Processos de Desenvolvimento de Software

O processo de software

- **Um conjunto estruturado de atividades, procedimentos, artefatos e ferramentas necessários para o desenvolvimento de um sistema de software**
  - **Atividades: Especificação, Projeto, Validação, Evolução**
  - **Exemplos: Processo Unificado (RUP), Programação Extrema, UML Components**
- **Um modelo de processo de software apresenta a descrição de um processo de uma perspectiva particular, normalmente focando apenas em algumas atividades.**
- **Fundamentais para qualidade**
  - **Processos ajudam mas...**
    - **No silver bullet!**

Atividades de um processo de desenvolvimento

1. **Especificação de software**
2. **Projeto e implementação de software**
3. **Validação de software**
4. **Evolução de software**

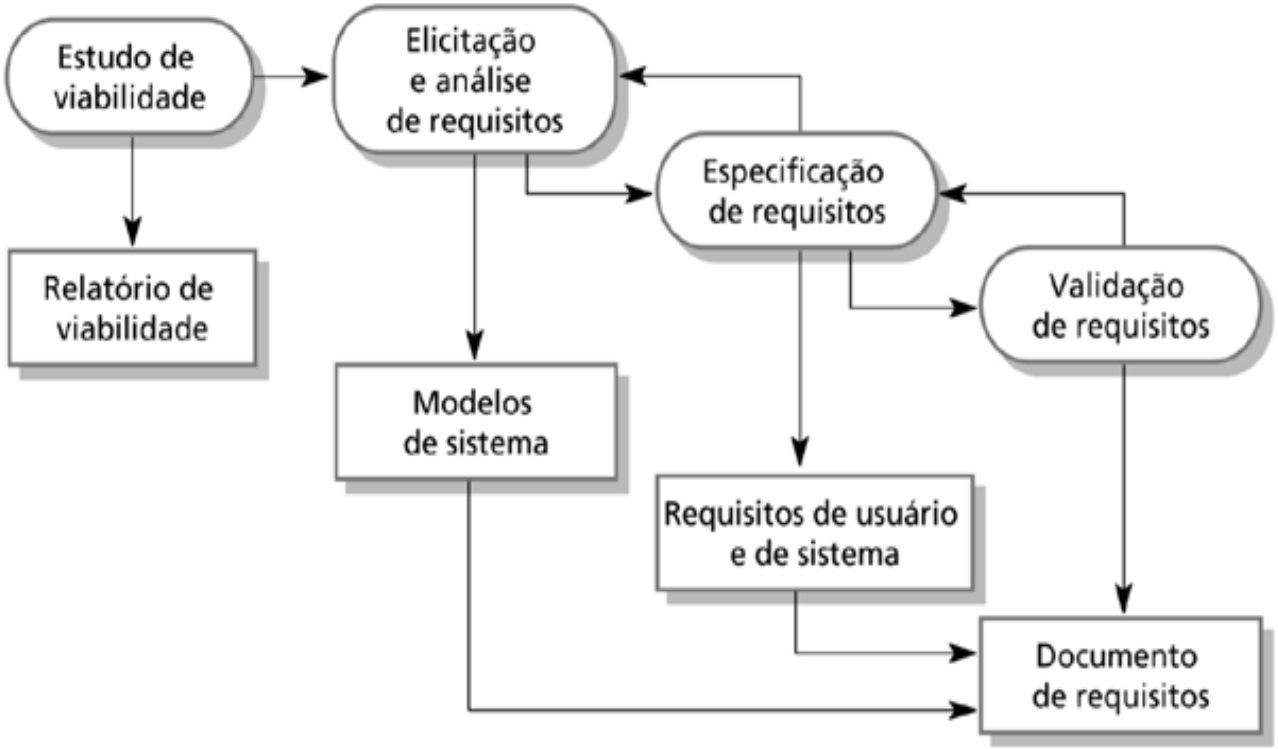
1 - Especificação de software

- **O processo para definir quais serviços são necessários e identificar as restrições de operação e de desenvolvimento do sistema.**
- **Processo de engenharia de requisitos**
  - **Estudo de viabilidade;**
    - **Realizado antes do projeto ser iniciado**
  - **Elicitação e análise de requisitos;**
  - **Especificação de requisitos;**
  - **Validação de requisitos.**

O processo de engenharia de requisitos

Figura 4.6

Processo de engenharia de requisitos.



Também pode envolver a prototipação de partes do sistema!

2 - Projeto e implementação de software

- **É o processo de conversão da especificação em um sistema de software**
- **Projeto de software**
  - **Projetar uma estrutura de software que atenda à especificação.**
- **Implementação**
  - **Transformar essa estrutura em um programa executável.**
- **As atividades de projeto e implementação são fortemente relacionadas e podem ser intercaladas.**

Atividades do processo de projeto

- **Projeto de arquitetura**
- **Especificação abstrata**
- **Projeto de interfaces entre componentes**
- **Projeto de componente**
- **Projeto de estrutura de dados**
- **Projeto de algoritmo**

Métodos estruturados

- **Abordagens sistemáticas para projetar sistemas de software**
  - **Project (gerenciamento) vs. Design (desenvolvimento)**
- **O projeto é, em geral, documentado como um conjunto de modelos gráficos**
- **Modelos possíveis**
  - **Modelo de objeto;**
  - **Modelo de sequência;**
  - **Modelo de transição de estado;**
  - **Modelo estruturado;**
  - **Modelo de fluxo de dados.**

Programação e depuração

- **É a transformação de um projeto em um programa e a remoção de defeitos desse programa.**
- **Programação é uma atividade pessoal – não há processo genérico de programação.**
  - **Há algumas práticas, porém, que são universalmente consideradas boas**
- **Programadores realizam alguns testes para descobrir defeitos no programa e removem esses defeitos no processo de depuração**

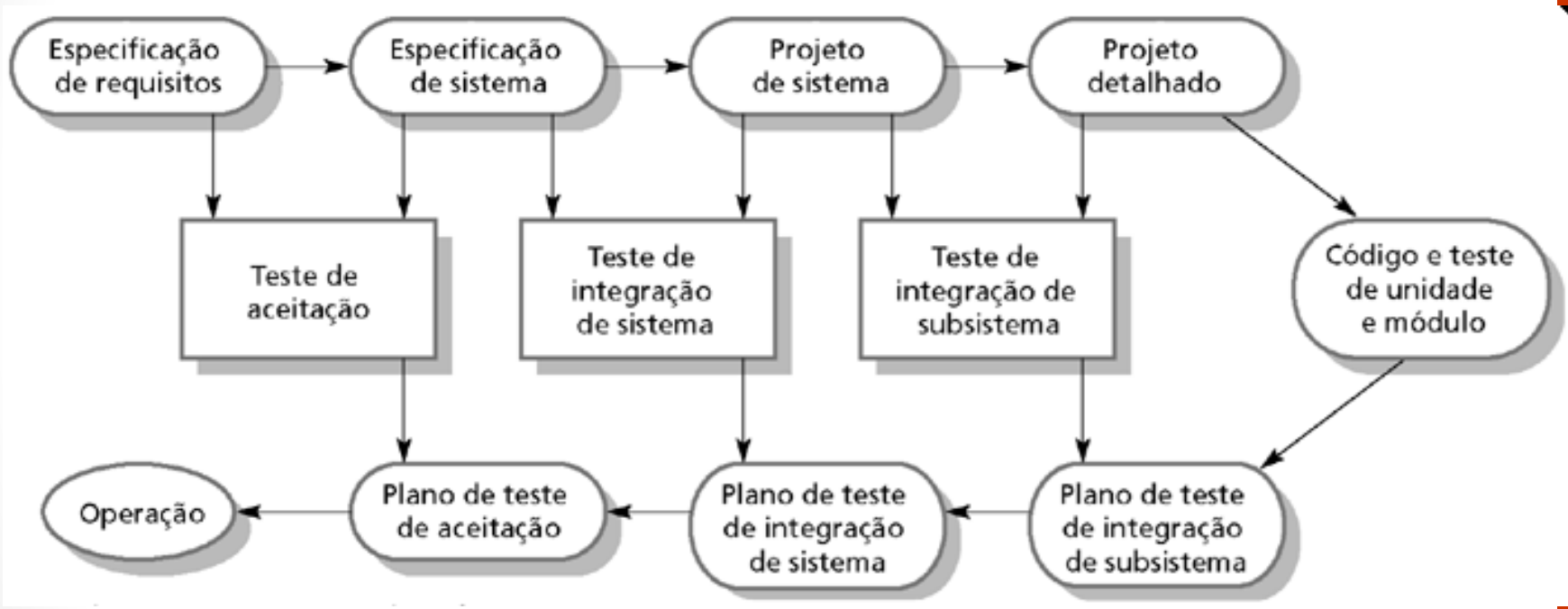
3 - Validação de software

- **Verificação e validação (V & V) têm a intenção de mostrar que um sistema está em conformidade com a sua especificação e que atende aos requisitos do cliente**
- **Verificação: "construímos o sistema corretamente?"**
  - **Ex: inspeção de código, análise estática**
- **Validação: "construímos o sistema correto?"**
  - **Ex: testes, animação de especificações**
- **Testes envolvem a execução do sistema com casos de teste que são derivados da especificação do sistema e de dados reais a ser processados por ele.**

Estágios de teste

- **Teste de componente ou unidade**
  - **Os componentes individuais são testados independentemente;**
  - **Esses componentes podem ser funções ou classes de objetos, ou grupos coerentes dessas entidades.**
- **Teste de sistema**
  - **Teste de sistema como um todo. O teste das propriedades emergentes é particularmente importante.**
- **Teste de aceitação**
  - **Teste com dados do cliente para verificar se o sistema atende às suas necessidades.**

