Processo de Desenvolvimento de Software

Karin Becker

Instituto de Informática - UFRGS

Processo de Desenvolvimento de SW

- Especificar artefatos que devem ser produzidos
- Indicar quem deve fazer, o que deve ser feito e quando, e se possível, como fazer
- Especificar o ciclo de vida de um sistema
  - da concepção à entrega e manutenção

FAQ: O que é processo de software?

- É um conjunto de atividades cuja meta é o desenvolvimento ou evolução de software.
- As atividades genéricas em todos os processos de software são:
  - **Especificação** – o que o sistema deve fazer e suas restrições de desenvolvimento.
  - **Desenvolvimento** – produção do sistema de software.
  - **Validação&verificação** – o software está correto e é o que o cliente deseja.
  - **Evolução** – mudança do software em resposta às demandas de mudança

Processo de software

- Atividades genéricas
  - Fazem parte de qualquer processo de software
  - Atividades complexas, subdivididas em várias outras atividades
  - O processo determina o refinamento em sub-atividades, e a forma e ordem de execução
- Atividades de apoio
  - Gerência de configuração e de mudanças
  - Gerência de projeto
  - Documentação
  - Gerência de ambientes
  - etc

Especificação: 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?

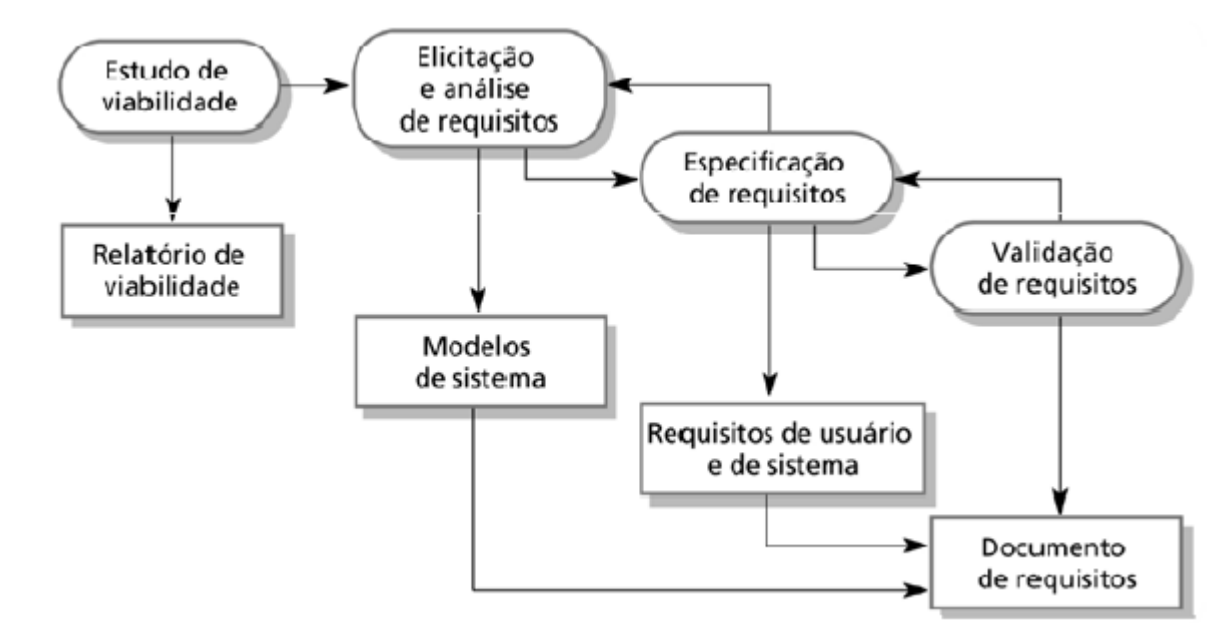
O que devo fazer então?

"documentar" requisitos ...

Especificação: O QUÊ?

- Especifica quais serviços são necessários e as restrições que têm impacto no desenvolvimento e operação do sistema
  - Estudo de Viabilidade
    - Necessidades podem ser atendidas pela TI
    - Viável do ponto de vista comercial, custo, tempo, etc
    - Definição do escopo
    - Rápido, barato e superficial: decisão de prosseguir ou não
  - Análise de Requisitos
    - Observação de sistemas existentes, conversas com potenciais usuários, análise de processos, etc
    - Pode envolver protótipos, provas de conceitos, modelos de sistema, etc
  - Especificação de Requisitos
    - Tradução das informações coletadas na fase de eliciação
    - Requisitos de negócio (abstrato)
    - Requisitos de Sistema (Funcionais, Não-Funcionais)
  - Validação de Requisitos

Especificação

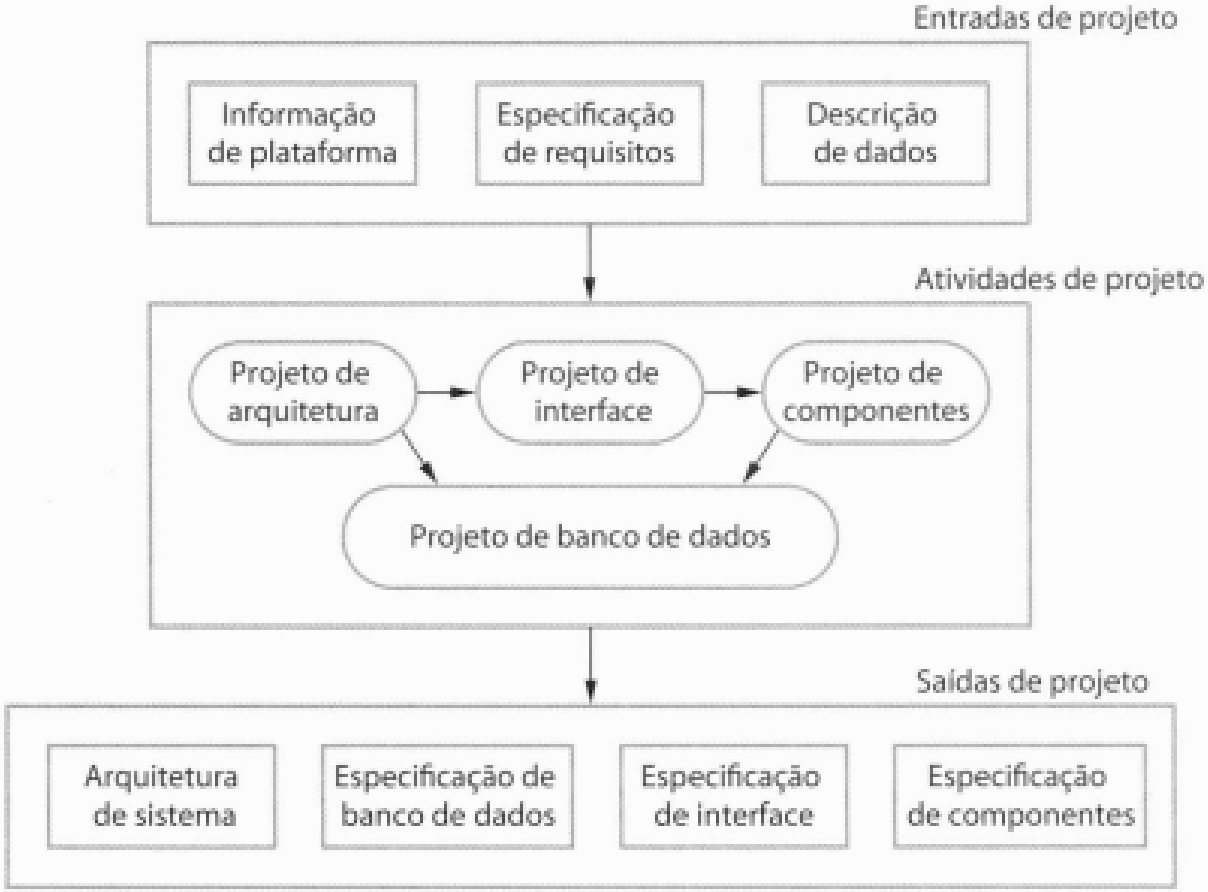


Sommerville

Desenvolvimento: COMO?

