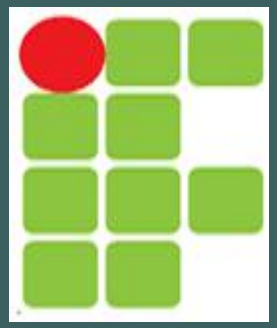


# Modelos de Processo de Software

Aula 02

Métodos e Técnicas em Engenharia de Software

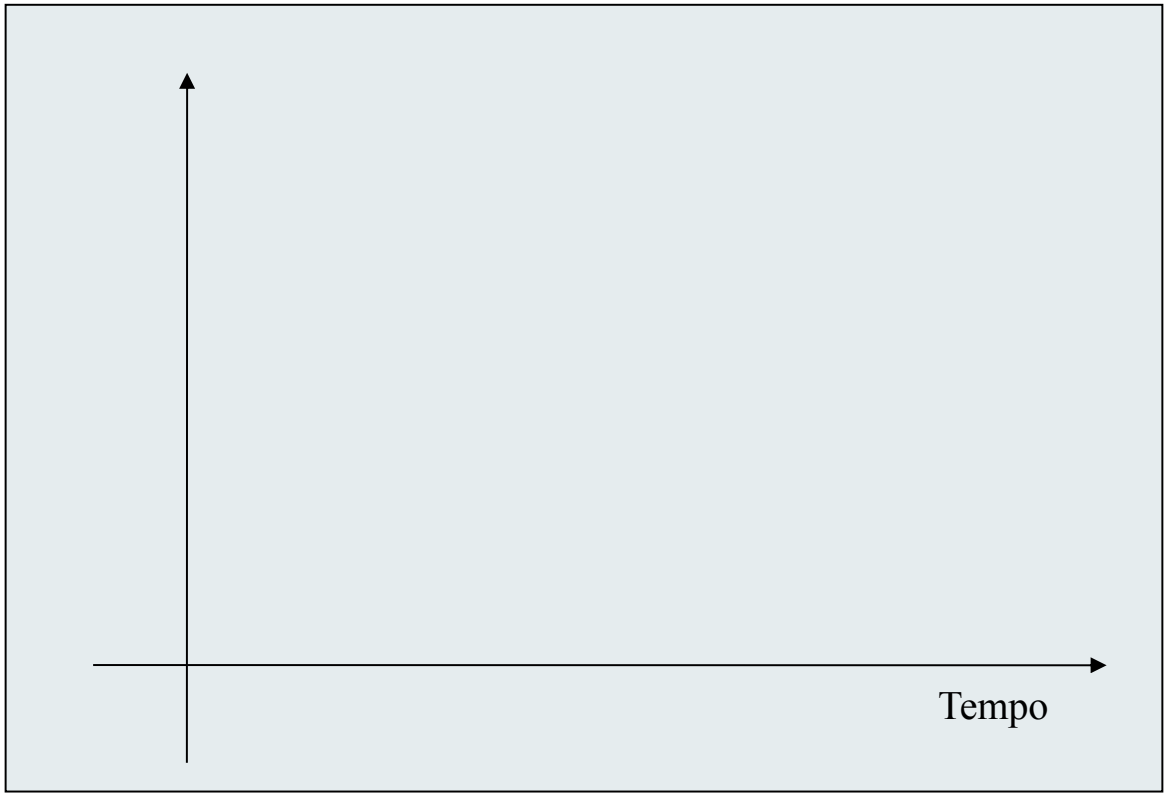
Prof. Thiago

Pós-Graduação em Gestão de Sistemas de Informação

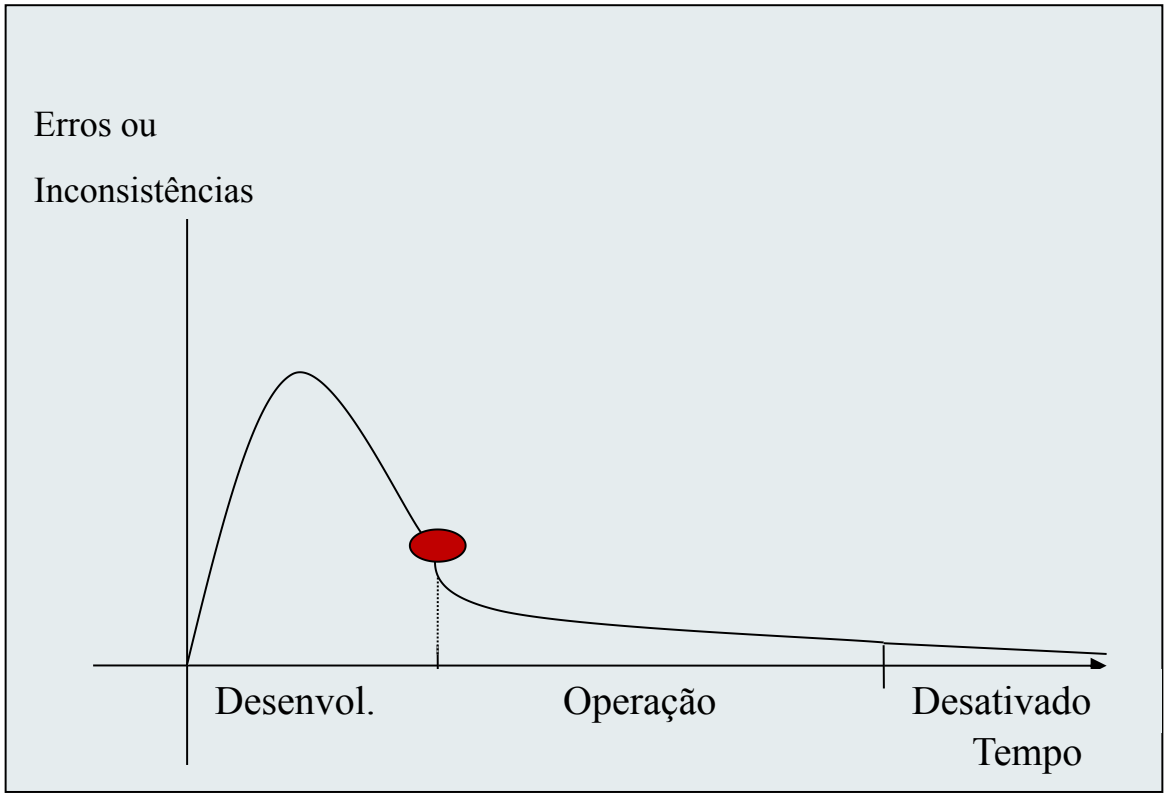
# Objetivos

- Após esta aula, você deverá ser capaz de:
  - Conceituar modelo de processo de software;
  - Compreender o impacto de alterações e correções de erros;
  - Diferenciar os principais modelos de processo de software