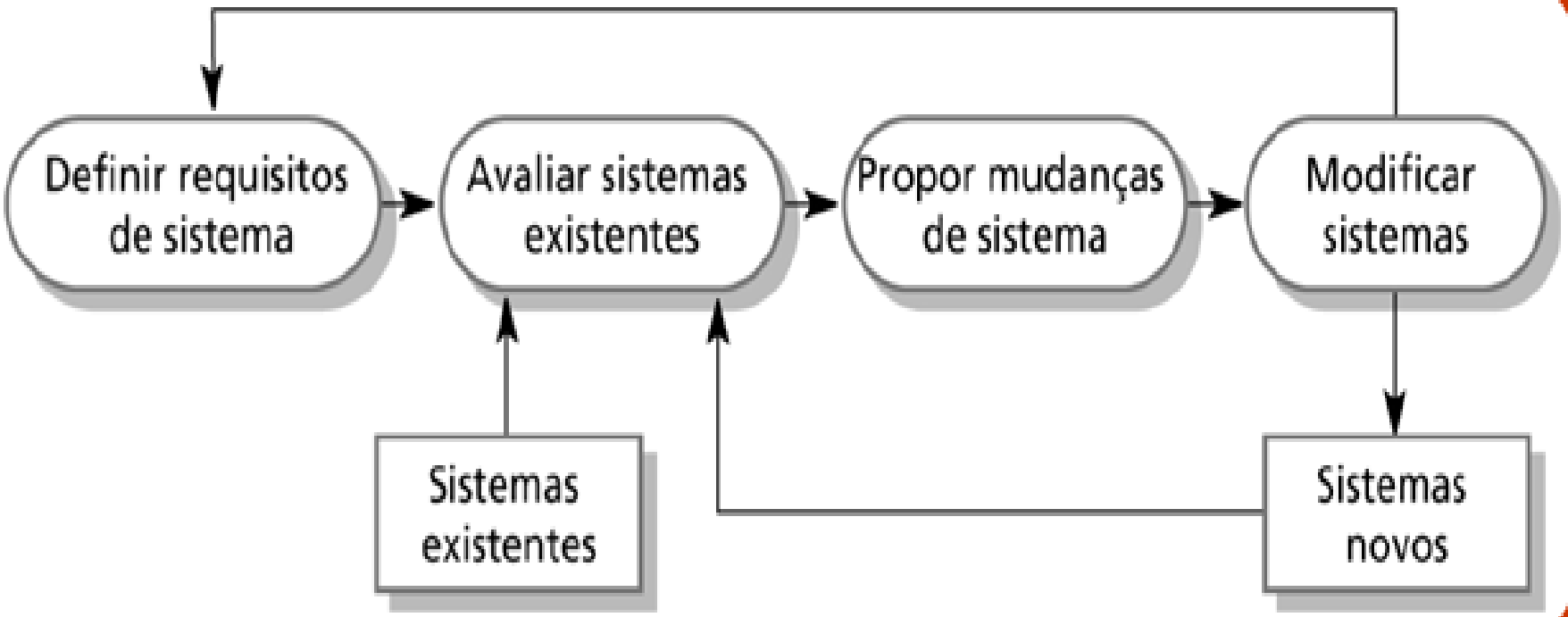
Estágios de teste



4 - Evolução de software

- **O software é inerentemente flexível e pode mudar**
- **Requisitos mudam devido a diversos fatores e o software deve acompanhar essas mudanças**
- **Processos antigos separavam explicitamente desenvolvimento de evolução**
  - **Processos e métodos iterativos (XP, RUP, Espiral) normalmente não fazem uma sepação explícita**
- **Evolução pode se dever a diversas razões:**
  - **Correções (patches)**
  - **Mudanças de requisitos**
  - **Melhoria de funcionalidades pré-existentes**

Evolução de software

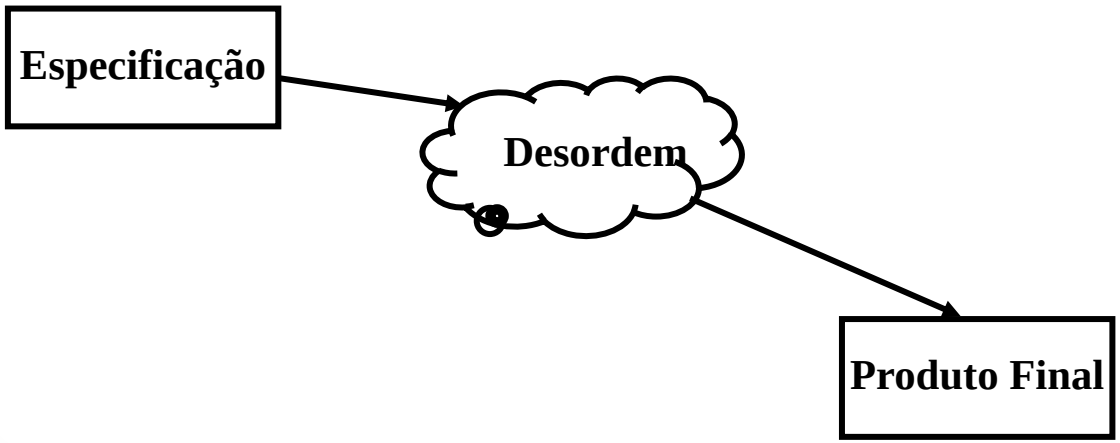


Alguns modelos genéricos de processo de software

- **Modelo cascata**
  - **Fases separadas e distintas de especificação e desenvolvimento.**
- **Prototipação**
  - **Através da construção de um protótipo do sistema, é possível demonstrar a viabilidade do sistema.**
- **Engenharia de software baseada em componentes**
  - **O sistema é montado a partir de componentes existentes.**
- **Desenvolvimento iterativo**
  - **Sistema desenvolvido através de várias etapas**
- **Existem muitas variantes destes modelos**
  - **Ex: desenvolvimento formal onde um processo semelhante ao cascata é usado, mas a especificação formal é refinada durante os vários estágios para um projeto implementável.**

Modelo Codifica-Remenda

- Muito usado (infelizmente...)
- Não exige gerência complexa
  - Nenhuma documentação
  - Nenhum controle gerencial
  - Atraente para alguns desenvolvedores



Ciclo de Vida tradicional (cascata)

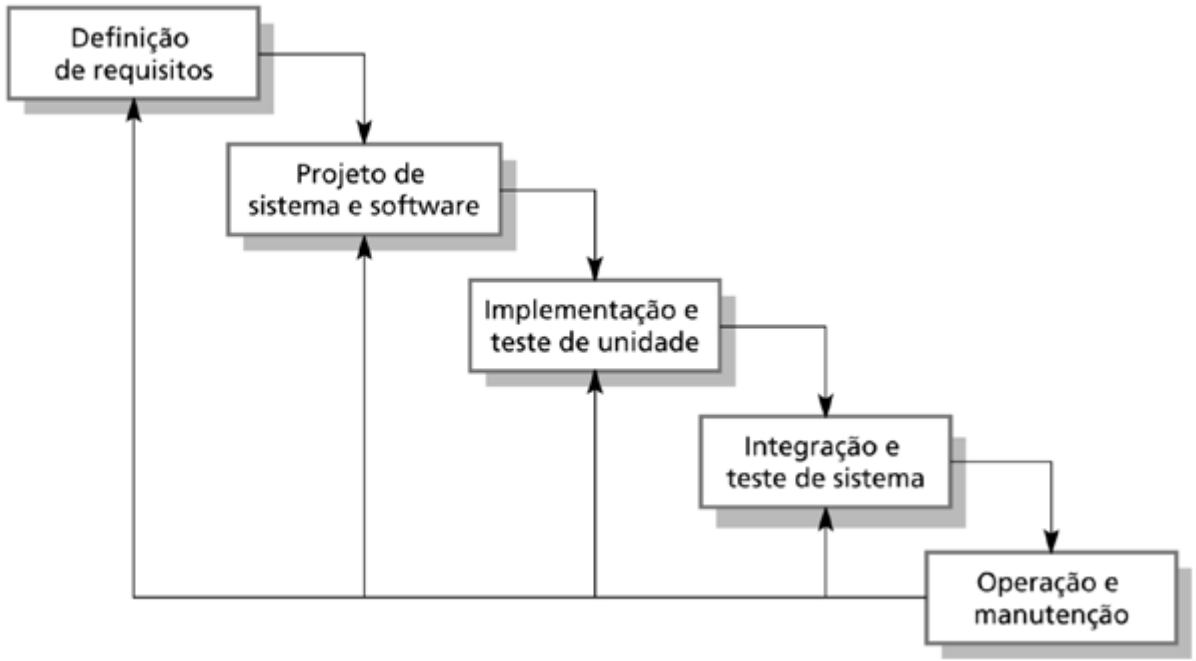
- Modelo mais antigo e o mais usado da engenharia de software
- Modelado em função do ciclo da engenharia convencional
- Requer uma abordagem sistemática, sequencial ao desenvolvimento de software
- Duas diretivas importantes que norteiam o desenvolvimento segundo o modelo cascata:
  - todas as etapas definidas no modelo devem ser realizadas, isto porque, em projetos de grande complexidade, a realização formal destas vai determinar o sucesso ou não do desenvolvimento; a realização informal e implícita de algumas destas etapas poderia ser feita apenas no caso de projetos de pequeno porte;
  - a ordenação das etapas na forma como foi apresentada deve ser rigorosamente respeitada;

Modelo cascata

- **Resposta ao modelo code-and-fix vigente na década de 70**

Figura 4.1

Ciclo de vida de software.