- Conversão da especificação de sistema em um sistema executável
  - Projeto de software: conceber uma estrutura de software que atenda aos requisitos especificados
    - Pode requerer refinamento da especificação
    - Requisitos funcionais e não funcionais
    - Atributos de qualidade
  - Implementação: Transformar essa estrutura em um programa executável
- As atividades de projeto e implementação são fortemente relacionadas e frequentemente são intercaladas

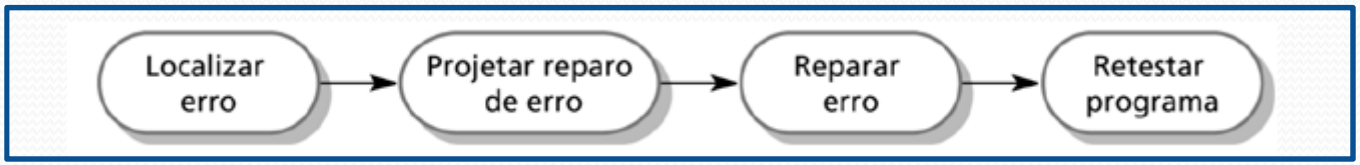
Projeto



Sommerville

Programação e Debugging

- Tradução de um projeto em programas (mapeamento para uma linguagem de programação)
- Programação é frequentemente uma atividade pessoal
  - Estilos de codificação
  - Melhores práticas
  - Padrões
- Programadores desenvolvem testes para descobrir possível falhas em seus programas, e remover estas falhas (debugging)



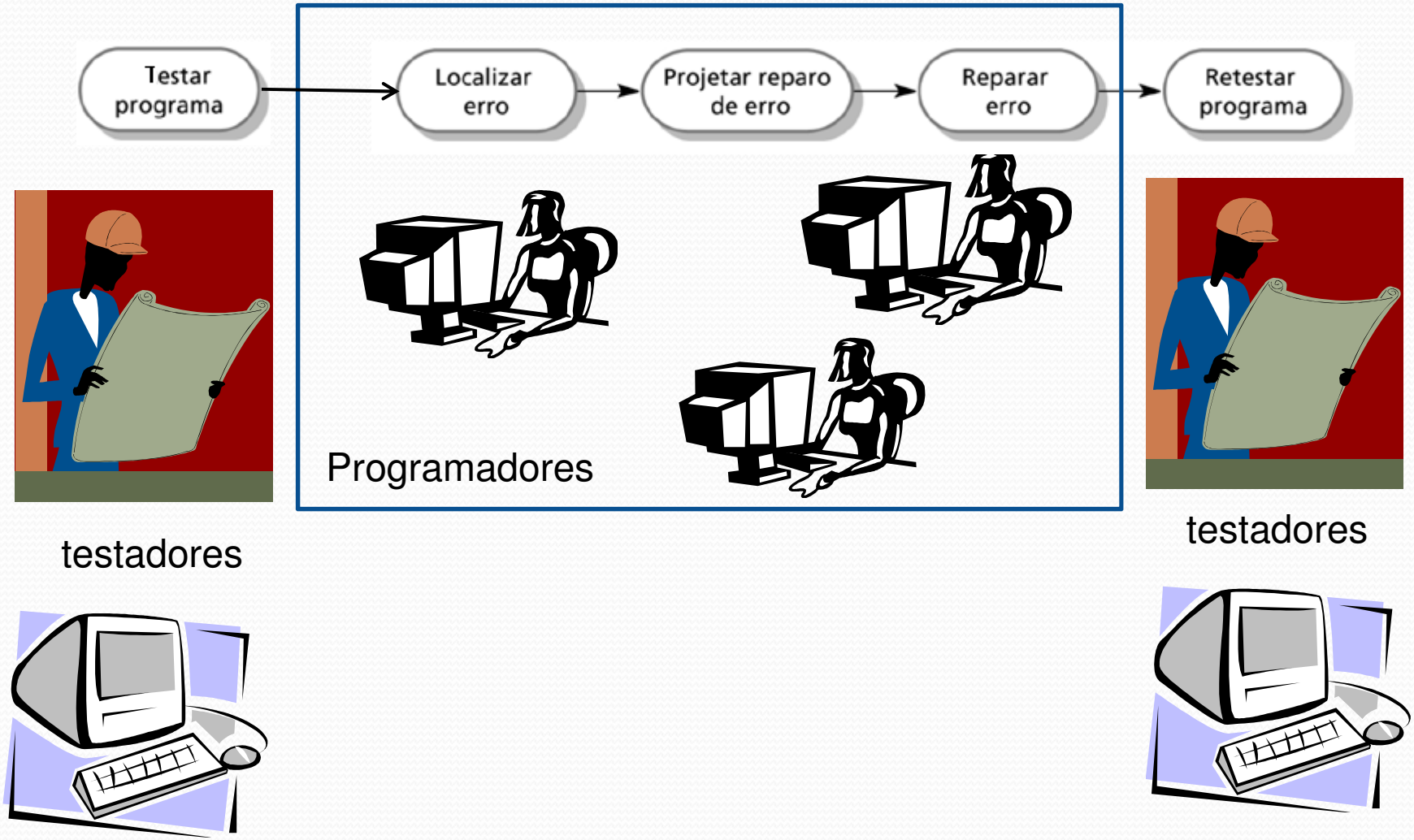
Programador

Sommerville

Verificação e Validação (V&V)

- mostrar que um sistema está em conformidade com a sua especificação e que atende às necessidades do cliente
- Envolve processos de inspeção, revisão e verificação em todas atividades do processo
- Maior concentração após a implementação, quando envolve a execução do sistema com casos de teste que são derivados da especificação, usando dados similares aos reais
  - tendência que o desenvolvimento de casos de teste faça parte do processo de refinamento de requisitos e guie o projeto/implementação
- Diferentes níveis de teste

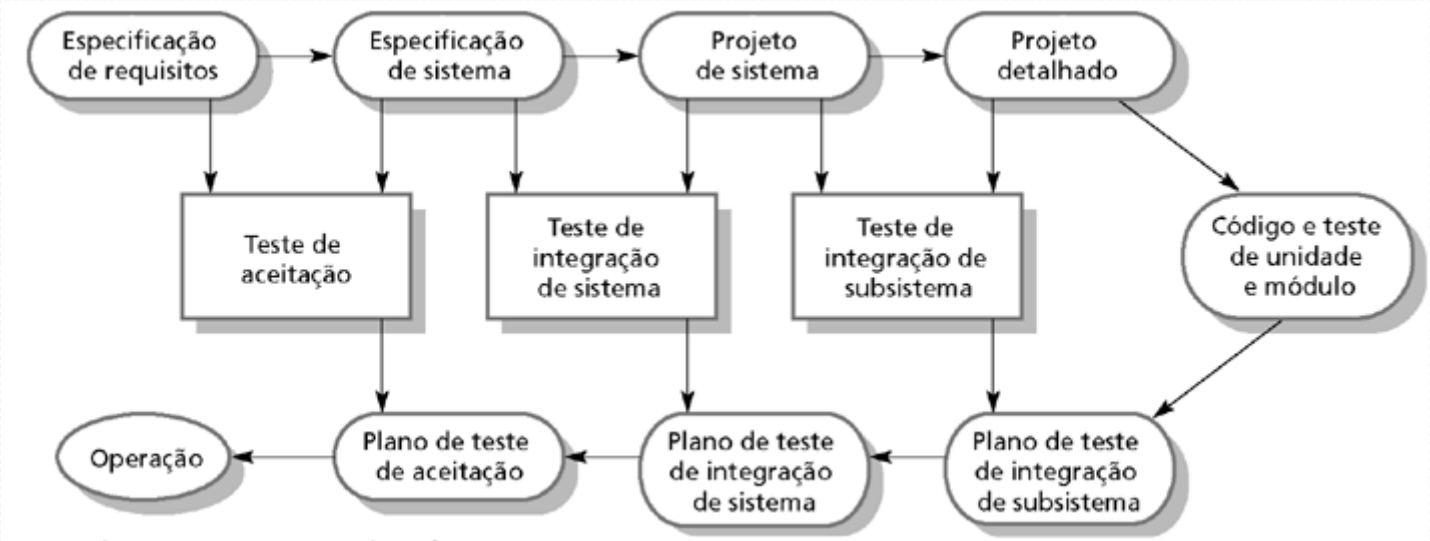
Teste


Testar programa
Localizar erro
Projetar reparo de erro
Reparar erro
Retestar programa
Programadores
testadores
testadores