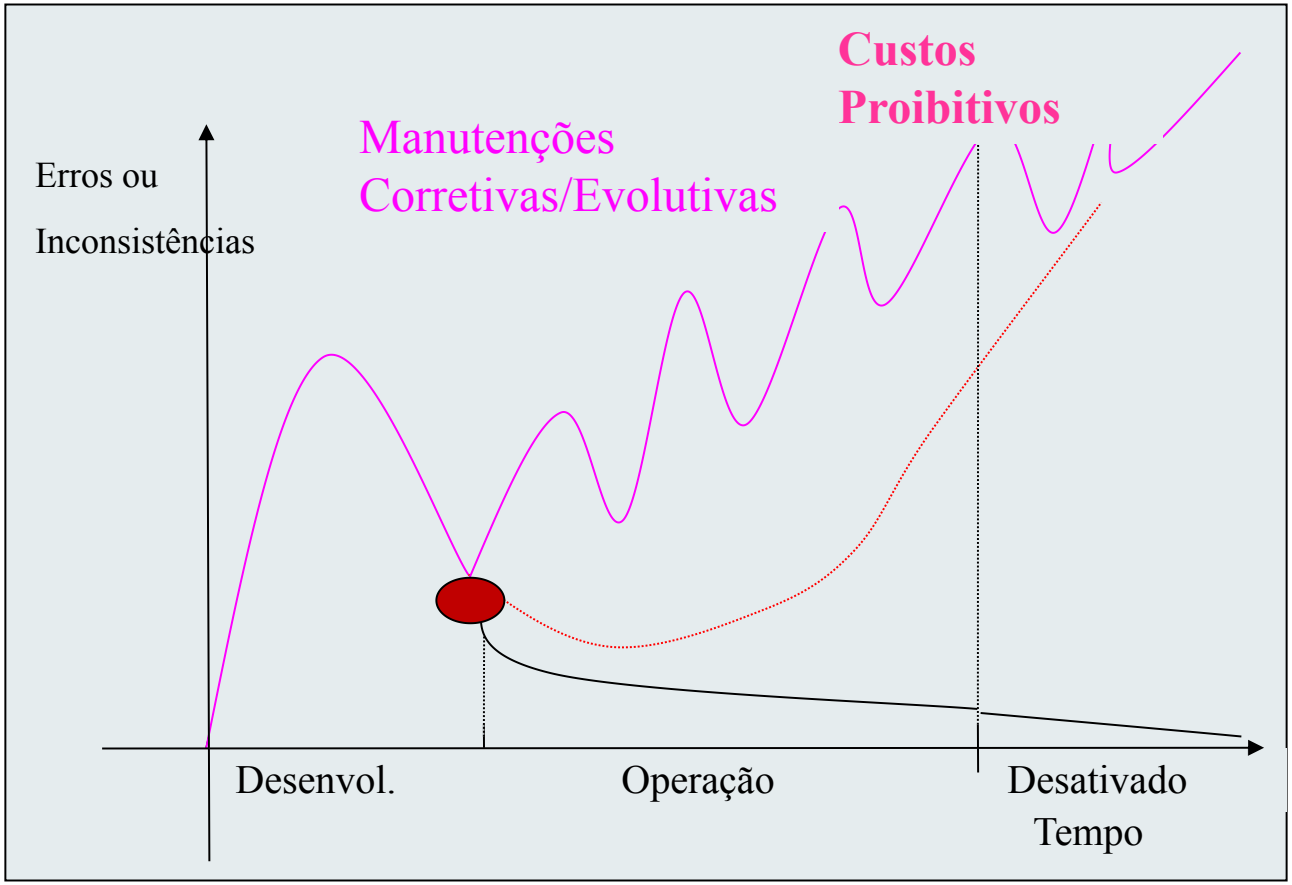
# Ciclo de Vida de um Software



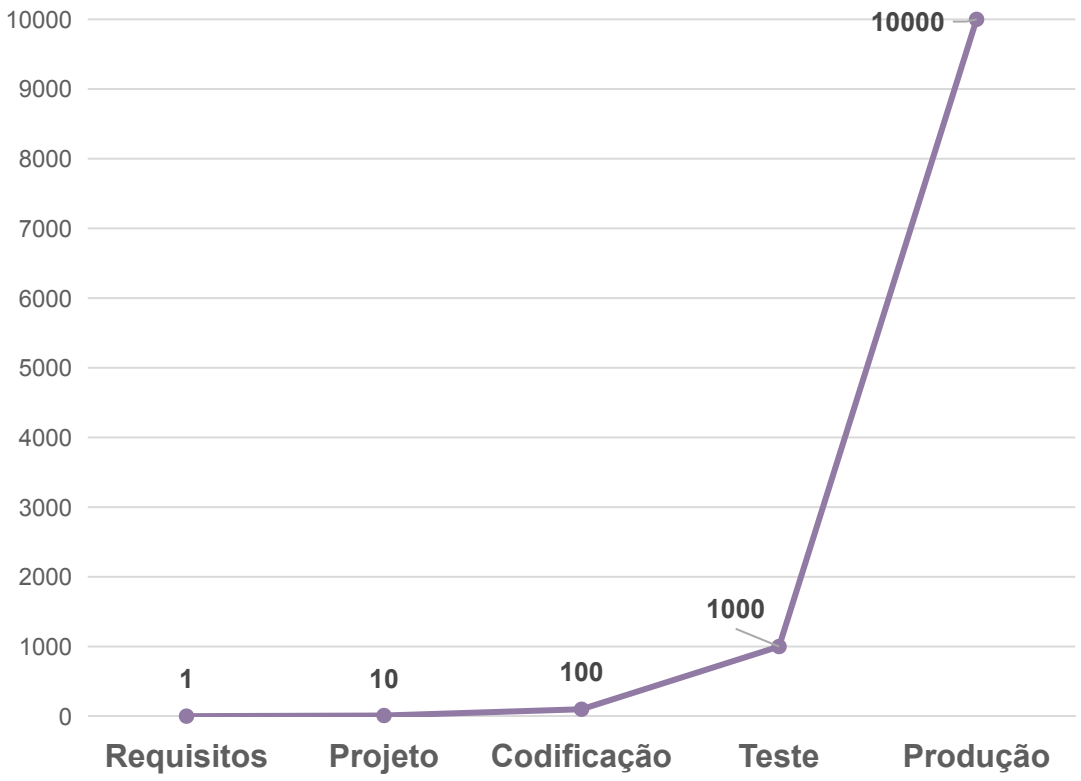
# Ciclo de Vida de um Software



# Ciclo de Vida de um Software



# Custo do erro




- Quanto mais tarde descobrimos o erro, maior o custo.
- Erro não identificado em cada fase, o custo para correção é multiplicado por 10.
- Erros em produção, além de possuirem custo financeiro alto, causam impactos significativos no negócio da empresa e na sua imagem.

# Processo de software

- O **processo de software** é uma sequência estruturada de passos para desenvolver um sistema de software

- A estrutura do processo é abstrata, e visualizada sob algum ponto de vista em particular
  - Muitas vezes, denominada *modelo de processo de software*

# Modelos de processo de software

- Há atividades fundamentais que são comuns a todos os processos de software [Sommerville, 2003]:

1. **Especificação do software**

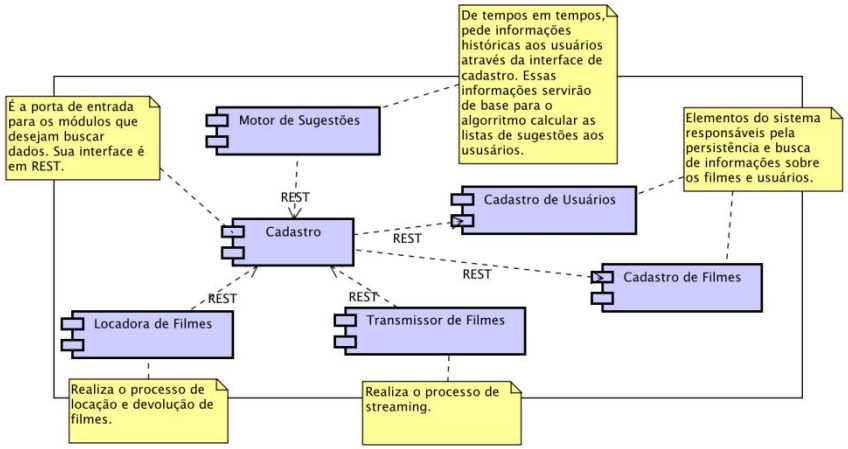
   - Definir a funcionalidade do software e restrições em sua operação e desenvolvimento

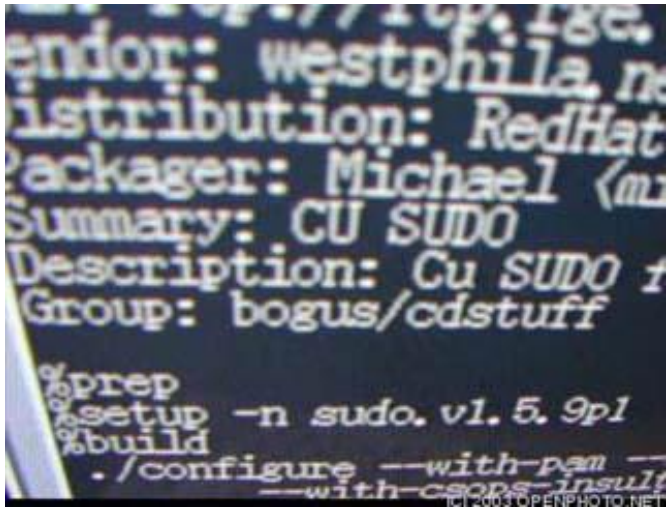
   - Envolve, em geral, as seguintes atividades:
     - Estudo de viabilidade
     - Elicitação e análise de requisitos
     - Especificação de requisitos
     - Validação de requisitos

# Modelos de processo de software

## 2. Projeto e implementação do software

- Especificar o funcionamento do software de acordo com algum modelo de representação, e;
- Produzir o software de modo que ele cumpra a sua especificação