Modelo cascata

- Fases
  - Análise e definição de requisitos
  - Projeto de sistema e software
  - Implementação e teste de unidade
  - Integração e teste de sistema
  - Operação e manutenção
- Primeiro modelo a organizar as atividades de desenvolvimento
- Uma fase tem de estar completa antes de passar para a próxima.
  - Saídas das fases são acordadas contratualmente!
- Todas as fases envolvem atividades de validação

Problemas com o Ciclo de Vida Clássico

- Projetos reais raramente seguem o fluxo sequencial que o modelo propõe
- No início é difícil estabelecer explicitamente todos os requisitos.
  - No começo dos projetos sempre existe uma incerteza natural
- O cliente deve ter paciência.
  - Uma versão executável do software só fica disponível numa etapa avançada do desenvolvimento
- Particionamento inflexível do projeto em estágios
  - Dificulta a resposta aos requisitos de mudança do cliente
- Documentos “completamente elaborados” são necessários para fazer as transições entre estágios
- Apropriado somente quando os requisitos são bem compreendidos e quando as mudanças são raras
  - Poucos sistemas têm requisitos estáveis

Ciclo de Vida Clássico

- Apesar da fragilidades, é melhor do que uma abordagem casual ao desenvolvimento de software

Desenvolvimento evolucionário

- Implementação inicial, exposição do resultado aos comentários do usuário e refinamento desse resultado em versões
- Especificação, desenvolvimento e validação intercaladas
- Dois tipos:
  - Desenvolvimento exploratório: explora requisitos e entrega um sistema final
    - Trabalha com clientes e evolui o sistema final de um esboço de especificação inicial. Requisitos devem estar bem entendidos
  - Prototipação *throwaway*: Objetivo é entender os requisitos do cliente.
    - Requisitos ainda pouco entendidos

Desenvolvimento evolucionário

- Vantagens
  - Especificação desenvolvida de forma incremental
- Problemas
  - Falta de visibilidade do processo
  - Sistemas são, em geral, mal estruturados devido às mudanças contínuas
  - Habilidades especiais podem ser requeridas(ex. linguagens para rápida preparação de protótipos
- Aplicabilidade
  - Para sistemas interativos pequenos ou médios
  - Para partes de sistemas grandes (ex. a interface de usuário)
  - Para sistemas de curto-prazo

O que é um protótipo?

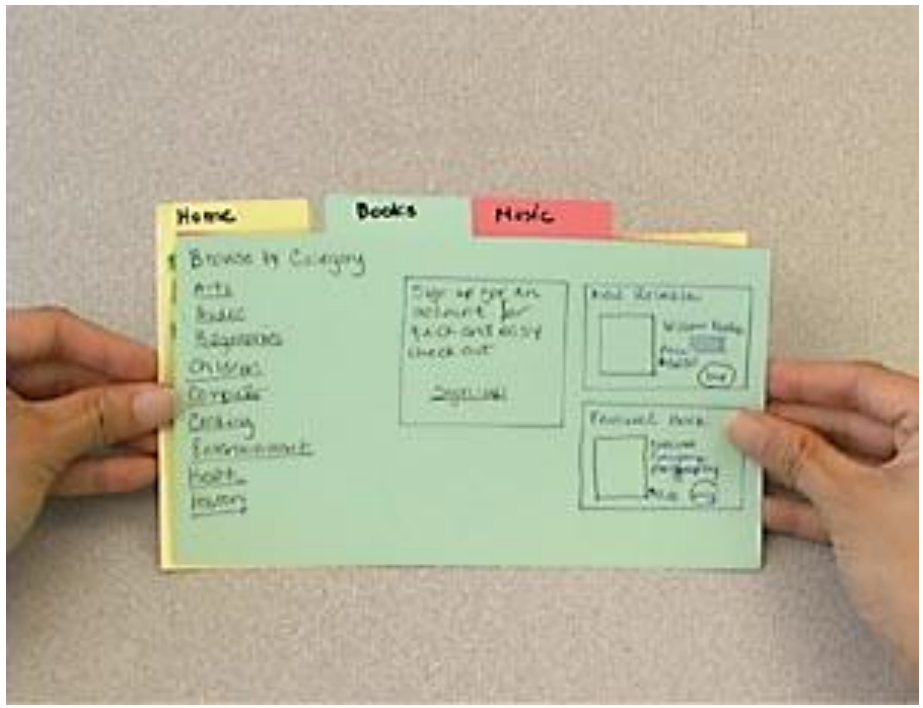
- Versão inicial de um sistema de software utilizada para mostrar conceitos, experimentar opções de projeto e, em geral, para conhecer mais sobre os problemas e suas possíveis soluções.
- Deve ser rapidamente elaborado para validar com o cliente o software esperado, além de ser com custo eficiente.
- Faz parte do desenvolvimento de software em vários momentos:
  - Levantamento de Requisitos;
  - Avaliação de Requisitos;
  - Análise e redução de risco.

Protótipo - Benefícios

- Possíveis equívocos entre desenvolvedores de software e usuários podem ser identificados à medida que as funções do sistema são apresentadas;
- A equipe de desenvolvimento de software pode encontrar requisitos incompletos e/ou inconsistentes quando o protótipo é desenvolvido;
- Um sistema operante, embora limitado, se torna rapidamente disponível, a fim de mostrar viabilidade e a utilidade da aplicação para a gerência;
- O protótipo pode ser utilizado como uma base para escrever a especificação para um sistema com qualidade de produção.

Formas de protótipo

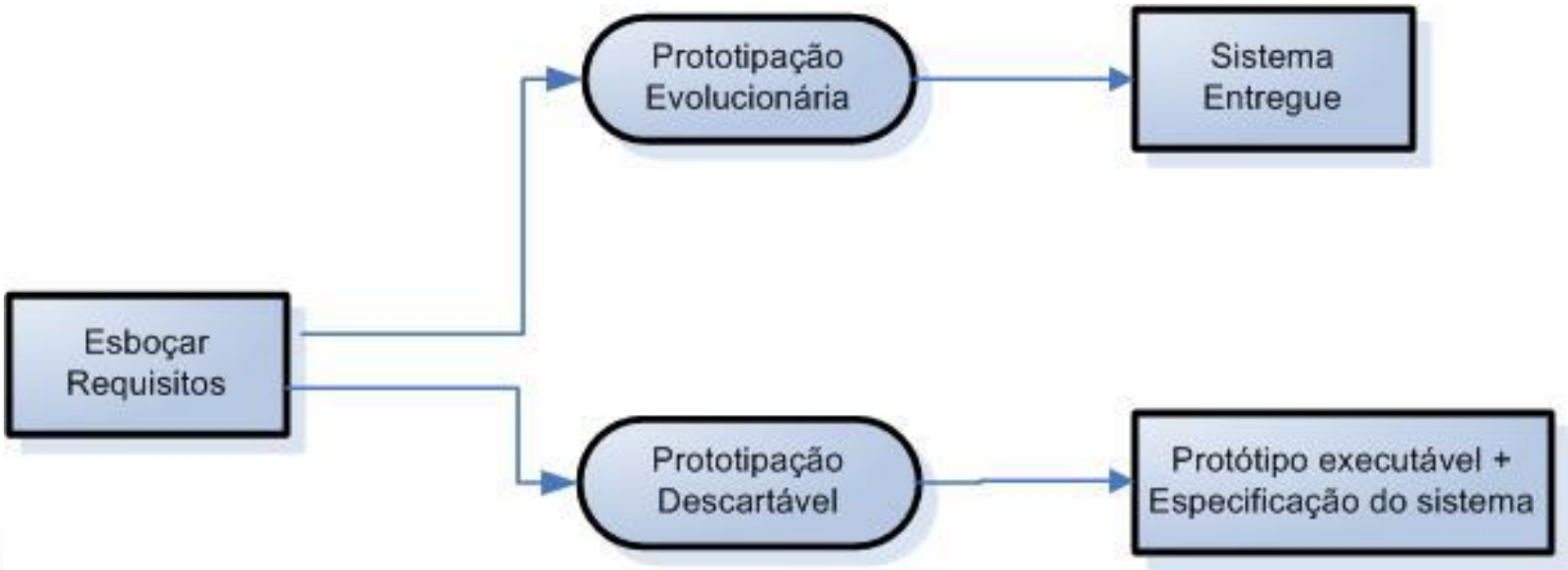
- Protótipo em papel ou modelo executável
  - retrata a interface homem – máquina, capacitando o cliente a compreender a forma de interação com o software



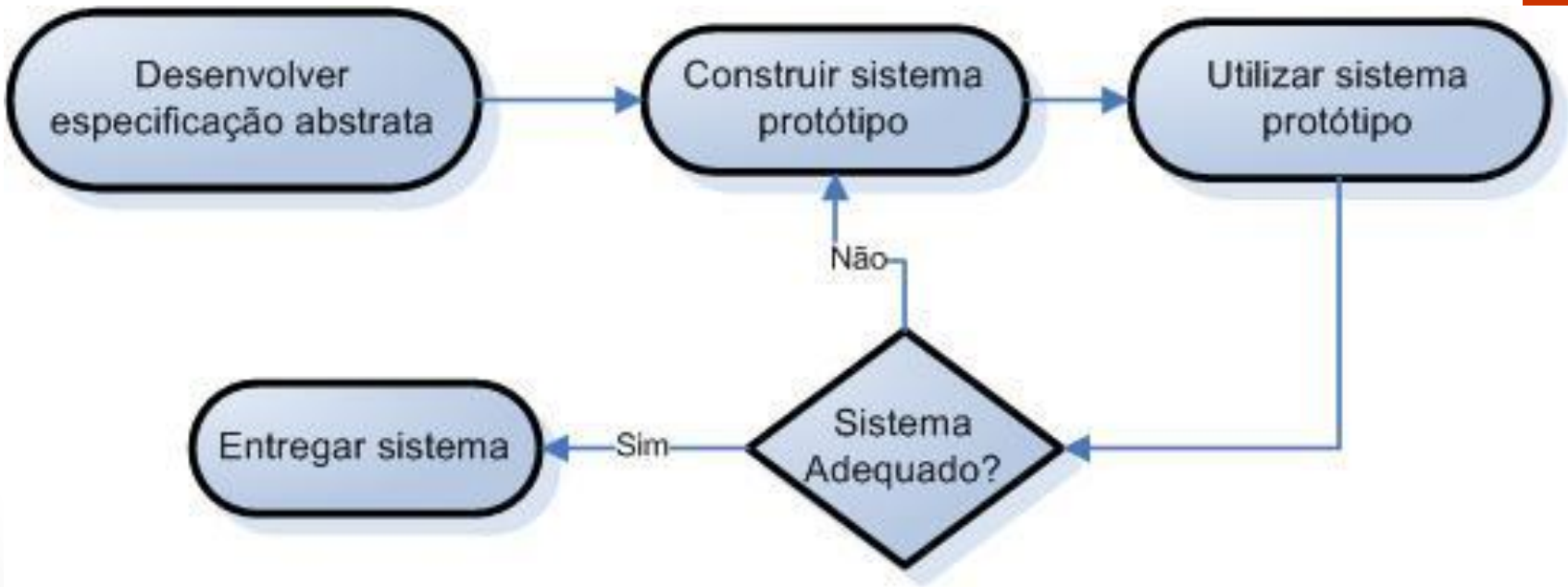
Formas de Protótipo

- Protótipo de trabalho
  - Um protótipo funcional, que implementa um subconjunto dos requisitos indicados
  - Não é um sistema completo, e deixa a desejar em alguns aspectos. Normalmente a interface com o usuário não é implementada no protótipo.
- Abordagens de produção:
  - Protótipo Evolutivo
    - O protótipo usado para a definição do sistema acaba sendo usado como base do desenvolvimento, fazendo parte do código final.
  - Protótipo Exploratório
    - Serve apenas para a definição do que fazer, sendo abandonado em seguida.