Níveis de Teste

- Teste de unidade (Componente, desenvolvimento)
  - Componentes individuais são testados independentemente
  - Componentes podem ser funções, abstrações (ex: classes) ou agrupamentos coerentes de entidades
- Teste de Sistema (Teste de Integração)
  - Teste do sistema como um todo
  - Propriedades funcionais, não funcionais (e.g. Performance, usabilidade, etc)
- Teste de Aceitação
  - Testar com os dados do cliente para verificar se o sistema atende suas necessidades

Fases de Teste no processo de SW



Sommerville

Evolução

- SW é flexível por natureza e por isto, pode mudar
  - Por exemplo, um requisito pode ser alterado devido a uma mudança no panorama econômico, ou...
  - ....porque o HW ficou mais barato ou...
  - ....porque o concorrente lançou um produto inovador ou....
  - .... porque o dono teve uma idéia brilhante ...
- É imprevisível
- Manutenção
  - Costuma representar até 70% do ciclo de vida do sistema
  - Costuma ser vista como uma atividade "desinteressante" ("castigo")

Causas da Manutenção

- Causas
  - Correção: defeitos descobertos pelo usuário
  - Adaptação: acomodação de mudanças no ambiente externo
    - Infraestrutura
    - Legislação
    - Processos de negócio
  - Melhoria
    - Adição de funcionalidades que vão melhorar o software
    - Melhor interface, mais funções, mais processos, etc
  - Prevenção
    - Reengenharia
    - Torná-lo mais robusto a mudanças
- A tendência dos métodos de desenvolvimento incrementais é que não haja distinção entre desenvolvimento/manutenção

Modelo de Processo

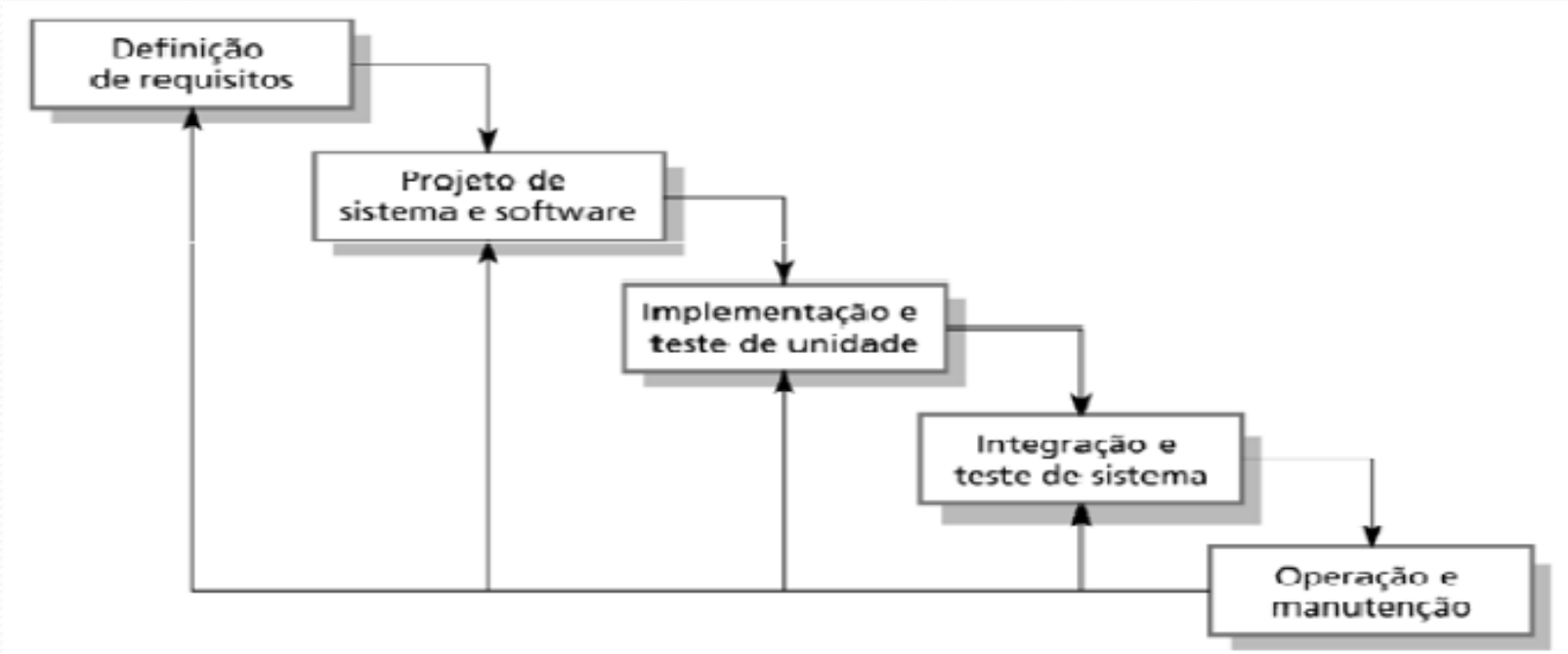
- Um modelo de processo de software é uma representação abstrata do processo. Ele apresenta a descrição de um processo a partir de uma perspectiva particular.
- Modelos mais comuns*
  - Linear
  - Prototipação
  - Formal
  - Desenvolvimento Incremental e Iterativo
  - Baseado em Componentes

*Observação: não há consenso sobre a classificação dos modelos

Modelo Linear ("Waterfall" - Cascata)

- Histórico: NATO 1968
- Pressupõe planejamento cuidadoso
- Divide processo em fases, ordenados linearmente
- os resultados de uma fase tornam-se entradas da próxima
  - qualquer ordenamento diferente resulta num produto inferior
- Os resultados de cada fase devem ser analisados e aprovados para que se passe à fase seguinte
  - certificar-se que resultados intermediários são consistentes com a entrada e com requisitos do sistema
  - Muito laços para fases anteriores
- Requisitos do usuário são "congelados" antes do início do projeto