# Modelos de processo de software

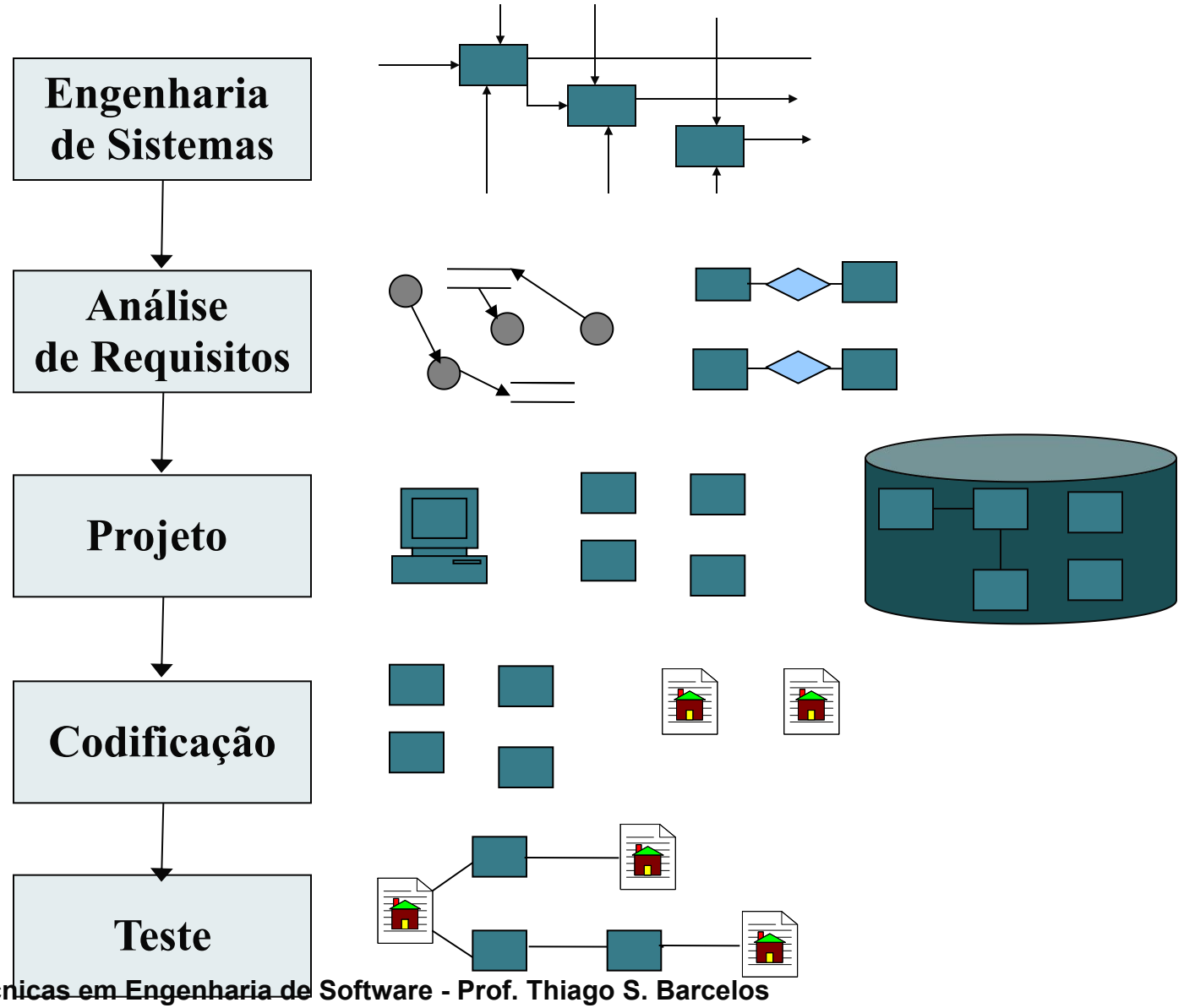
3. **Validação do software**
   - Verificar se o software construído de fato faz o que o cliente deseja

4. **Evolução do software**
   - Alterar o software para que ele atenda a mudanças nas necessidades do cliente

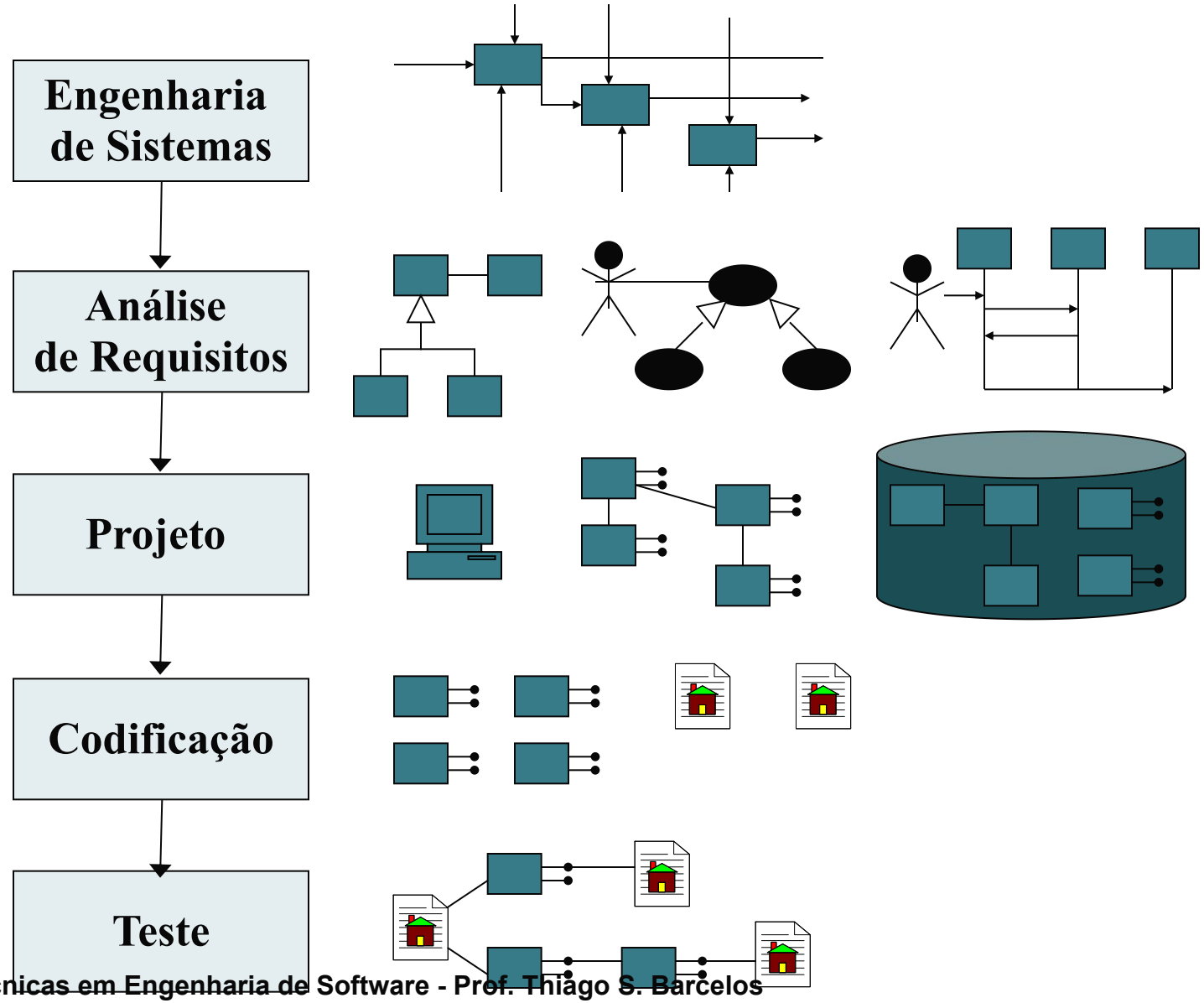
# Modelos de processo de software

**Apesar das diferentes terminologias, as atividades do processo de software são as mesmas. O que muda é só a forma de organizá-las!**