Abordagens do Desenvolvimento de Protótipos



Prototipação Evolucionária



Prototipação Evolucionária

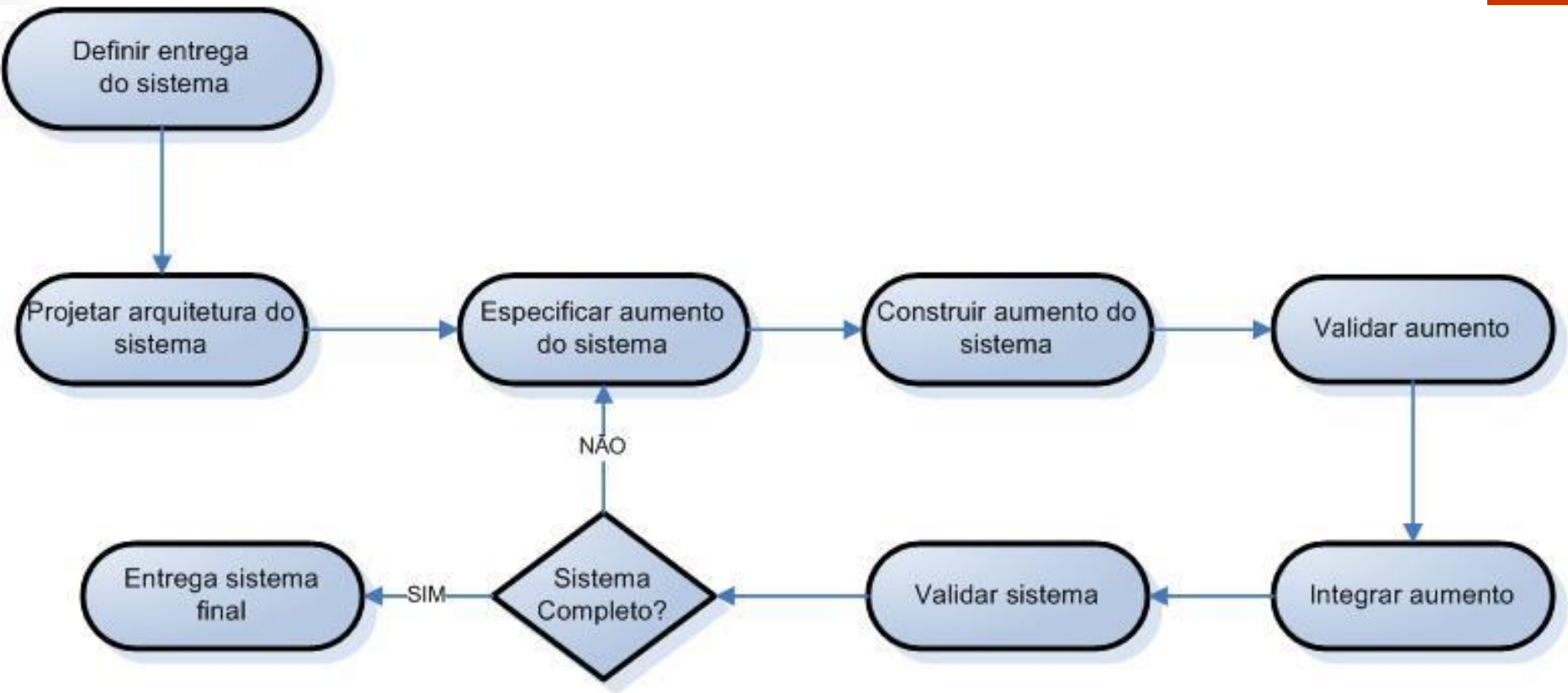
- Problemas
  - gerenciamento: não controle da documentação, difícil avaliação;
  - manutenção: corrupção da estrutura do sistema decorrente das mudanças, mudança de mantenedor, falta de documentos;
  - contratuais: Como cobrar pelo sistema? Por horas, o cliente fica insatisfeito. Por preço fixo não será viável para o desenvolvedor.

Prototipação Evolucionária

- Bom para o desenvolvedor
  - Codificação rápida
- Bom para o cliente
  - Versão desenvolvida rapidamente
- Ruim porquê...
  - Cliente não vê “remendos”

Prototipação Evolucionária

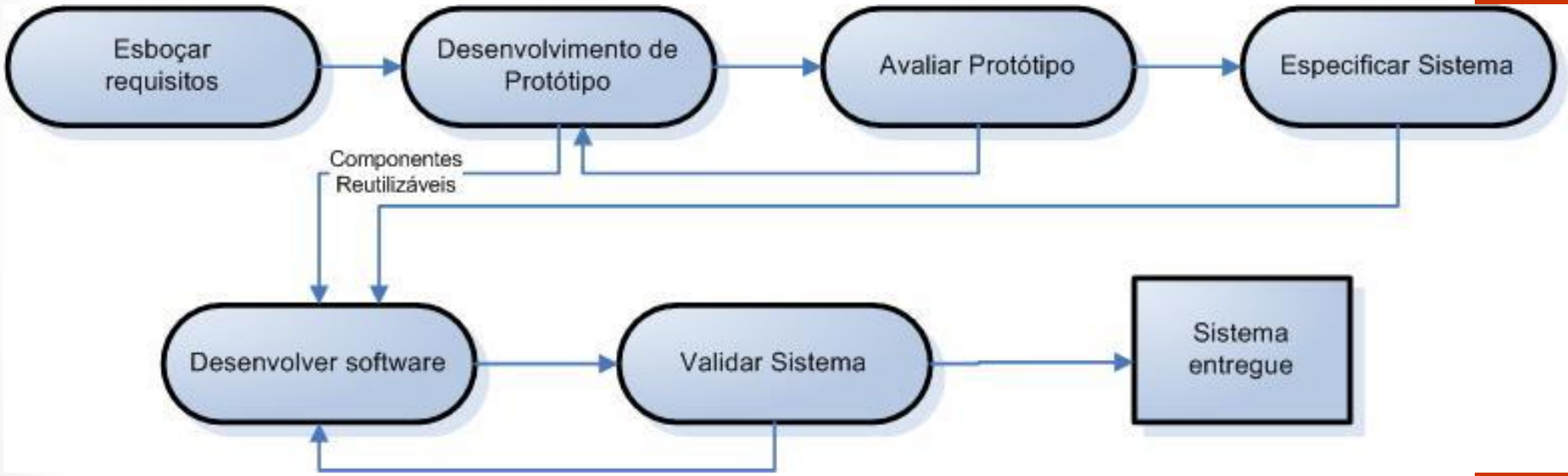
- Variação: Prototipação incremental



Prototipação Descartável

- Amplia o processo de análise de requisitos, com a intenção de reduzir os custos totais do ciclo de vida;
- O protótipo vem para esclarecer requisitos e produzir informações para os gerentes avaliarem o risco do projeto;
- Depois da avaliação, o protótipo é descartado, não sendo mais usado no desenvolvimento do sistema.

Prototipação Descartável



Prototipação Descartável

- Problemas
  - Diferença entre o protótipo e a implementação: falta de requisitos não funcionais, características faltando, falta de apoio legal (contrato);
  - Pressão para empurrar o protótipo: inconsistência do sistema, falta de documentação, baixa qualidade.