Modelo Cascata



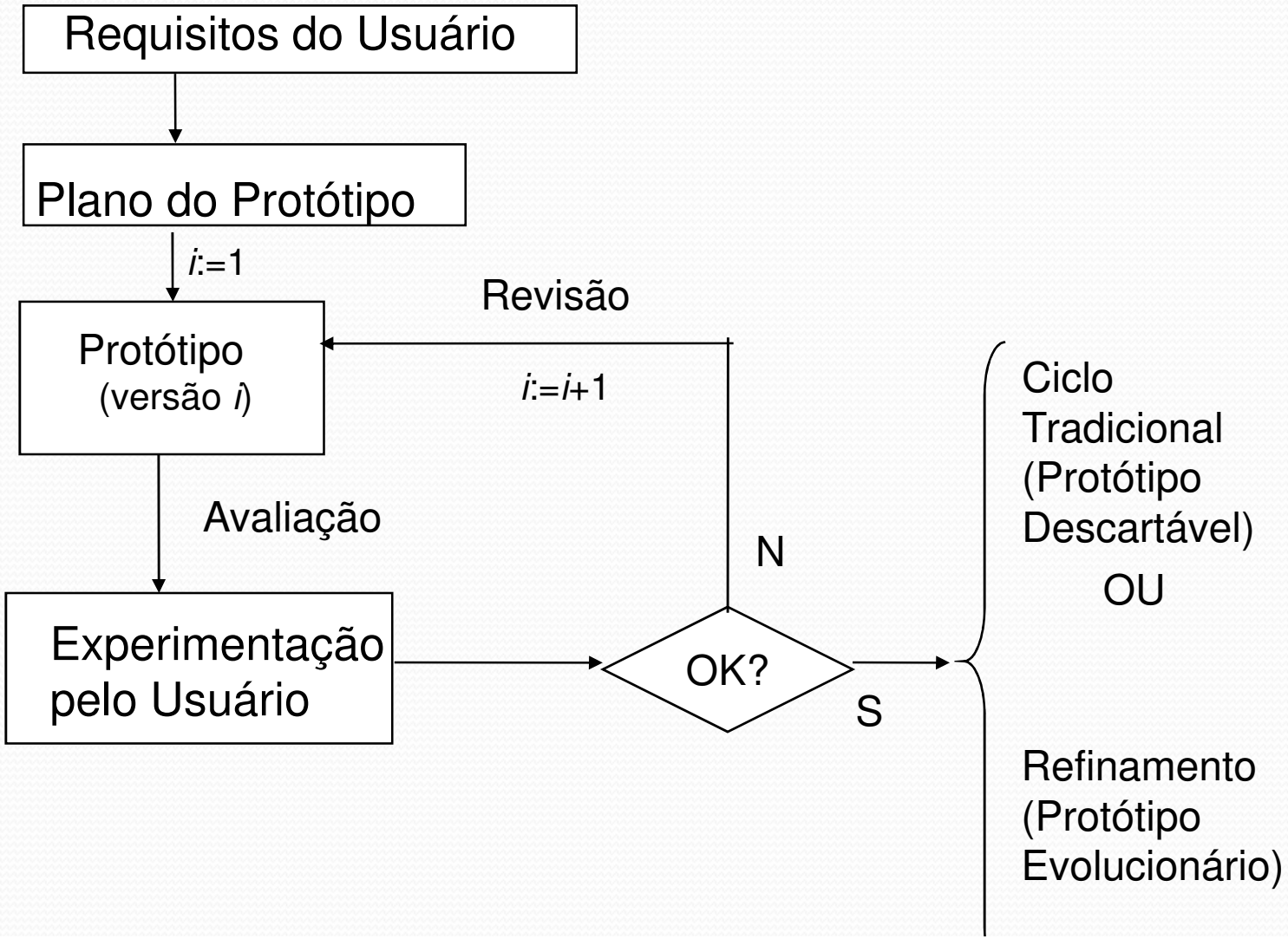
Modelo Cascata ("Waterfall")

- Particionamento inflexível do processo em fases distintas
  - Uma fase deve estar completa para se passar à fase seguinte
  - Tipicamente a ssinatura de documentos marca a transição
- Custo do erro é muito grande, quando descoberto em fases adiantadas
  - Desvios de compreensão são detectados tardiamente (muitas vezes no teste de aceitação ou implantação)
- Permite pouco *feedback* do usuário
  - requisitos "capturados"
  - usuário não acompanha desenvolvimento pois não entende modelos e documentação intermediária
- Dificuldade de assimilar e reagir a mudanças (de requisitos, de critérios de qualidade , etc) depois de iniciado o processo

Modelo Cascata ("Waterfall")

- **Grande mérito:** distinguiu o desenvolvimento de software de programação, e diferenciou a natureza das diferentes atividades envolvidas
- À origem de pesquisas sobre como coletar e representar requisitos, quais as melhores formas de documentar os resultados de cada fase, como validar artefatos, uma melhor compreensão das fases e detalhamento em subatividades, entre tantas outras
- Apropriado somente quando os requisitos estão bem compreendidos e requisitos são estáveis
  - Para grande parte de aplicações, requisitos estáveis é uma utopia
- Bastante usado em projetos de engenharia de sistemas de grande porte, onde um sistema é desenvolvido em várias localidades

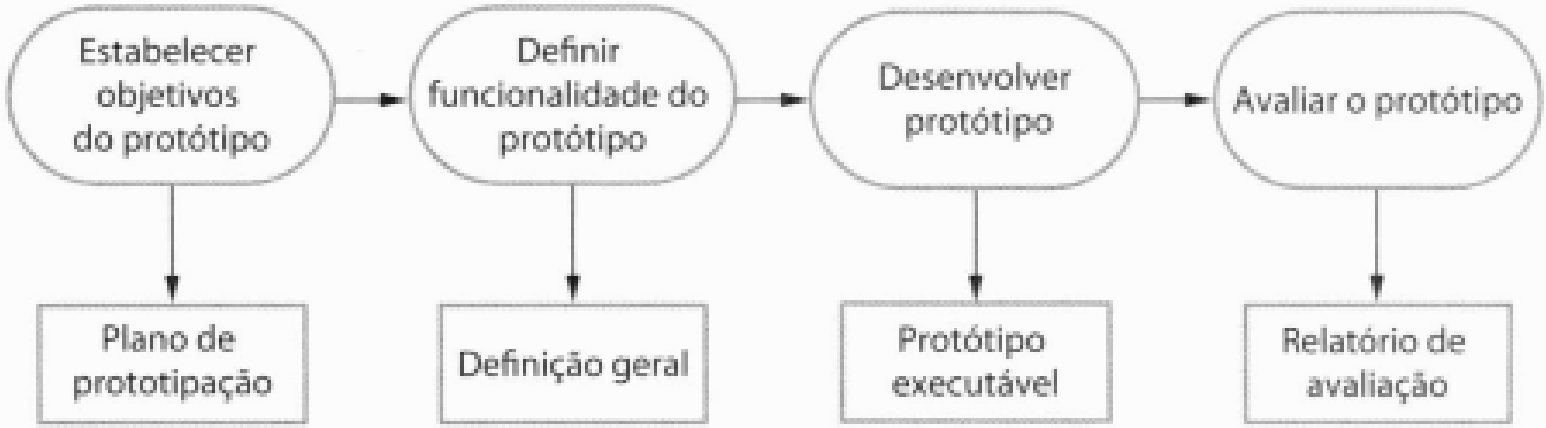
Modelo de Prototipação



Modelo de prototipação

- Protótipo: versão inicial de sistema
  - Compreender e validar requisitos
  - Testar idéias
  - Mostrar viabilidade
  - Testar opções de projeto
  - Demonstrar conceitos
  - Projeto de interfaces
- Rápido e barato
- Não precisa envolver implementação
- Protótipo ≠ sistema final

Modelo de Prototipação



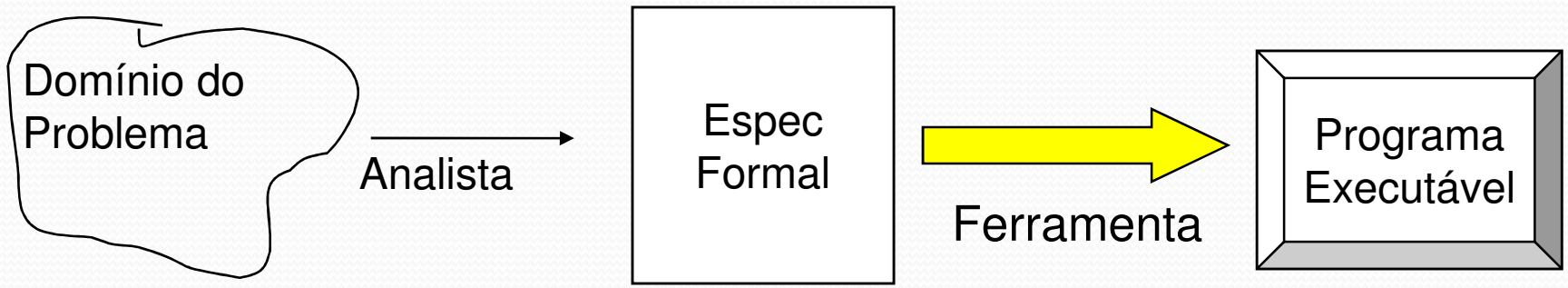
- Protótipo **Vertical** (uma funcionalidade em profundidade) *versus* Protótipo **Horizontal** (amplitude de funções superficiais)
- Protótipo **Evolucionário** (objetivo é evoluir até chegar a um produto) *versus* Protótipo **Descartável** (objetivo é entender requisitos)

Modelo de Prototipação: características

- Boa ferramenta para
  - compreensão de requisitos
  - Discussão de projeto de interface
  - provas de conceitos
  - Diminuir riscos, viabilidade técnica
- É difícil convencer o cliente que é descartável
  - Efeito colcha de retalhos
  - Descartável → evolucionário (incremental)
- Problemas de visibilidade do processo
  - Gerentes verificam o andamento do projeto através dos artefatos gerados
  - Devido a velocidade de entrega de versões executáveis, a documentação fica dificuldade já que torna o custo de documentar cada versão muito alto

