Metodologia

- Alguns autores consideram que processos incluem
  - uma metodologia
  - pessoas
  - tecnologia (suporte de ferramentas)
- Outros consideram que uma metodologia é a especialização de um processo com um conjunto de métodos

# Método

- Descrição sistemática de como deve-se realizar uma determinada atividade ou tarefa
- A descrição é normalmente feita através de padrões e guias
- Exemplos: Método para descoberta de atores e casos de uso no RUP.

# Modelo de Processo

- é uma representação de um processo, usualmente envolvendo
  - atividades a serem realizadas
  - agentes que realizam as atividades
  - artefatos (produtos) gerados
  - recursos necessários (consumidos)

# Exemplos de processos

- Processos tradicionais (pesados)
  - RUP, OPEN, Catalysis
- Processos ágeis (leves)
  - XP, Agile modeling, Crystal, pragmatic programming, Internet Speed, Scrum, ...

# Polêmica

- Se a tendência é processos leves, afinal o desenvolvimento de software é
  - Arte+Sociologia+Psicologia+...  
ou
  - Lógica+Modelos+Engenharia+...???
- E todo o esforço de consolidação da Engenharia de Software como uma ciência exata???
- Compromisso
  - *Balancing agility and discipline*



# Na prática....

# O Início de Tudo...



“A intenção do cliente é...”

O Mais Importante Aqui é...

A Idéia  
é  
Viável???

# O Que Devo Fazer Exatamente?

Ou, em outras  
palavras, quais  
são os **requisitos**  
da aplicação?

# Requisitos

- O Que devo fazer?
  - Funcionalidades
- Há restrições sobre as funcionalidades?
  - Limites de tempo, memória, etc.?
- Há restrições mais amplas?
  - Empresa, Governo, etc.?

# O que faço então?

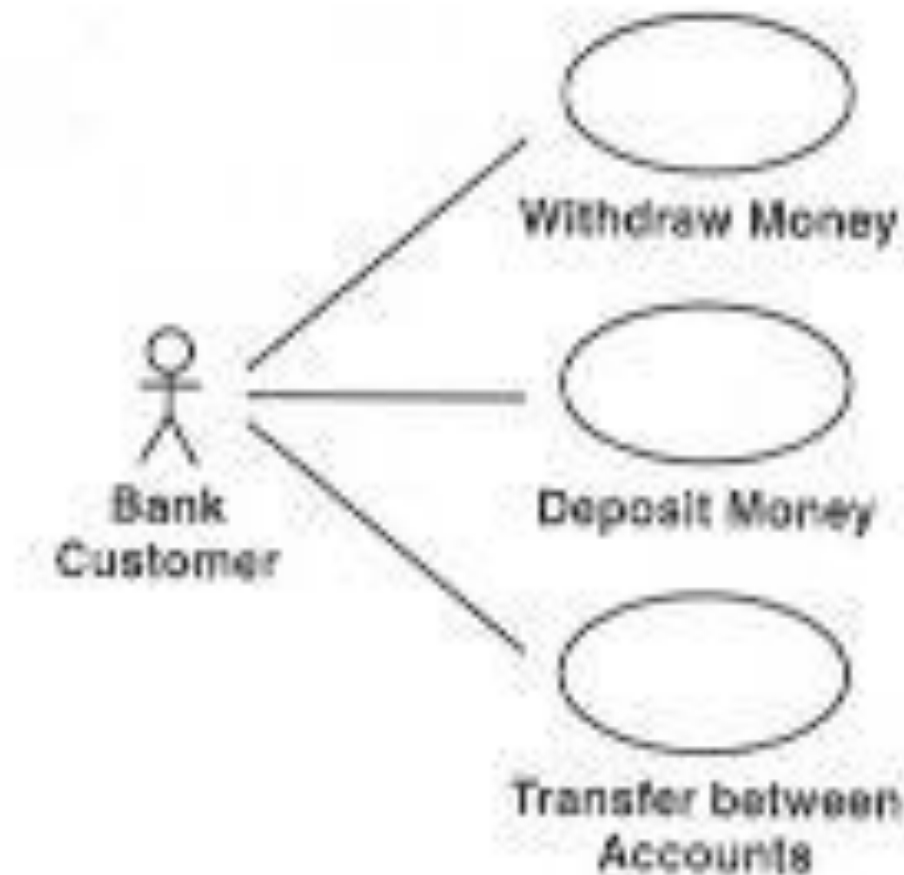


## "O documento de requisitos..."

# Como apresentar ao Cliente?

"O cliente  
não vai ler  
500 páginas  
de  
requisitos!!!"

# Uma Figura Vale Mais Que ..





# Mas Paralelamente ...

"Precisamos saber  
quanto tempo  
levaremos para fazer  
nosso trabalho, quanto  
isso custará e o que  
pode nos atrapalhar...  
Precisamos Planejar!!!"

# Estimando Esforço

- Modelo de casos de uso pode ser usado para calcular estimativa
- Baseia-se em uma série de fatores que determinam a complexidade da aplicação
- Há ferramentas para realizar o cálculo

## Iniciando a Solução...

"Temos que identificar em nossos requisitos, quais são os elementos essenciais para satisfazê-los..."

# Sedimentando a Solução...

"A partir dos elementos essenciais, precisamos definir estratégias para satisfazê-los incluindo suas restrições..."

# Operacionalizando a Solução...

"Com a solução definida, o passo final é operacionalizá-la. Isto é, codificá-la."

# Classe Account...

```
public class Account {  
    private int balance;  
    /*@ invariant balance >= 0 @*/  
    ...  
    void debit(int amount) {  
        /*@ requires amount <= balance @*/  
        /*@ ensures balance = \old(balance) - amount @*/  
    }  
    ...  
}
```

# Funciona???

"Com a implementação feita, podemos então executar os testes!!!"

# Avaliando a qualidade

```
public class AccountTest extends TestCase
{
    void testDebit() {
        Account acc = new Account(10);
        acc.debit(10);
        assertEquals(0, acc.getBalance());
    }
}
```



# Referências

- Básica
  - Sommerville, I. Software Engineering.
- Extra
  - Kruchten, P. The Rational Unified Process: An Introduction. 2nd Ed
  - Booch, G. et al. The Unified Modeling Language User Guide.
  - Pressman, R. Software Engineering.