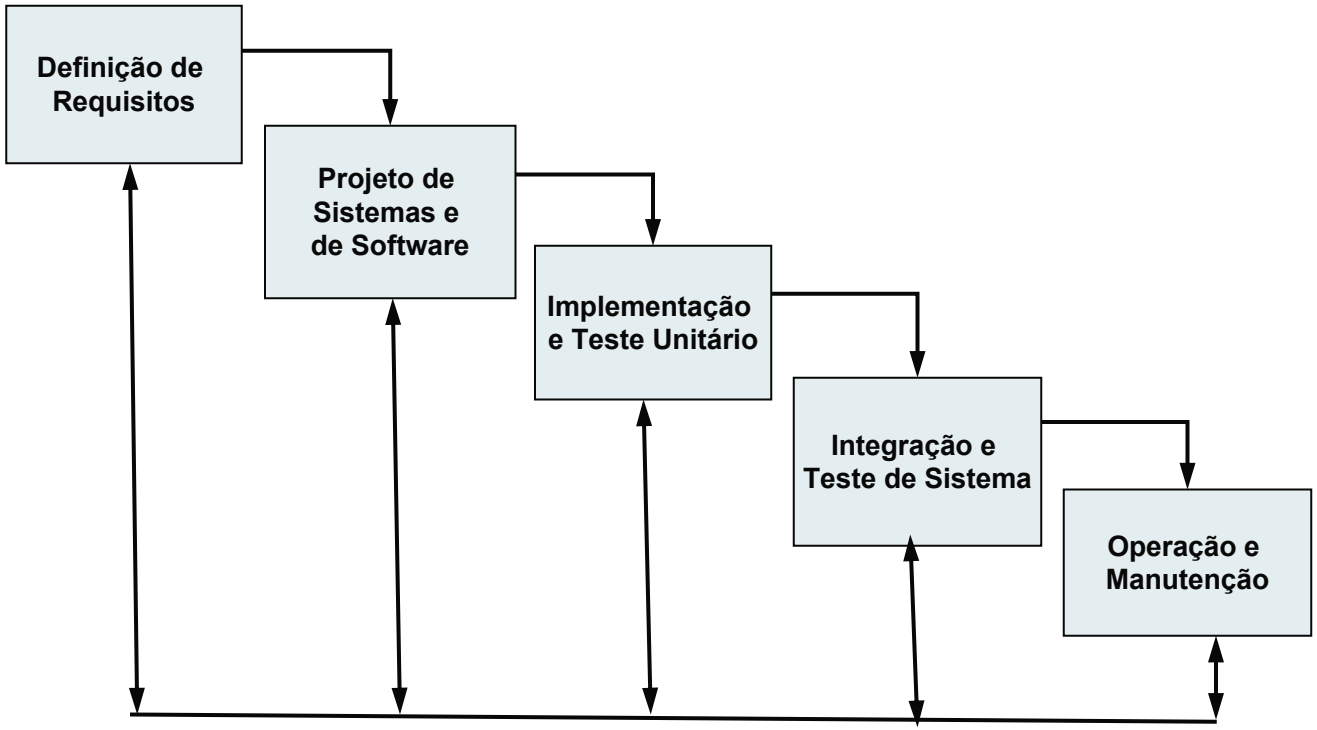
# Paradigma da Análise Estruturada



# Paradigma Orientado a Objetos



# Modelo Cascata



# Modelo Cascata

- **Pontos Fracos**
  - Particionamento inflexível do projeto em fases Distintas.
  - Dificuldade em  responder ou acomodar as mudanças de  requisitos após o processo ter sido iniciado.
  - Só é apropriado quando os requisitos são bem compreendidos.

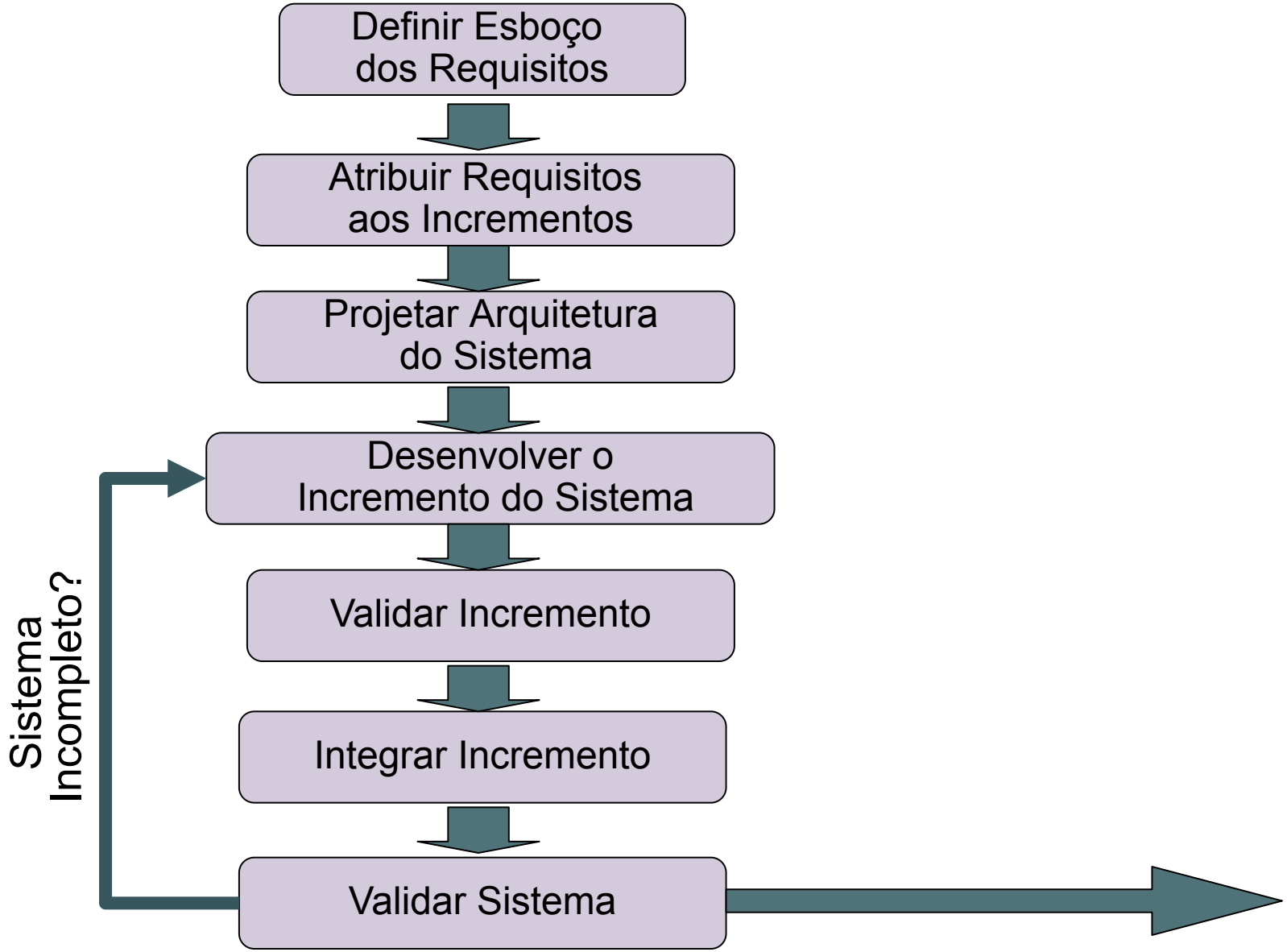
- **Pontos Fortes**
  - Fixa pontos específicos para a entrega de artefatos.
  - É simples e fácil de aplicar, facilitando o planejamento.

# Desenvolvimento Incremental

- Foi sugerida por Mills (Mills *et al.,* 1980) como um meio de reduzir o “retrabalho” no processo de desenvolvimento e de proporcionar  aos clientes algumas oportunidades de adiar descisões sobre seus requisitos detalhados, até que eles tenham alguma experiência com o sistema.

- Neste processo, os clientes identificam, em um esboço, as funções a serem fornecidas pelo sistema, classificando em quais são as mais e menos importantes para eles.

- Em seguida é definida uma série de estágios de entrega, com cada um fornecendo um subconjunto das funcionalidades do sistema. A alocação de funções aos estágios depende da prioridade da função. As funções prioritárias são entregues primeiramente ao cliente.




Fonte: (Sommerville, 2007)

# Desenvolvimento Incremental

- Uma vez identificados os incrementos, os requisitos para as funções a serem entregues no primeiro incremento são definidos em detalhes, e esse incremento é desenvolvido, utilizando-se o processo de desenvolvimento mais apropriado.

- **Durante esse desenvolvimento, outras análises de requisitos para os incrementos seguintes podem ocorrer, mas as mudanças nos requisitos para o incremento atual não são aceitas.**

- Uma vez que um incremento é concluído e entregue, os clientes podem colocá-lo em operação, ou seja, recebem com antecedência parte do sistema.

- Com isso podem utilizar o sistema o que facilita esclarecer seus requisitos para os próximos incrementos.

# Desenvolvimento Incremental

- **Pontos Fortes**
  - Os clientes não precisam esperar até que todo o sistema seja entregue.
  - Pode-se usar os primeiros incrementos como um protótipo e obter a uma experiência que forneça os requisitos para estágios posteriores do sistema.
  - Existe um risco menor de fracasso completo do sistema. Embora possam ser encontrados problemas em alguns incrementos, é provável que alguns incrementos sejam entregues com sucesso ao cliente.
  - Como as funções prioritárias são entregues primeiro e incrementos posteriores são integrados a ela, as funções mais importantes do sistema passam pela maior parte dos testes.