Modelos Formais/Transformação de Modelos

- Requisitos são definidos em uma especificação detalhada (uso de notações formais)
- Processo de desenvolvimento transformacional



* Transformações são AUTOMÁTICAS suportadas por ferramentas (geradores)
* Analistas criam modelos
* Dispensa o papel de programador
* Modificações sempre feitas na especificação (modelos)

Um pequeno exemplo: Z

Tipos

$$[NAME, DATE].$$

State Space (Variáveis, Relações)

BirthdayBook

$$known : \mathbb{P}\, NAME$$
$$birthday : NAME \pfun DATE$$
$$known = \mathrm{dom}\, birthday$$

State change (pré e pós condições)

AddBirthday

$$\Delta BirthdayBook$$
$$name? : NAME$$
$$date? : DATE$$
$$name? \notin known$$
$$birthday' = birthday \cup \{name? \mapsto date?\}$$

$$known' = known \cup \{name?\}.$$

$$\begin{aligned}
known' &= \mathrm{dom}\, birthday' && \text{[invariant after]}\\
&= \mathrm{dom}(birthday \cup \{name? \mapsto date?\}) && \text{[spec. of } AddBirthday\text{]}\\
&= \mathrm{dom}\, birthday \cup \mathrm{dom}\, \{name? \mapsto date?\} && \text{[fact about 'dom']}\\
&= \mathrm{dom}\, birthday \cup \{name?\} && \text{[fact about 'dom']}\\
&= known \cup \{name?\}. && \text{[invariant before]}
\end{aligned}$$

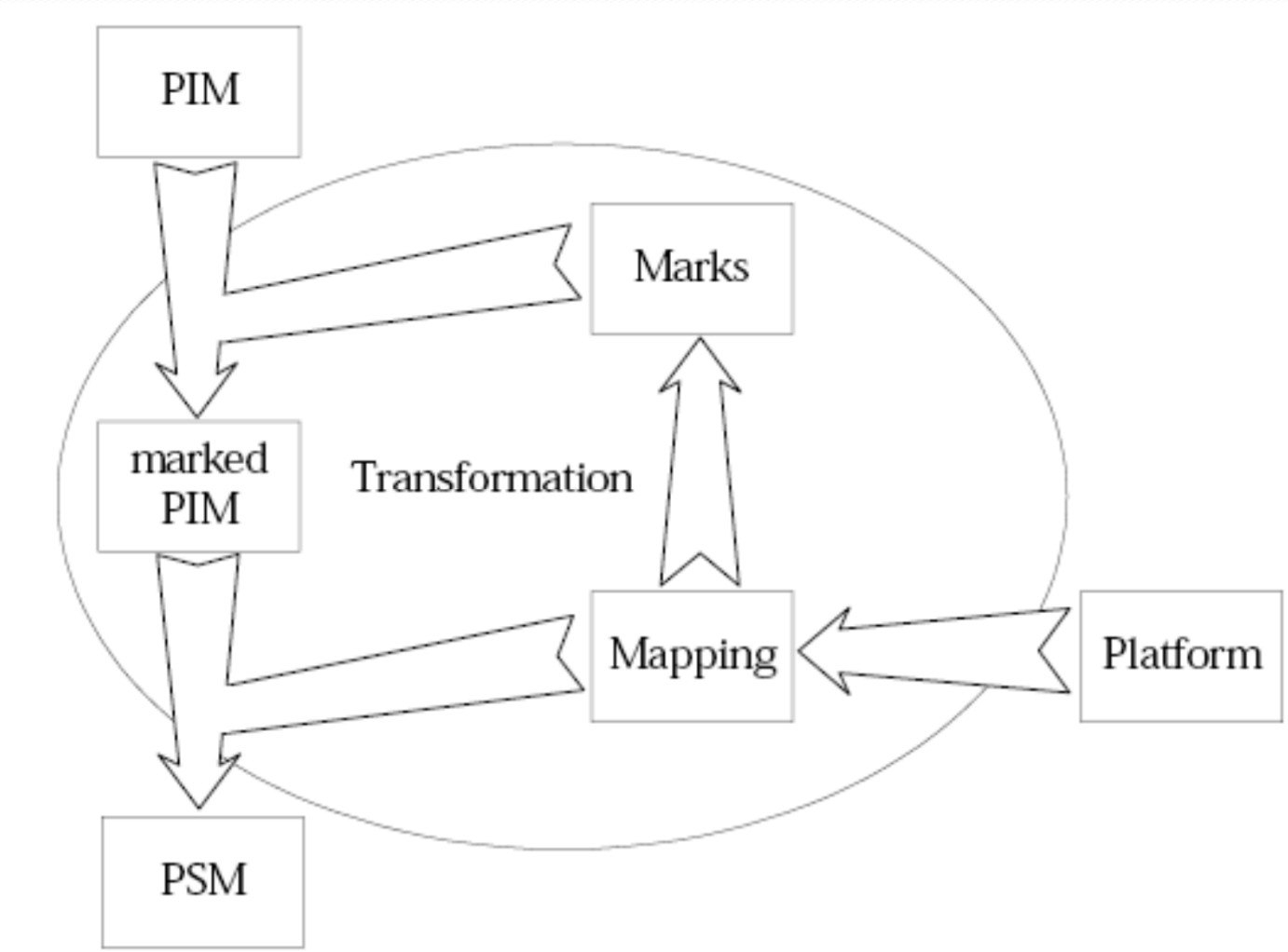
Modelos Formais

- Vantagens
  - Especificação compacta
  - Correção da especificação pode ser provada
  - Geração automática de transformações
- Problemas
  - Necessidade de habilidades não usuais dos analistas (conhecer notações/linguagens formais e formas de verificação de corretude e consistência de uma especificação)
  - Dificuldade de especificar formalmente algumas partes de um sistema
  - Ainda carece de ferramentas de apoio ao desenvolvimento
- Aplicabilidade
  - Sistemas críticos especialmente aqueles em que aspectos de segurança e confiabilidade devem ser verificados ANTES de entrar em operação

MDA: Model Driven Architecture

- proposta da OMG para a construção de sistemas através da confecção de modelos que assumam a posição central no processo de desenvolvimento, elevando o nível de abstração em todas as etapas do projeto, e por fim permitindo aumentar a automação na implementação de sistemas.
- Modelos
  - CIM (**Computation Independent Model**): Visão do sistema do ponto de vista independente de computação.
  - PIM (**Platform Independent Model**): Visão do sistema do ponto de vista independente de plataforma.
  - PSM (**Platform Specific Model**): Visão do sistema do ponto de vista específico de uma plataforma.

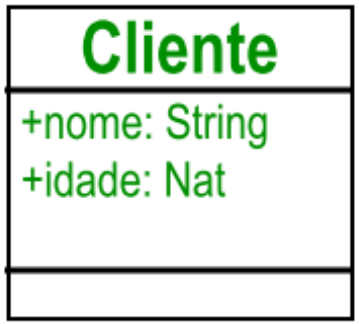
Transformações MDA

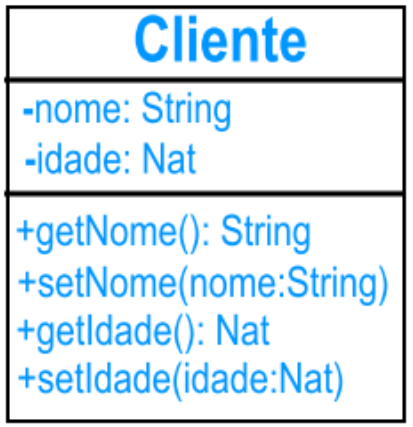


MDA

- Transformações PIM→PSM e PSM→código são essenciais em MDA
- Transformações devem ser bem definidas e, de preferência, apoiadas por ferramentas
- Objetivos das transformações
  - otimização
  - refatoração
  - geração de código
  - engenharia reversa
  - migração

Transformações MDA





Para cada atributo público nomeAtr: Tipo da classe **nomeClasse** do PIM, criar os seguintes atributos e operações na classe **nomeClasse** do PSM:

- um atributo privado nomeAtr:Tipo
- uma operação pública getAtr():Tipo
- uma operação pública setAtr(atr:Tipo)

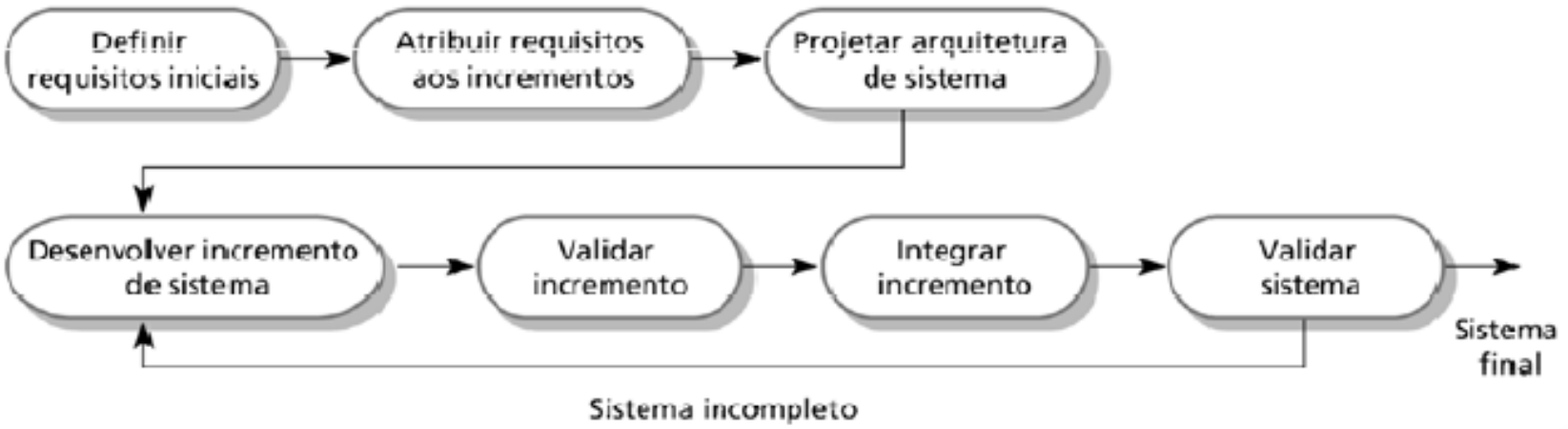
Modelo Iterativo Incremental

- Organização das atividades em incrementos, desenvolvida ao longo de ciclos
  - iteração
  - Duração fixa ou variável
- Cada ciclo visa realizar um conjunto completo de atividades do desenvolvimento
- Ao fim de cada ciclo, decide-se como será o próximo ciclo
- Precursor: Modelo Espiral
- Tendências
  - Processo Unificado e suas variantes (heavy weight)
  - Métodos Ágeis (XP, SCRUM, Modelagem Ágil, etc – light weight)

Modelo Iterativo Incremental

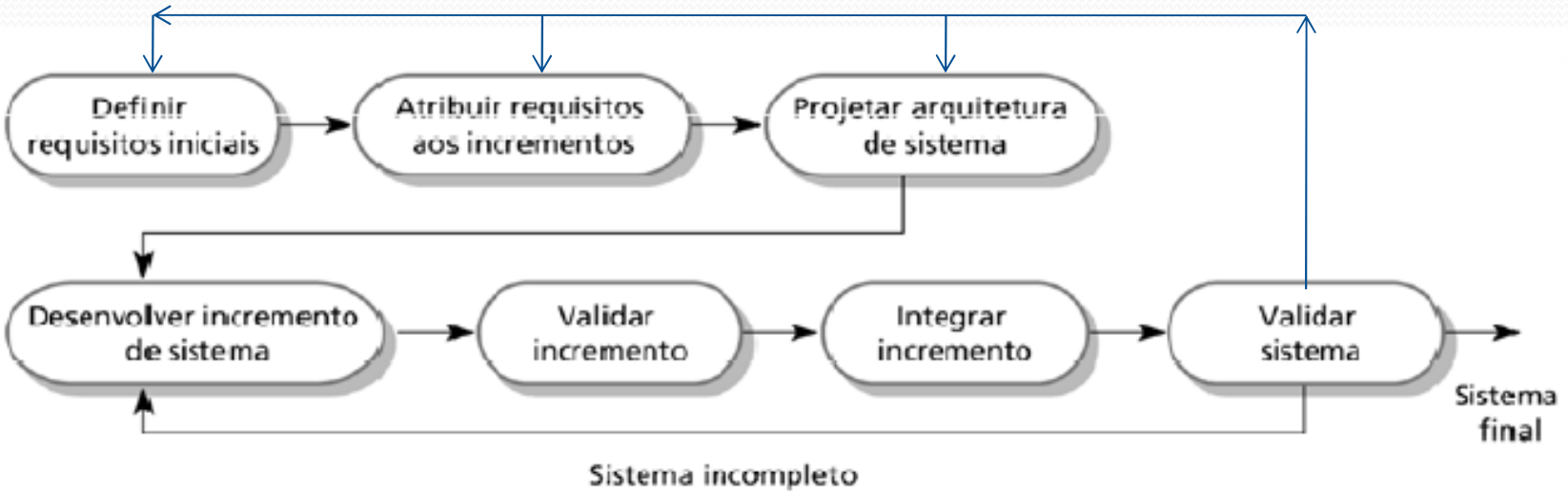
- Ao invés de entregar o sistema como uma única entrega, o desenvolvimento e a entrega são separados em incrementos, sendo que cada incremento fornece parte da funcionalidade solicitada.
- Os requisitos de usuário são priorizados e os requisitos de prioridade mais alta são incluídos nos incrementos iniciais.
- Cada incremento pode usar uma abordagem interna em cascata (requisito bem entendido) ou evolutiva (requisito ainda não estável)