Problemas com a Prototipação

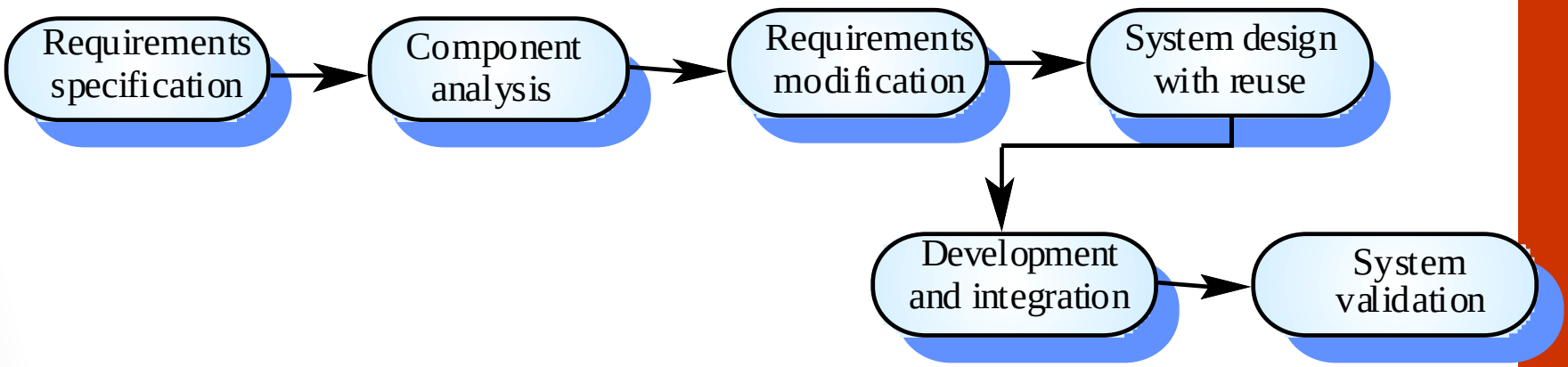
- Cliente não sabe que o software que ele vê não considerou, durante o desenvolvimento, a qualidade global e a manutenibilidade a longo prazo.
  - Não aceita a ideia que a versão final do software vai ser construída e "força" a utilização do protótipo como produto final
- Desenvolvedor frequentemente faz uma implementação comprometida (utilizando o que está disponível) com o objetivo de produzir rapidamente um protótipo.
  - Depois de um tempo ele familiariza com essas escolhas, e esquece que elas não são apropriadas para o produto final.
- Mesmo com problemas, a prototipação é um ciclo de vida eficiente.
- a chave é definir-se as regras do jogo logo no começo.
- o cliente e o desenvolvedor devem ambos concordar que o protótipo seja construído para servir como um mecanismo a fim de definir os requisitos

Engenharia de software baseada em componentes

- **Baseado em reuso sistemático onde sistemas são integrados a partir de componentes existentes ou de sistemas COTS (Commercial-of-the-shelf)**
- **Estágios do processo**
  - **Análise de componentes;**
  - **Modificação de requisitos;**
  - **Projeto de sistema com reuso;**
  - **Desenvolvimento e integração.**

- **Esta abordagem está se tornando cada vez mais usada à medida que padrões de componentes têm surgido.**
- **Reuso acidental vs. Reuso planejado**

Desenvolvimento orientado ao reuso



Processos Iterativos

- **Requisitos de sistema SEMPRE evoluem no curso de um projeto**
- **Algum retrabalho é necessário**
- **A abordagem iterativa pode ser aplicada a qualquer um dos modelos genéricos do processo.**
- **Duas abordagens (relacionadas)**
  - **Entrega incremental;**
  - **Desenvolvimento espiral.**

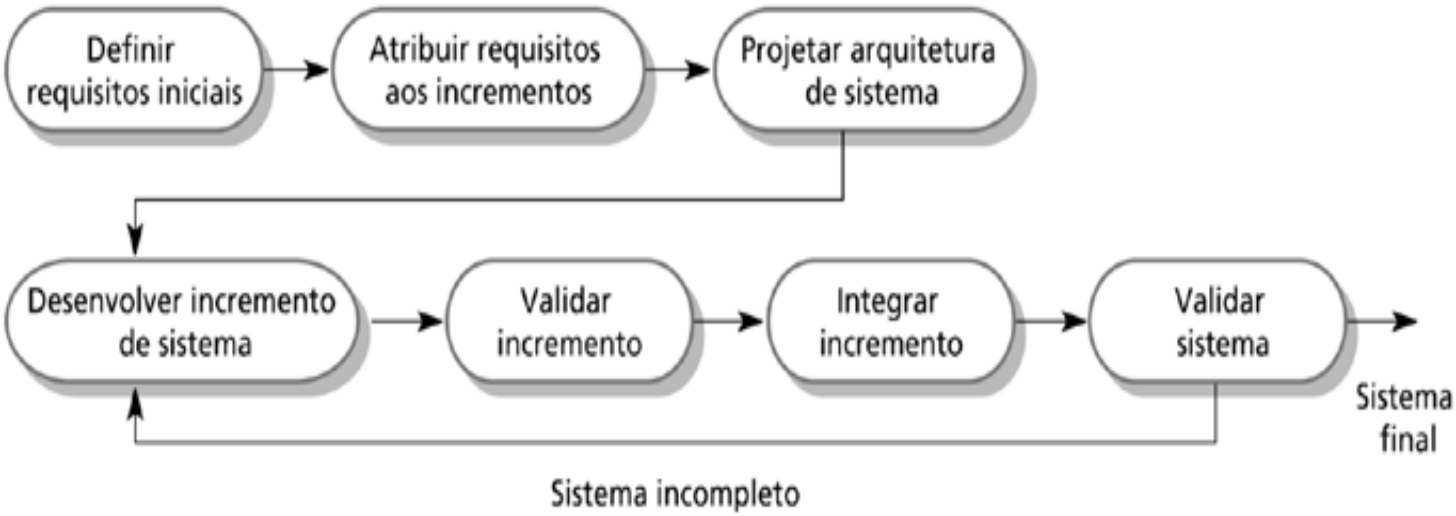
Entrega incremental

- **O sistema é entregue ao cliente em incrementos**
  - **Cada incremento fornece parte da funcionalidade**
- **Os requisitos são priorizados**
  - **Requisitos de prioridade mais alta são incluídos nos incrementos iniciais**
- **Uma vez que o desenvolvimento de um incremento é iniciado, os requisitos são congelados**
  - **Os requisitos para os incrementos posteriores podem continuar evoluindo (e incluir requisitos já implementados!)**

Desenvolvimento incremental

Figura 4.4

Entrega incremental.



Vantangens do desenvolvimento incremental

- **Incrementos podem ser entregues regularmente ao cliente e, desse modo, a funcionalidade de sistema é disponibilizada mais cedo.**
- **Os incrementos iniciais agem como protótipos para elicitar os requisitos para incrementos posteriores do sistema.**
- **Riscos menores de falha geral do projeto.**
- **Os serviços de sistema de mais alta prioridade tendem a receber mais testes.**

Metodologias Ágeis

Desenvolvimento rápido de software

- A entrega e o desenvolvimento rápidos têm sido o requisito mais importante nos sistemas de software:
  - Os negócios operam com requisitos que mudam rapidamente e é praticamente impossível produzir um conjunto estável de requisitos de software.
  - O software precisa evoluir rapidamente para refletir as necessidades de negócio em constante mudança.
- Objetivo dos métodos ágeis é reduzir o overhead nos processos de software (ex. limitando a documentação) e permitir uma resposta rápida aos requisitos em constante mudança sem retrabalho excessivo.
- Especificação, projeto, implementação e teste são intercalados e os produtos do processo de desenvolvimento são decididos através de um processo de negociação, durante o processo de desenvolvimento do software.

Os princípios dos métodos ágeis

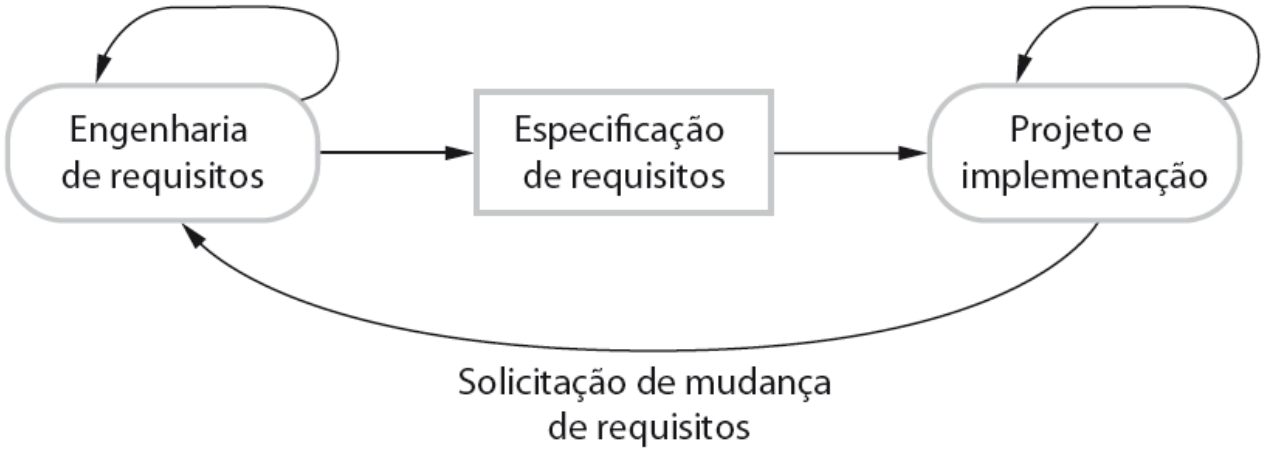
Princípios	Descrição
Envolvimento do cliente	Os clientes devem estar intimamente envolvidos no processo de desenvolvimento. Seu papel é fornecer e priorizar novos requisitos do sistema e avaliar suas iterações.
Entrega incremental	O software é desenvolvido em incrementos com o cliente, especificando os requisitos para serem incluídos em cada um.
Pessoas, não processos	As habilidades da equipe de desenvolvimento devem ser reconhecidas e exploradas. Membros da equipe devem desenvolver suas próprias maneiras de trabalhar, sem processos prescritivos.
Aceitar as mudanças	Deve-se ter em mente que os requisitos do sistema vão mudar. Por isso, projete o sistema de maneira a acomodar essas mudanças.
Manter a simplicidade	Focalize a simplicidade, tanto do software a ser desenvolvido quanto do processo de desenvolvimento. Sempre que possível, trabalhe ativamente para eliminar a complexidade do sistema.