- **Pontos Fracos**
  - Os incrementos devem ser relativamente pequenos (não mais de 20 mil linhas de código). Cada incremento deve produzir alguma funcionalidade do sistema. Assim, pode ser difícil mapear requisitos dentro de incrementos de tamanho correto.
  - Funcionalidades básicas que são utilizadas por diversas partes do sistema são difíceis de serem identificadas.

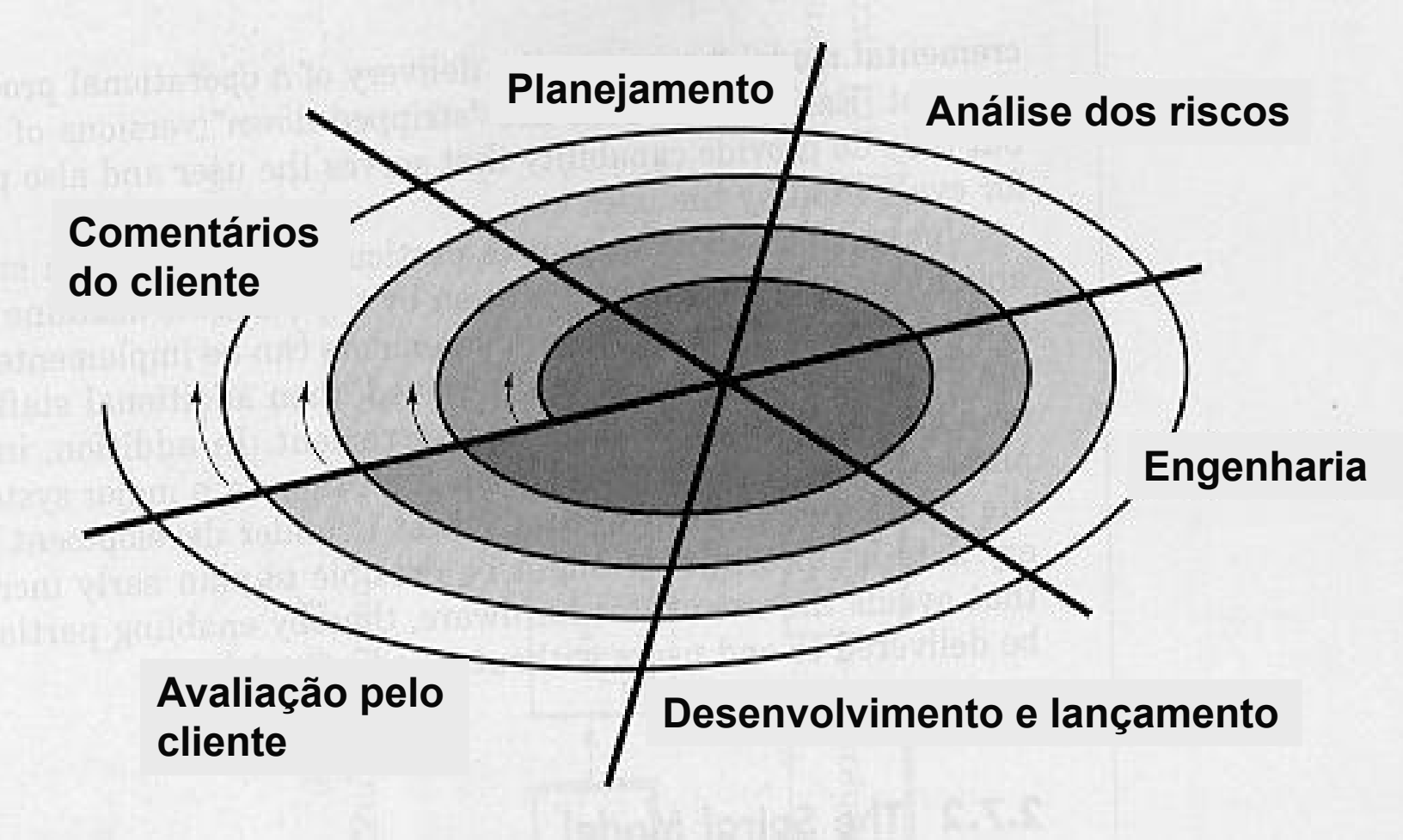
# Desenvolvimento incremental

- Esse paradigma deu origem a vários modelos de processo:
  - **Espiral**
  - **RUP**
  - **MSF**
  - **Metodologias ágeis**

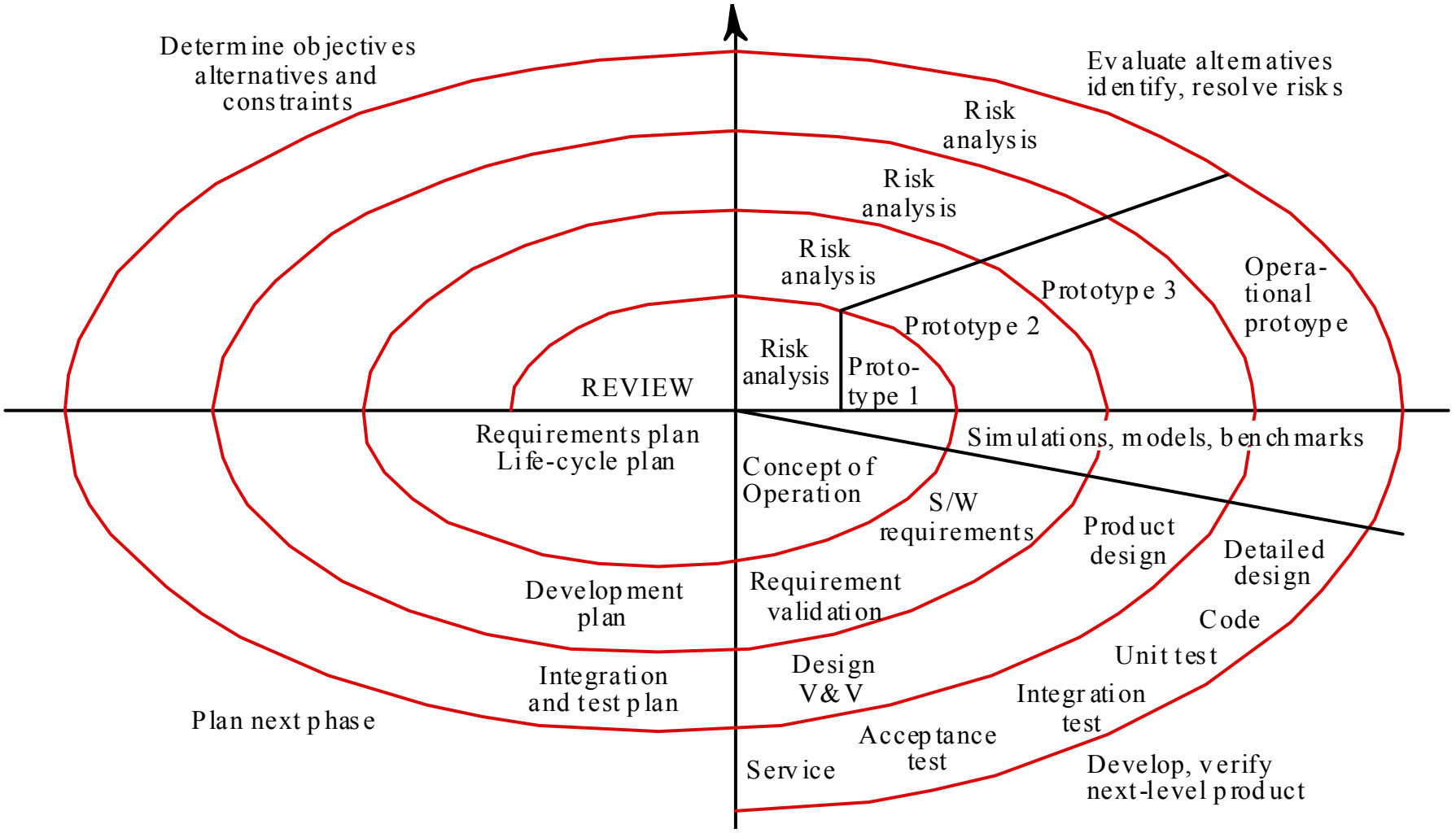
# Desenvolvimento Incremental - Espiral

- Proposto por Boehm (Boehm,1988)
- Representa o processo de software não como uma sequência de atividades com algum retorno entre alguma atividade e outra
- Representa o processo de software com uma espiral, cada loop da espiral representa uma fase do processo de software
- A principal diferença desse modelo com os outros modelos do processo de software é o reconhecimento explícito do risco no processo de desenvolvimento