Modelo Iterativo Incremental



Sommerville

Modelo Iterativo Incremental

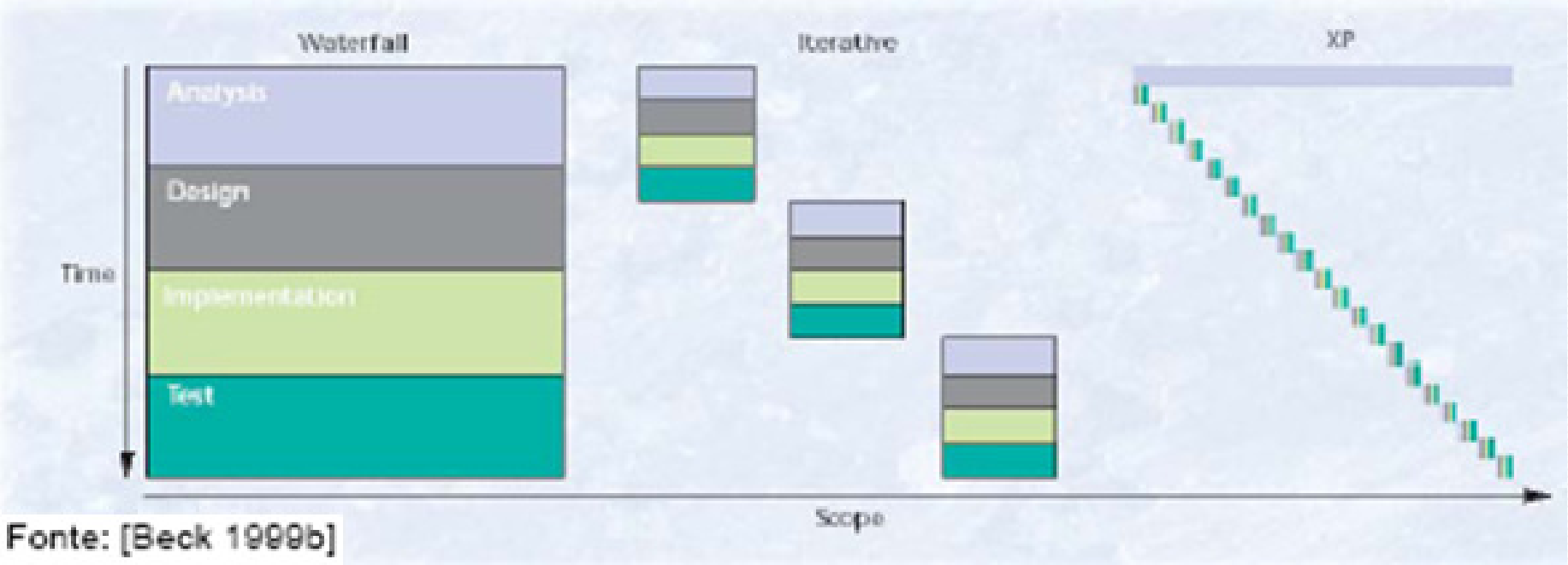


Sommerville + Karin

Modelo Iterativo Incremental

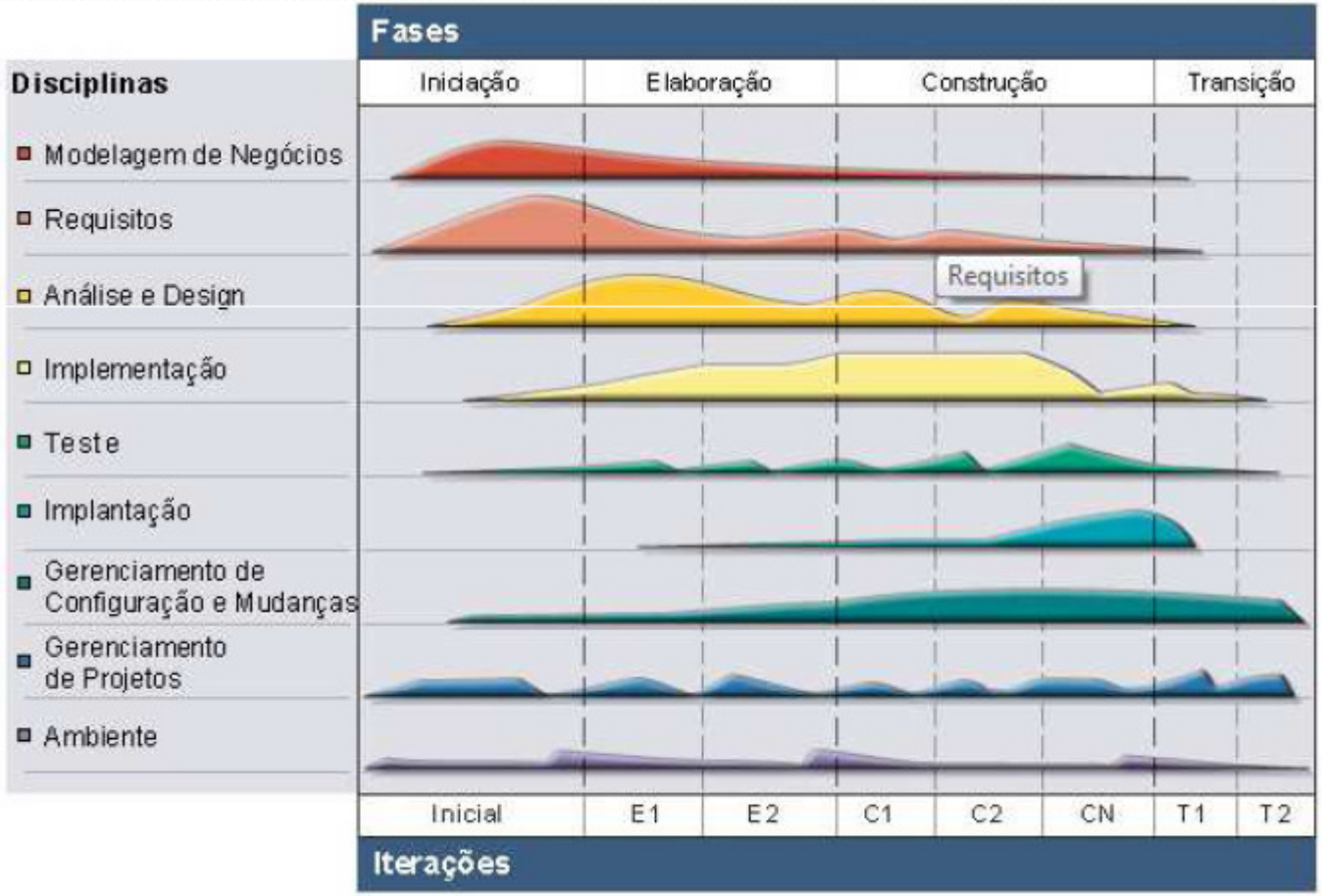
- Vantagens
  - Primeiros serviços (mais importantes) são entregues antes do término do projeto
  - Primeiros incrementos servem como protótipo para os usuários tomarem conhecimento do sistema
  - Diminuição do risco para o projeto como um todo
  - Como os serviços iniciais são mais testados, os usuários irão encontrar menos erros nos serviços críticos do sistema

Iterações

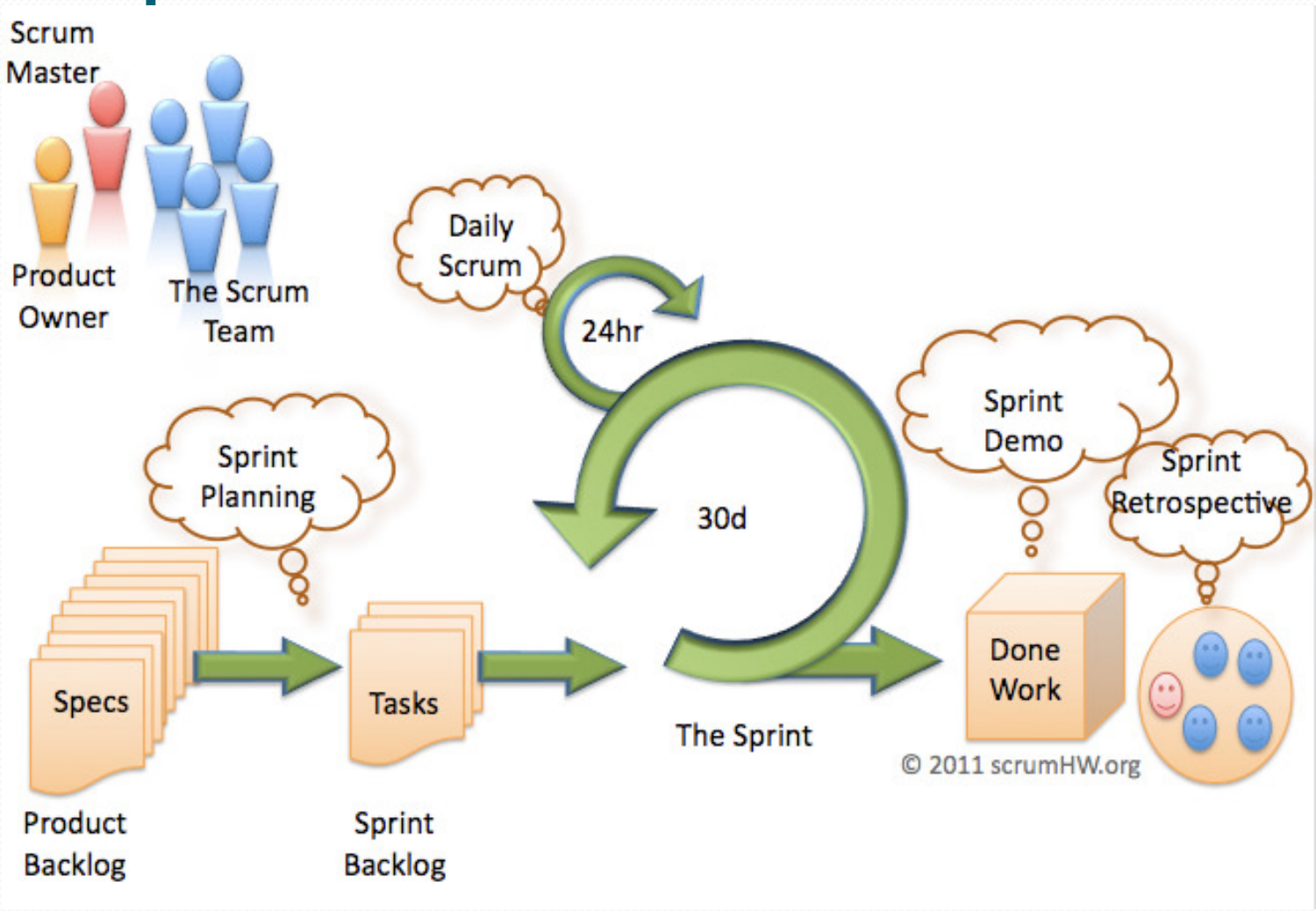


Fonte: [Beck 1999b]