Aplicabilidade dos métodos ágeis

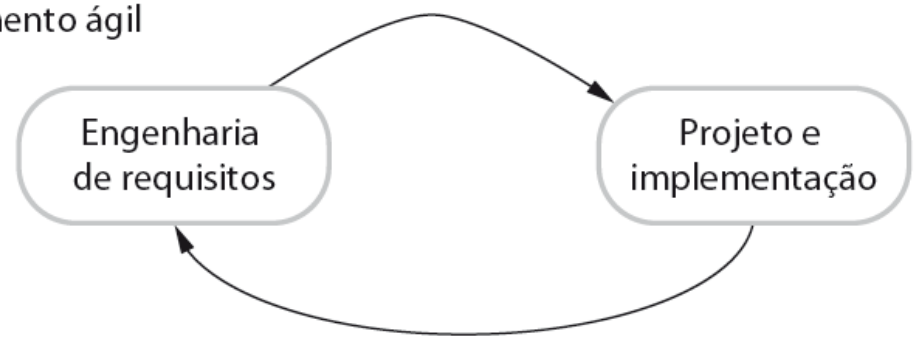
- Desenvolvimento de produto, quando a empresa de software está desenvolvendo um produto pequeno ou médio para venda.
- Desenvolvimento de sistema personalizado dentro de uma organização, quando existe um compromisso claro do cliente em se envolver no processo de desenvolvimento e quando não existem muitas regras e regulamentos externos que afetam o software.
- Devido ao foco em equipes pequenas e fortemente integradas, existem problemas na escalabilidade de métodos ágeis em sistemas grandes.

Especificações dirigida a planos e ágil

Desenvolvimento baseado em planos



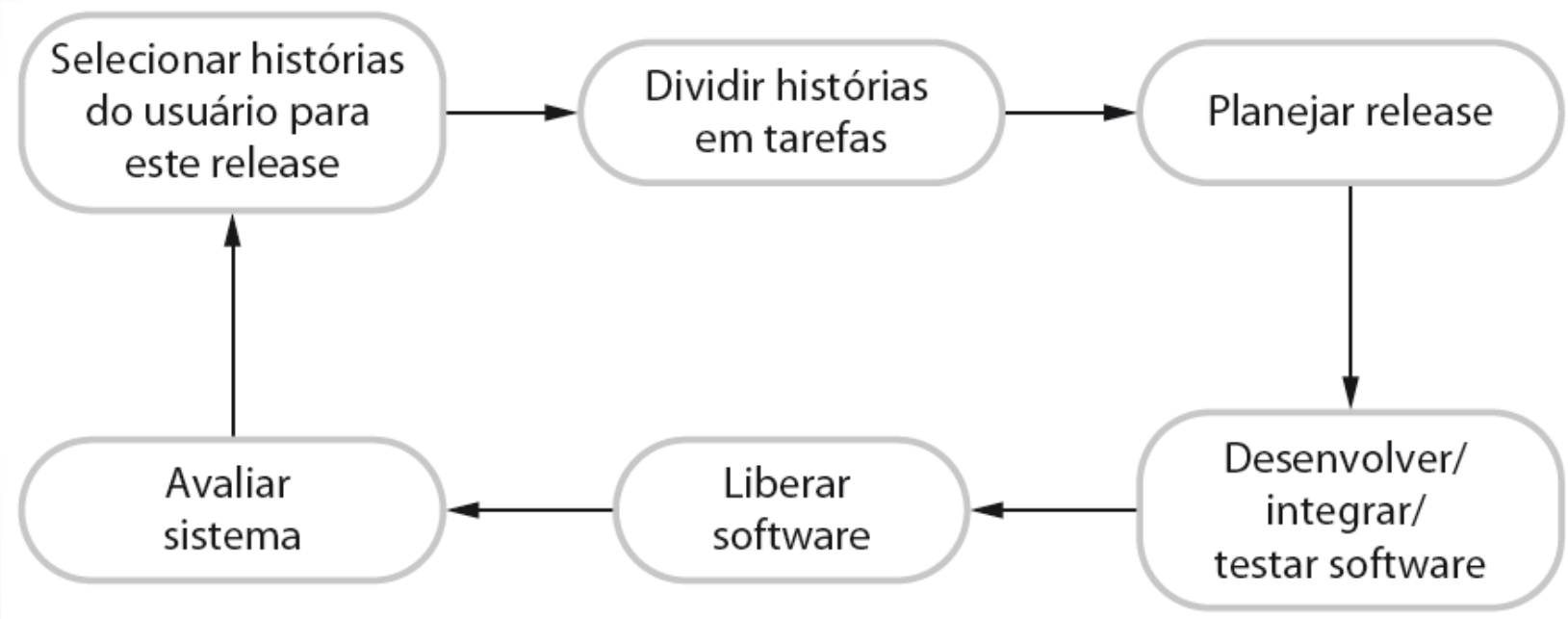
Desenvolvimento ágil



Programação extrema

- Método ágil mais conhecido e usado
- Baseado no desenvolvimento e entrega de incrementos de funcionalidade bem pequenos
- Abordagem “extrema” de desenvolvimento iterativo:
  - Novas versões construídas várias vezes por dia
  - Incrementos entregues ao cliente a cada 2 semanas
  - Testes em todas versões e a versão só é aceita se os testes forem concluídos com sucesso.
- Conta com melhoramento constante do código, envolvimento do usuário no time de desenvolvimento e programação em pares

O ciclo de um release em Extreme Programming



Práticas da Extreme Programming

Princípio ou prática	Descrição
Planejamento incremental	Os requisitos são gravados em cartões de história e as histórias que serão incluídas em um release são determinadas pelo tempo disponível e sua relativa prioridade. Os desenvolvedores dividem essas histórias em 'Tarefas'. Veja os quadros 3.1 e 3.2.
Pequenos *releases*	Em primeiro lugar, desenvolve-se um conjunto mínimo de funcionalidades útil, que fornece o valor do negócio. *Releases* do sistema são frequentes e gradualmente adicionam funcionalidade ao primeiro *release*.
Projeto simples	Cada projeto é realizado para atender às necessidades atuais, e nada mais.
Desenvolvimento *test-first*	Um *framework* de testes iniciais automatizados é usado para escrever os testes para uma nova funcionalidade antes que a funcionalidade em si seja implementada.
Refatoração	Todos os desenvolvedores devem refatorar o código continuamente assim que encontrarem melhorias de código. Isso mantém o código simples e manutenível.

Práticas da Extreme Programming

Princípio ou prática	Descrição
Programação em pares	Os desenvolvedores trabalham em pares, verificando o trabalho dos outros e prestando apoio para um bom trabalho sempre.
Propriedade coletiva	Os pares de desenvolvedores trabalham em todas as áreas do sistema, de modo que não se desenvolvam ilhas de *expertise*. Todos os conhecimentos e todos os desenvolvedores assumem responsabilidade por todo o código. Qualquer um pode mudar qualquer coisa.
Integração contínua	Assim que o trabalho em uma tarefa é concluído, ele é integrado ao sistema como um todo. Após essa integração, todos os testes de unidade do sistema devem passar.
Ritmo sustentável	Grandes quantidades de horas-extra não são consideradas aceitáveis, pois o resultado final, muitas vezes, é a redução da qualidade do código e da produtividade a médio prazo.
Cliente no local	Um representante do usuário final do sistema (o cliente) deve estar disponível todo o tempo à equipe de XP. Em um processo de Extreme Programming, o cliente é um membro da equipe de desenvolvimento e é responsável por levar a ela os requisitos de sistema para implementação.

Problemas com métodos ágeis

- Pode ser difícil manter o interesse dos clientes que estão envolvidos no processo.
- Membros da equipe podem não ser adequados ao envolvimento intenso que caracteriza os métodos ágeis.
- Priorizar mudanças pode ser difícil onde existem múltiplos stakeholders.
- Manter a simplicidade requer trabalho extra.
- Os contratos podem ser um problema assim como em outras abordagens que usam o desenvolvimento iterativo.
- Escalamento de métodos ágeis para sistemas de grande porte é difícil. Tais sistemas precisam de mais projeto inicial e alguma documentação.

Desenvolvimento espiral

- **O processo é representado como uma espiral ao invés de uma seqüência de atividades com realimentação.**
- **Cada loop na espiral representa uma fase no processo.**
- **Sem fases definidas, tais como especificação ou projeto – os loops na espiral são escolhidos dependendo do que é requisitado.**
- **Os riscos são explicitamente avaliados e resolvidos ao longo do processo.**

Ciclo de Vida em Espiral

- engloba as melhores características do ciclo de vida Clássico e da Prototipação, adicionando um novo elemento: a Análise de Risco
- segue a abordagem de passos sistemáticos do Ciclo de Vida Clássico incorporando-os numa estrutura iterativa que reflete mais realisticamente o mundo real
- usa a Prototipação, em qualquer etapa da evolução do produto, como mecanismo de redução de riscos