# Modelo espiral



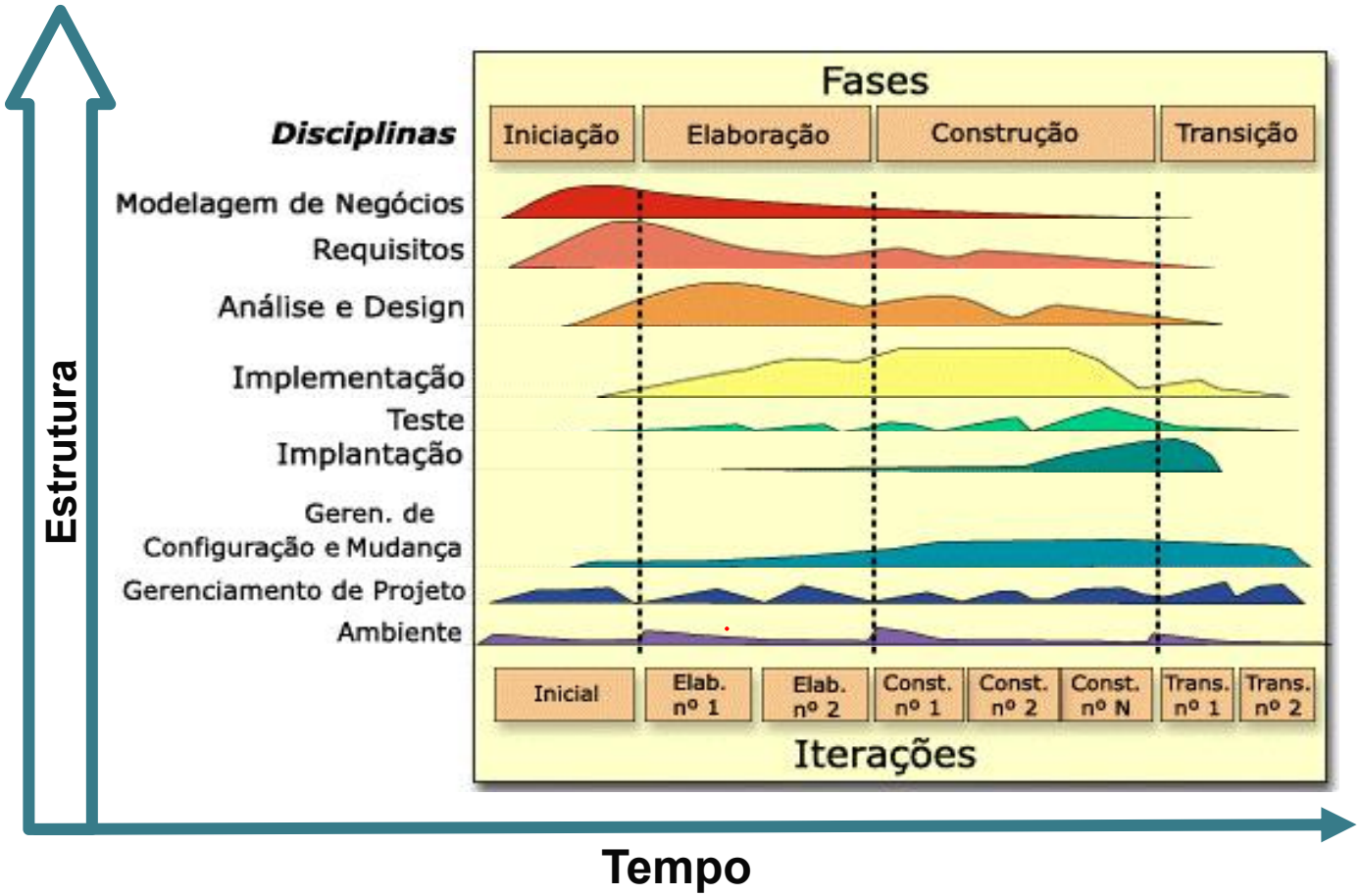
# Desenvolvimento incremental - espiral



# Desenvolvimento Incremental - Espiral

- **Pontos Fortes**
  - Modelo evolutivo possibilita uma maior integração entre as fases e facilita a depuração e a manutenção do sistema.
  - Permite que o projetista e cliente possa entender e reagir aos riscos em cada etapa evolutiva.
- **Pontos Fracos**
  - Avaliação dos riscos exige muita experiência.

# Desenvolvimento Incremental – RUP



# Desenvolvimento Incremental - RUP

- Visão Geral

  - Ciclo de Desenvolvimento de Software.
  - Abordagem baseada em disciplinas para atribuir tarefas e responsabilidades.
  - Composto por fases, iterações e marcos.
  - Baseado em casos de uso.
  - Garantir a produção de software de alta qualidade.
  - Atender as necessidades dos usuários.
  - Dentro de um cronograma e de um orçamento previsíveis.

# Desenvolvimento Incremental - RUP

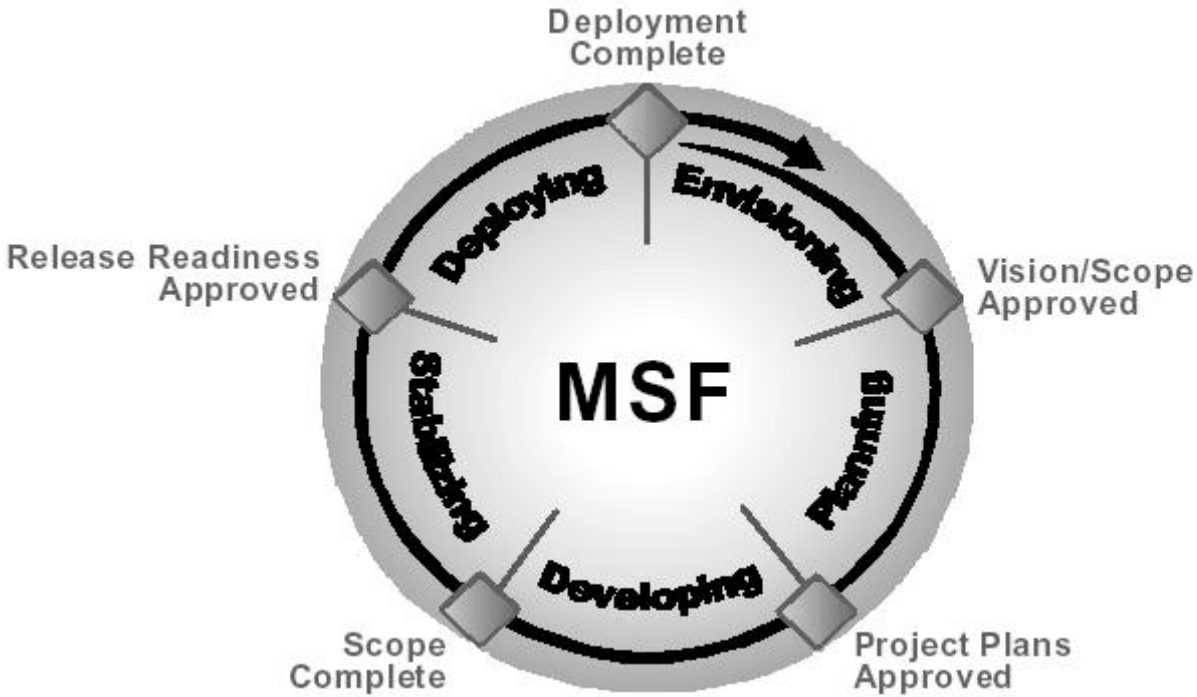
- **Pontos Fortes**
  - Prover a diminuição dos riscos
    - Ataque os riscos antes que eles o ataquem.
    - O RUP ataca os riscos na primeiras iterações.
  - Obter visão clara do andamento do projeto
    - Marcos intermediários.
    - Aumentar o feedback com os clientes.
    - Entregas parciais.

- **Pontos Fracos**
  - Avaliação dos riscos exige muita experiência.
  - Complexidade do Processo.

# MSF - Microsoft Solutions Framework

- Breve Histórico
  - O MSF foi criado em 1994 através da coleta de um conjunto de boas utilizadas pelos desenvolvedores de produtos Microsoft em seu projetos.
  - Se baseia nas práticas recomendadas de desenvolvimento de produto da Microsoft e das organizações de TI.
  - A Microsoft utiliza pesquisas contínuas e comentários dos clientes para aprimorar os modelos, os princípios e as práticas recomendadas para o MSF.