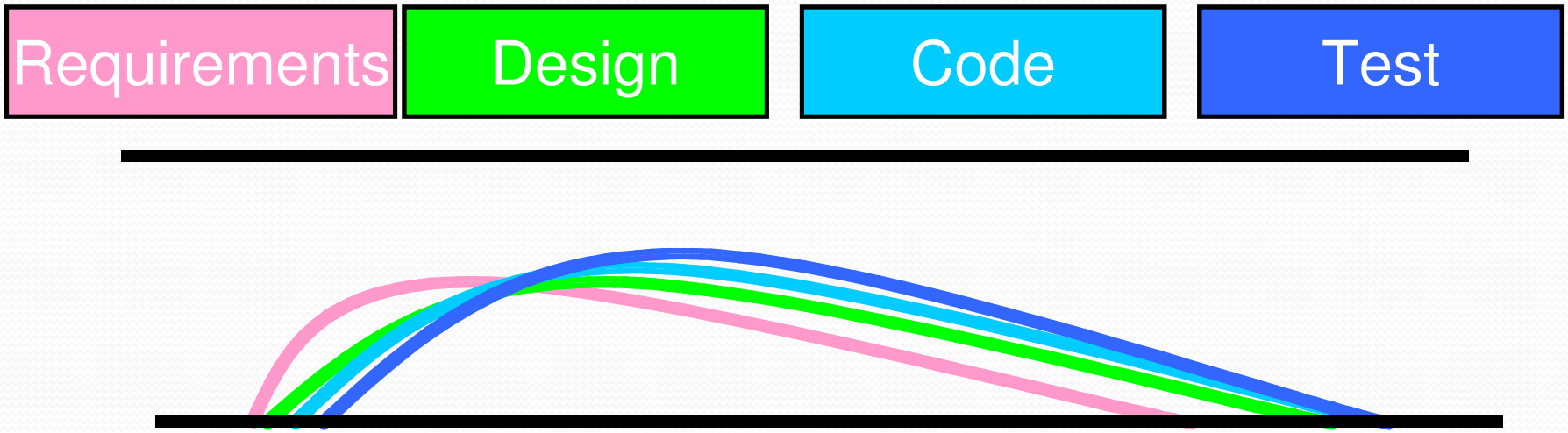
Exemplo: Processo Unificado



Exemplo: SCRUM



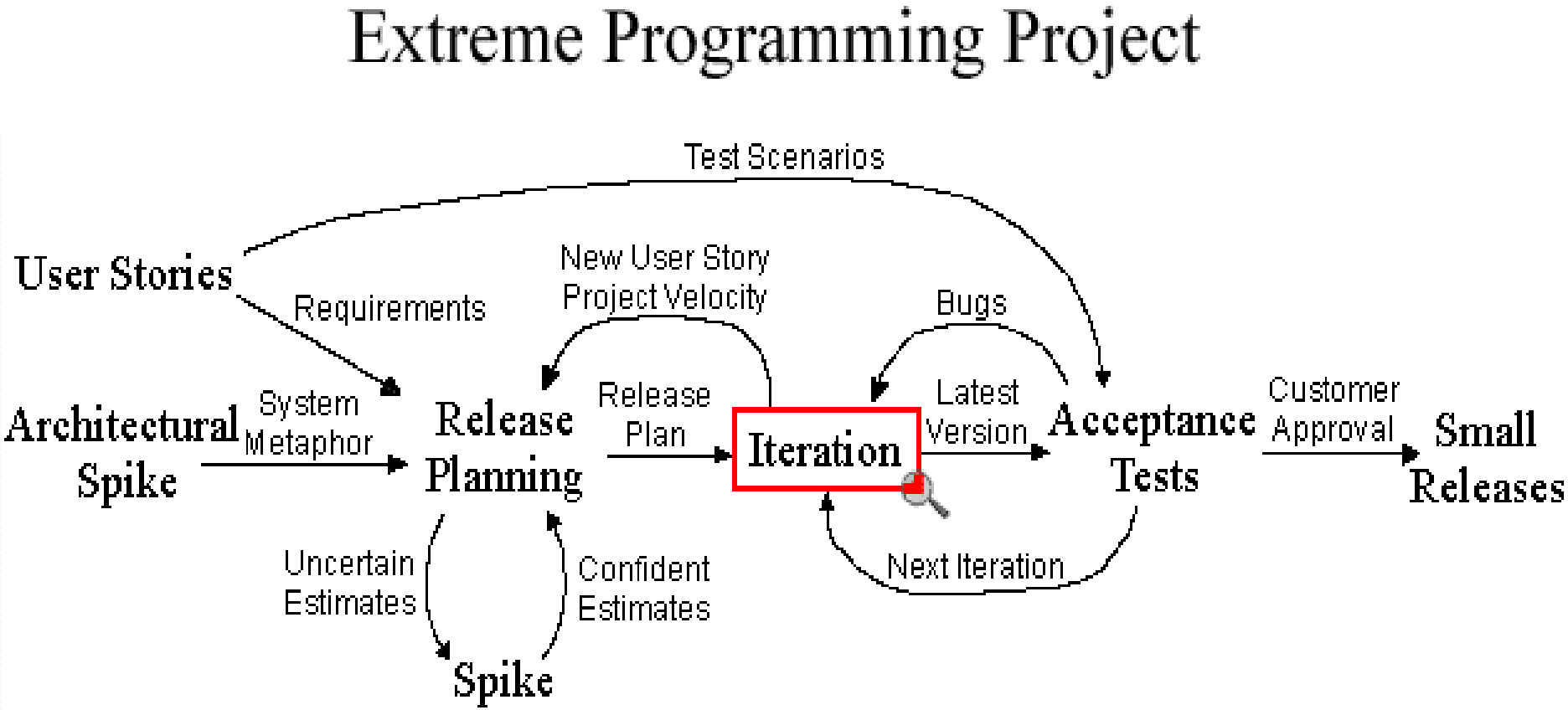
Scrum : Sprint



Source: “The New New Product Development Game” by Takeuchi and Nonaka. *Harvard Business Review,* January 1986.

DONE !!!

Exemplo : XP



Fonte: www.extremeprogramming.org

Modelos de Ciclo de Vida do Software

- Discussão:
  - Qual o modelo de ciclo de vida adotado no seu ambiente de trabalho?
  - A seu ver, quais as razões para a adoção deste modelo de ciclo?
- **Aspectos comuns a todos modelos**:
  - Todos começam com (ou visam) uma compreensão melhor dos requisitos do sistema a ser desenvolvido
  - Todos visam aumento da qualidade dos resultados parciais e finais
  - Todos visam melhoria da gerência do processo de desenvolvimento (alguns, o controle do processo; outros, o acompanhamento do processo)

Para saber mais ...

- Recomendada
  - Sommervile, Ian. Engenharia de software. 8ª edição. Pearson Education. São Paulo:, 2007.
    - Capítulo 4 – Descreve os diferentes modelos, e discute com mais detalhes as atividades genéricas
    - Obs: agrupa os processos de forma alternativa
- Outra boa alternativa
  - Pressman, Roger. Engenharia de Software. Ed. Makron Books, 2006.
  - Capítulo 2

Créditos

- Este material inclui slides cedidos ou adaptados com permissão de seus autores
  - Marcelo Pimenta - UFRGS
  - Material didático Pearson Education
    - SOMMERVILLE, Ian. Engenharia de software. 8ª edição. Pearson Education. São Paulo:, 2007