Modelo espiral do processo de software

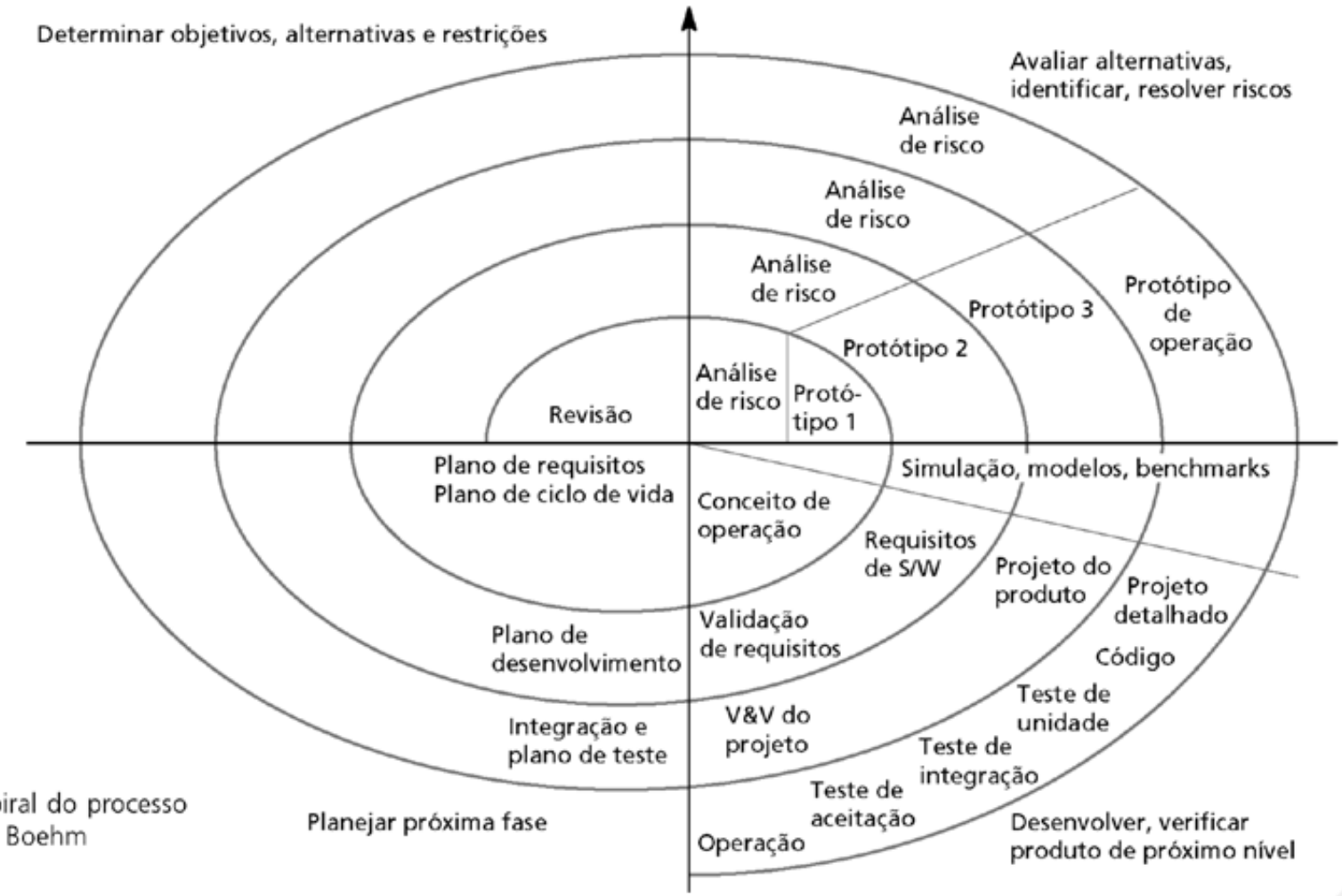


Figura 4.5

Modelo em espiral do processo de software de Boehm (©*IEEE*,1988).

Setores do modelo espiral

- **Definição de objetivos**
  - **Objetivos específicos para a fase são identificados.**
- **Avaliação e redução de riscos**
  - **Riscos são avaliados e atividades são realizadas para reduzir os riscos-chave.**
- **Desenvolvimento e validação**
  - **Um modelo de desenvolvimento para o sistema, que pode ser qualquer um dos modelos genéricos, é escolhido.**
- **Planejamento**
  - **O projeto é revisado e a próxima fase da espiral é planejada.**
- **Processo de Desenvolvimento vs. Processo de Gerenciamento**

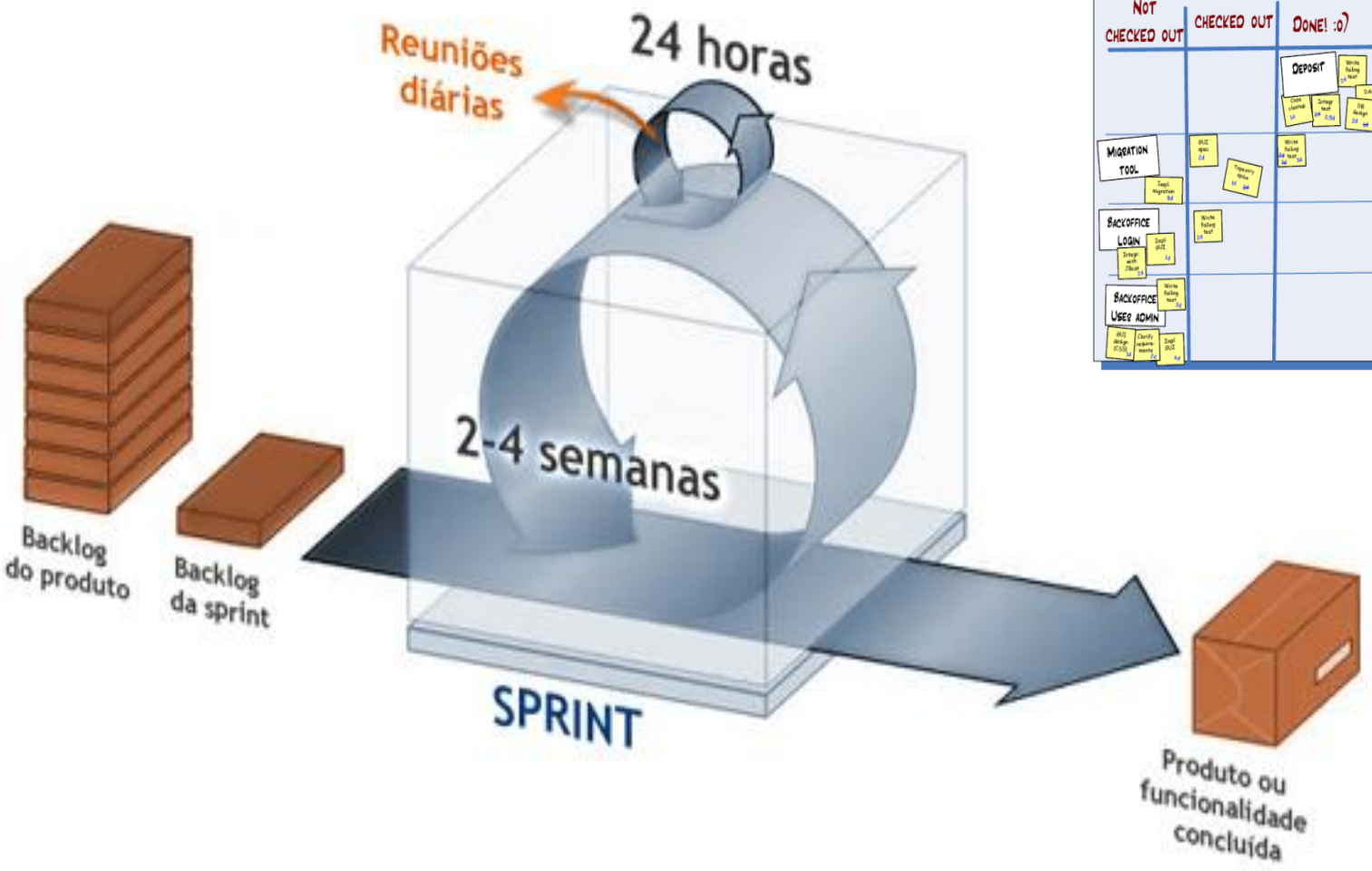
Comentários sobre o Ciclo de Vida em Espiral

- Modelo relativamente novo e somente agora tem sido amplamente usado.
- Leva um tempo até que a eficácia desse modelo possa ser percebida com certeza absoluta.
- Abordagem mais realística para o desenvolvimento de software em grande escala.
- Capacita o desenvolvedor e o cliente a entender e reagir aos riscos em cada etapa evolutiva.
- Pode ser difícil convencer os clientes que uma abordagem "evolutiva" é controlável
- Exige considerável experiência na determinação de riscos e depende dessa experiência para ter sucesso