# MSF Process Model

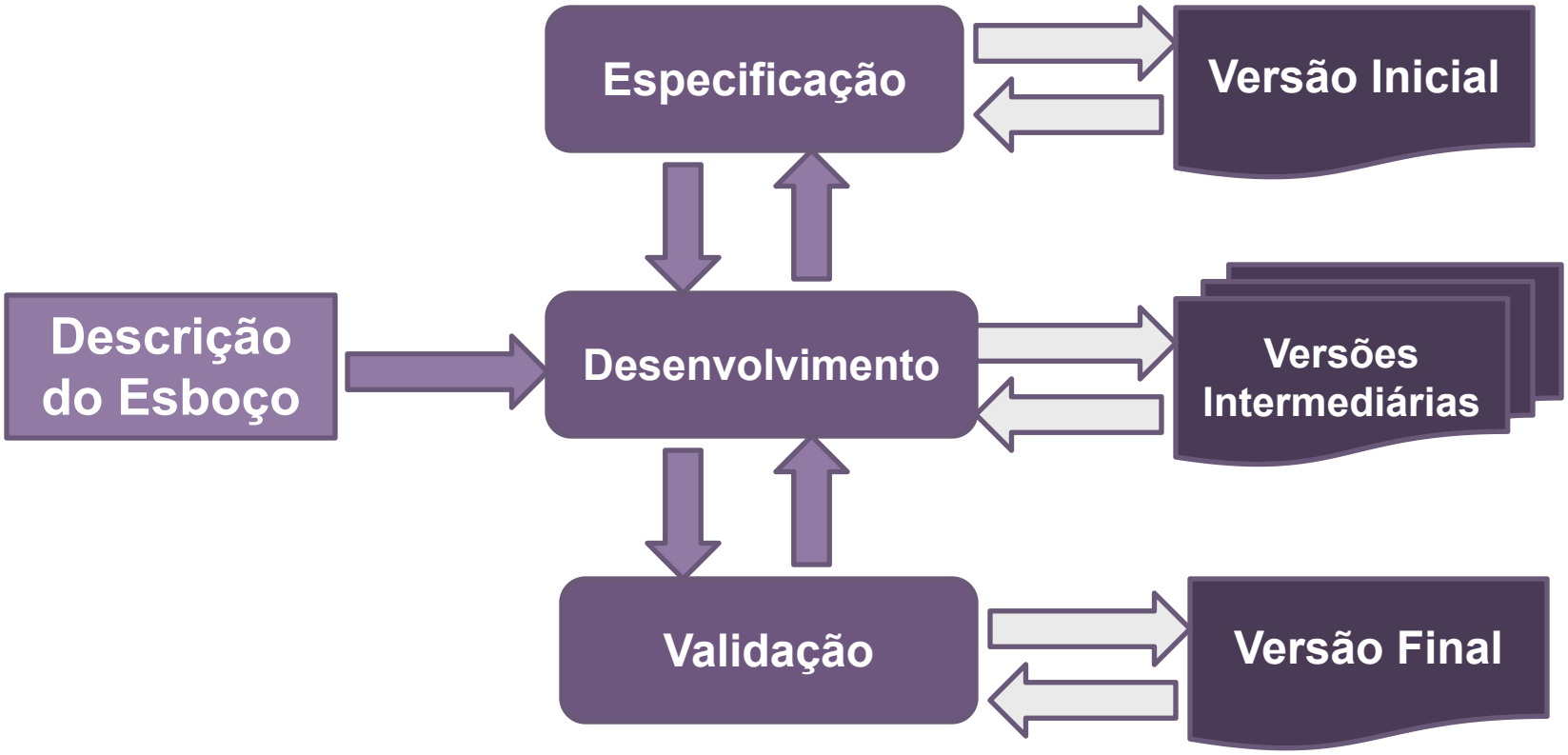



Fonte: Material de referência Microsoft, Whitepaper: Microsoft Solutions Framework

# MSF Process Model

| Marco | Principal Responsável |
|---|---|
| Aprovação da Visão/Escopo (Vision Scope Approved ) | Gerência de Produto |
| Aprovação dos Planos de Projeto (Project Plans Approved) | Gerência de Programa |
| Finalização do Escopo (Scope Complete) | Desenvolvimento e Experiência do Usuário |
| Aprovação de *Release* (Release Readness Aproved) | Teste e Gerência de *Release* |
| Finalização da Implantação (Deployment Complete) | Gerência de *Release* |

# Desenvolvimento evolucionário



# Desenvolvimento evolucionário

- Visão Geral
  - Base: idéia de desenvolver uma implementação inicial, expor o resultado ao usuário e fazer o aprimoramento por meio de muitas versões até que um sistema adequado tenha sido desenvolvido
  - As atividades de especificação, desenvolvimento e validação são feitas concorrentemente, com um rápido feedback entre essas atividades
  - Baseia-se na atividade de **prototipação**

# Desenvolvimento evolucionário

- Possui dois tipos:
  - **Desenvolvimento Exploratório**
    - Trabalhar com o cliente a fim de explorar os seus requisitos e entregar um sistema final
    - O sistema começa com as partes compreendidas e vai evoluindo com o acréscimo de novas características
    - Também chamado de *prototipação evolucionária*
  - **Fazer Protótipos Descartáveis**
    - Objetivo é compreender os requisitos do sistema junto ao cliente utilizando o protótipo para desenvolver experimentos com partes dos requisitos que estejam mal compreendidos

# Desenvolvimento evolucionário

- **Pontos Fracos**
  - Dificuldade de medir o progresso do projeto
  - Deficiência de arquitetura

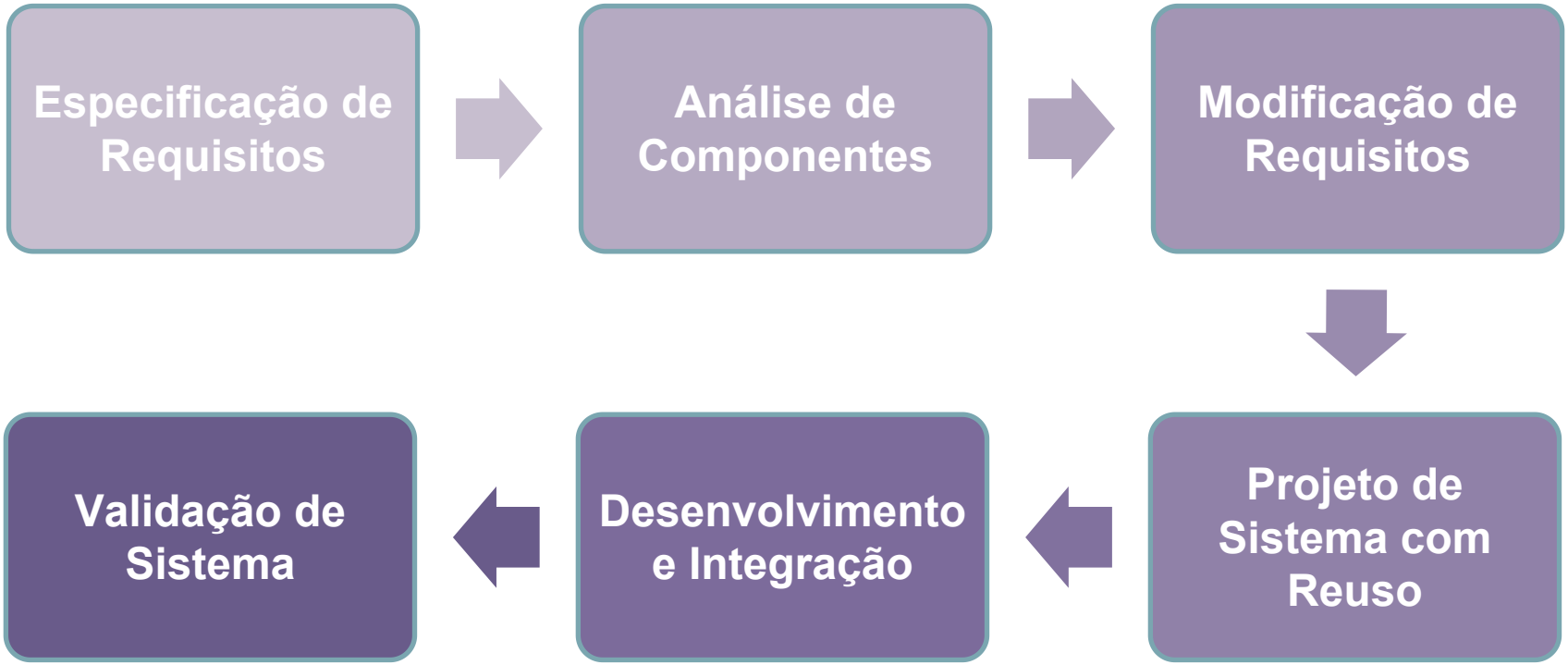
- **Pontos Fortes**
  - Adequado para sistemas de pequeno e médio porte
  - Grande interação com os clientes levando a um melhor entendimento das necessidades

# Desenvolvimento orientado a reuso

## Visão Geral

- Utilizado em situações onde se conta com uma ampla base de componentes de software reutilizável
- Componentes são acessados através de uma infra-estrutura de integração
- Estágios de especificação de requisitos e de validação são iguais aos demais modelos de desenvolvimento, porém os estágios intermediários são diferentes:

# Desenvolvimento orientado a reuso



# Desenvolvimento orientado a reuso

- **Análise de Componentes**
  - Baseado nos requisitos é efetuada uma busca sobre a base de componentes para implementá-los.

- **Modificação dos Requisitos**
  - Trata-se da análise dos requisitos para adequá-los aos componentes selecionados, caso não seja possível modificá-los, deve ser efetuada uma nova busca de componentes.

- **Projeto de Sistemas com reuso**
  - Nesta fase a estrutura do sistema é projetada ou infra-estrutura existente é reutilizada. Os projetistas levam em conta os componentes que são reutilizados e organizam a infra-estrutura para lidar com esse aspecto.

- **Desenvolvimento e Integração**
  - O software que não puder ser reutilizado será desenvolvido e os componentes que serão reutilizados integrados, a fim de criar um sistema.

# Desenvolvimento orientado a reuso

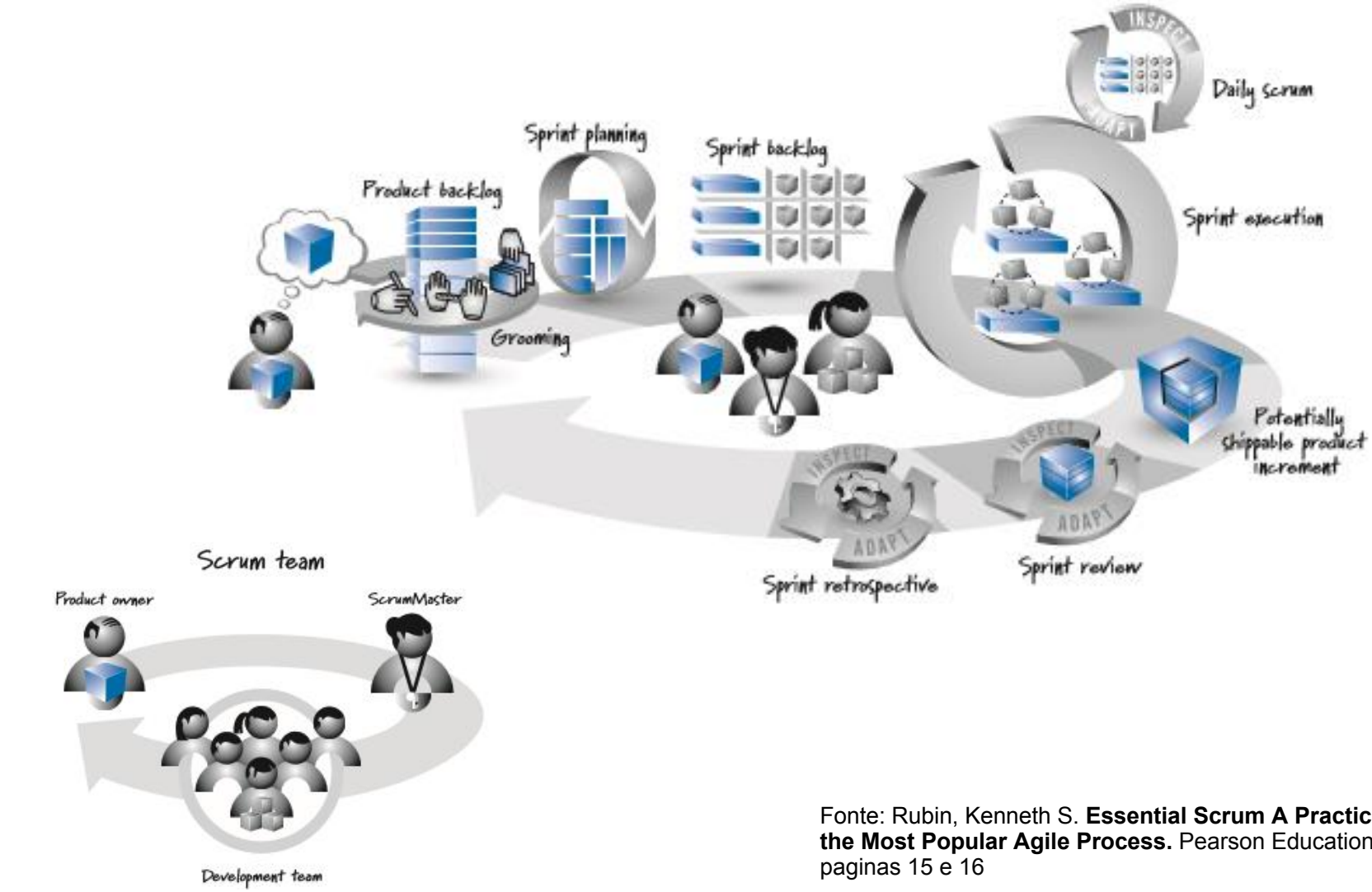
- **Pontos Fracos**
  - O sistema pode não atender as reais necessidades do usuário.
  - Forte controle sobre a evolução dos componentes.
  - Baixo controle da evolução do software, principalmente se os componentes utilizados são comerciais de prateleiras.
  - Generalização de Componentes tem alto custo, deve ser projetada com cautela.

- **Pontos Fortes**
  - Reduz a quantidade de software produzido.
  - Reduz riscos e custos.
  - Entrega mais rápida de software.

# Metodologias Ágeis

- **Princípios**
  - Indivíduos e interações: mais importantes que processos e ferramentas.
  - Valiosos no tratamento de projetos parcialmente definidos.
  - Resultado incremental aumenta ROI (return on investiment)
    - Stakeholders podem visualizar/utilizar os resultados.
    - Tem maior poder de escolha dos caminhos do projeto.
  - Decisões baseadas em fatos
    - Melhor do que qualquer previsão
  - Decisão distribuída entre os envolvidos no projeto (não apenas o gerente).

# Scrum




Fonte: Rubin, Kenneth S. **Essential Scrum A Practical Guide to the Most Popular Agile Process.** Pearson Education, 2013. paginas 15 e 16

# Conclusão

- Frederick P. Brooks, Jr.
- Gerente de projeto do IBM OS/360
- Ganhador do Prêmio Turing, 1999
- Fundador do Departamento de Ciência da Computação da University of North Carolina at Chapel Hill

# Conclusão

- Software é como um lobisomem: *parece* normal até que a lua apareça e o transforme em um monstro!

- Orçamentos estourados
- Bugs de software
- Prazos não cumpridos

- Então, queremos uma bala-de-prata que mate o monstro, mas…

# Não há bala-de-prata!

**“ (...) nenhuma técnica de Engenharia de Software vai promover um incremento de uma ordem de magnitude (10x) na produtividade do desenvolvimento nos próximos 10 anos.”**

"No Silver Bullet - Essence and Accidents of Software Engineering", Brooks, F. P., *IEEE Computer* 20, 4 (April 1987), pp. 10-19.

# Conclusão

- Fred Brooks separa os problemas do desenvolvimento de software em duas categorias:

## 1. Problemas acidentais

- Expressividade das linguagens de programação
- Velocidade de processamento
- Integração (ou não) de ferramentas de codificação, compilação e teste

# Conclusão

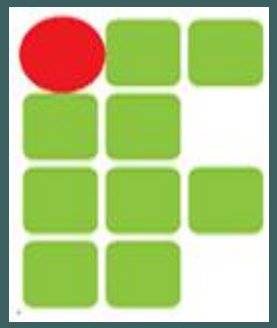
## 2. Problemas essenciais

- Software é complexo: a quantidade de estados e interações cresce de forma exponencial quando o software cresce
- Software deve se adaptar a restrições arbitrárias impostas após sua criação (restrições legais, novas regras de negócio…)
- Software é um produto *lógico*, ou seja, não está sujeito a restrições do mundo físico
- O efeito de mudanças no software é invisível e quase imprevisível

# Conclusão

- Se os problemas acidentais fossem 9/10 do problema, ferramentas e técnicas teriam reduzido esse problema a 0, multiplicando por 10 a produtividade

**Discussão: Qual é realmente o problema?**



# Obrigado!

Profa. Alexandra Aparecida de Souza

Prof. Thiago Schumacher Barcelos