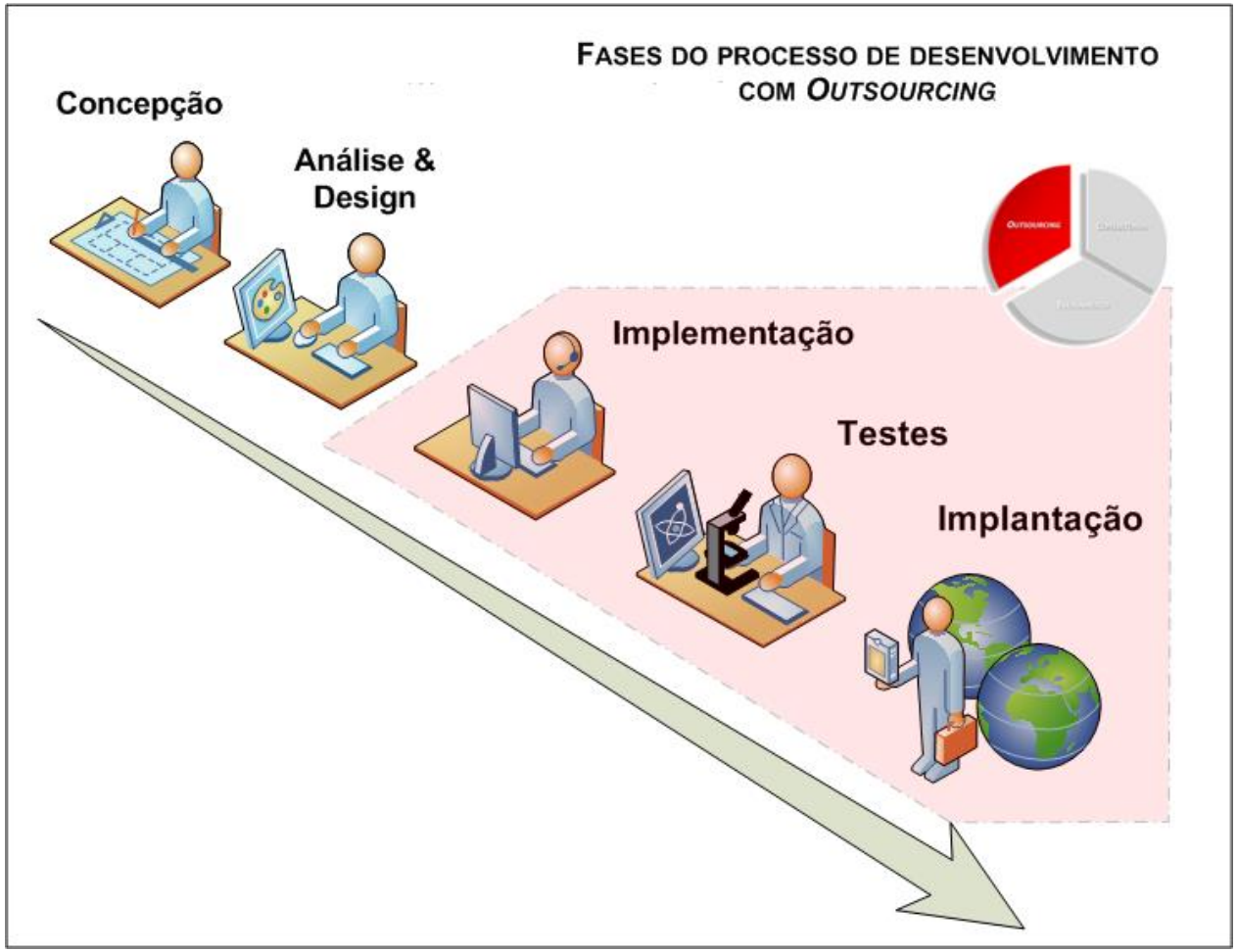
ALGUNS MODELOS USADOS NA PRÁTICA

Formato de Fábrica de Software

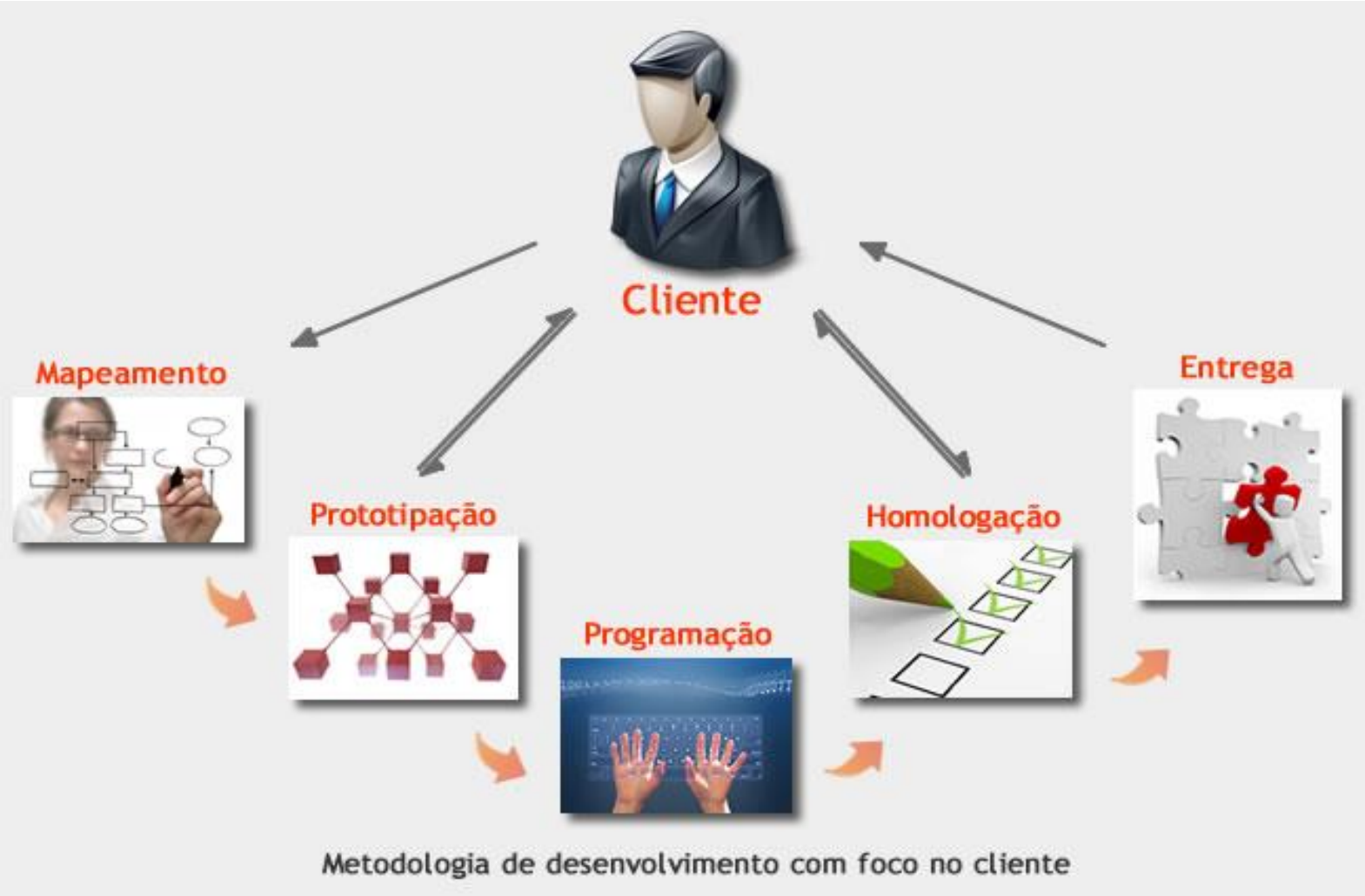
Agil com Sprints



Desenvolvendo com outsourcing



Metodologia com foco no cliente



Metodologia de desenvolvimento com foco no cliente

PROCESSO UNIFICADO (RUP)

(Rational) Unified Process

- **É um ("modelo de"?) processo moderno baseado na UML**
  - **Tenta cobrir todos os aspectos do desenvolvimento de software**
- **Fortemente focado na documentação do sistema**
- **Normalmente descrito a partir de três perspectivas:**
  - **Uma perspectiva dinâmica que mostra as fases ao longo do tempo;**
  - **Uma perspectiva estática que mostra atividades de processo;**
  - **Uma perspectiva prática que sugere bons princípios e práticas de desenvolvimento**

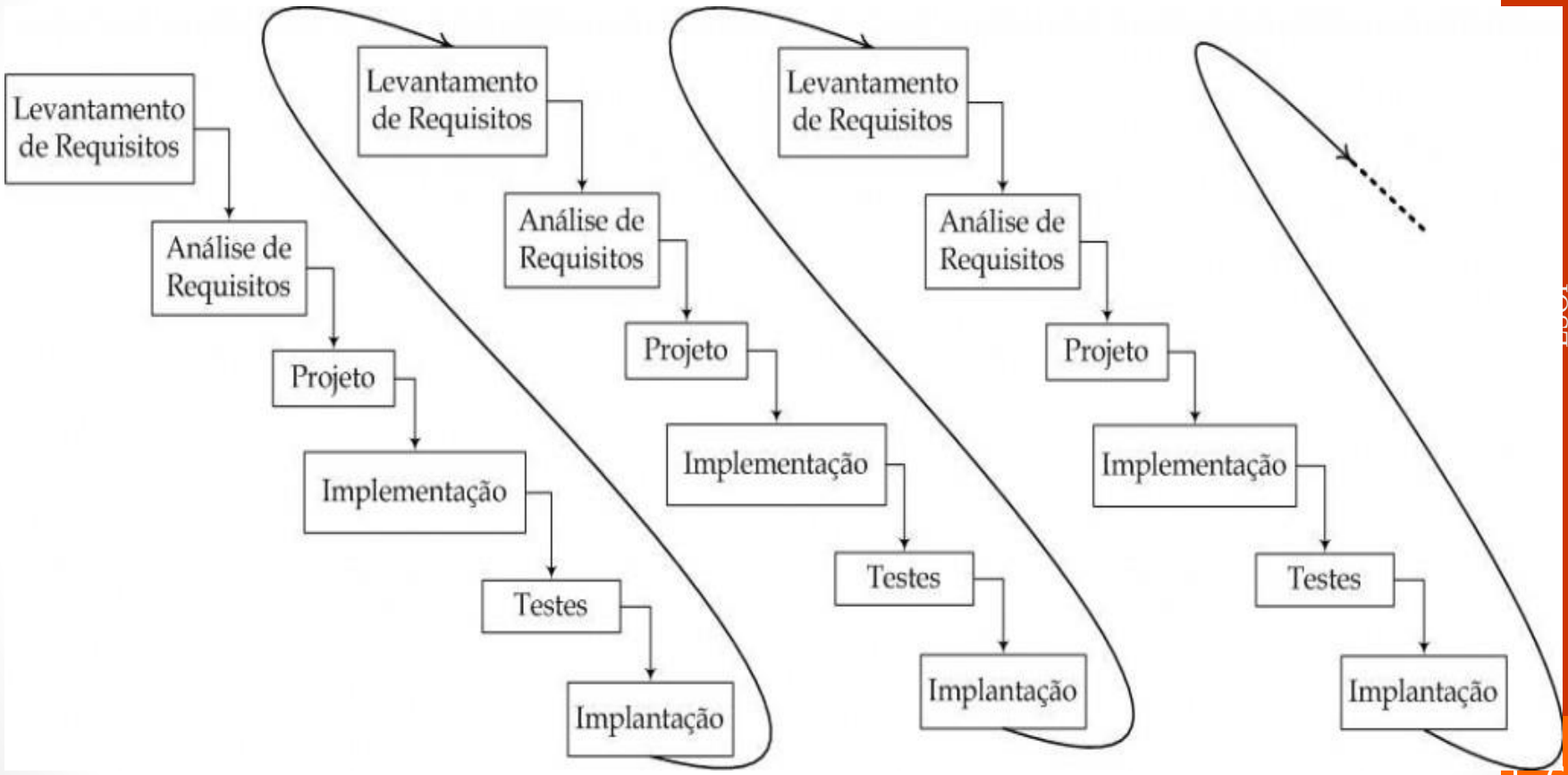
Boas práticas do RUP

- **Desenvolver o software iterativamente**
- **Gerenciar requisitos**
- **Usar arquiteturas baseadas em componentes**
- **Modelar o software visualmente**
- **Verificar a qualidade de software**
- **Controlar as mudanças do software**

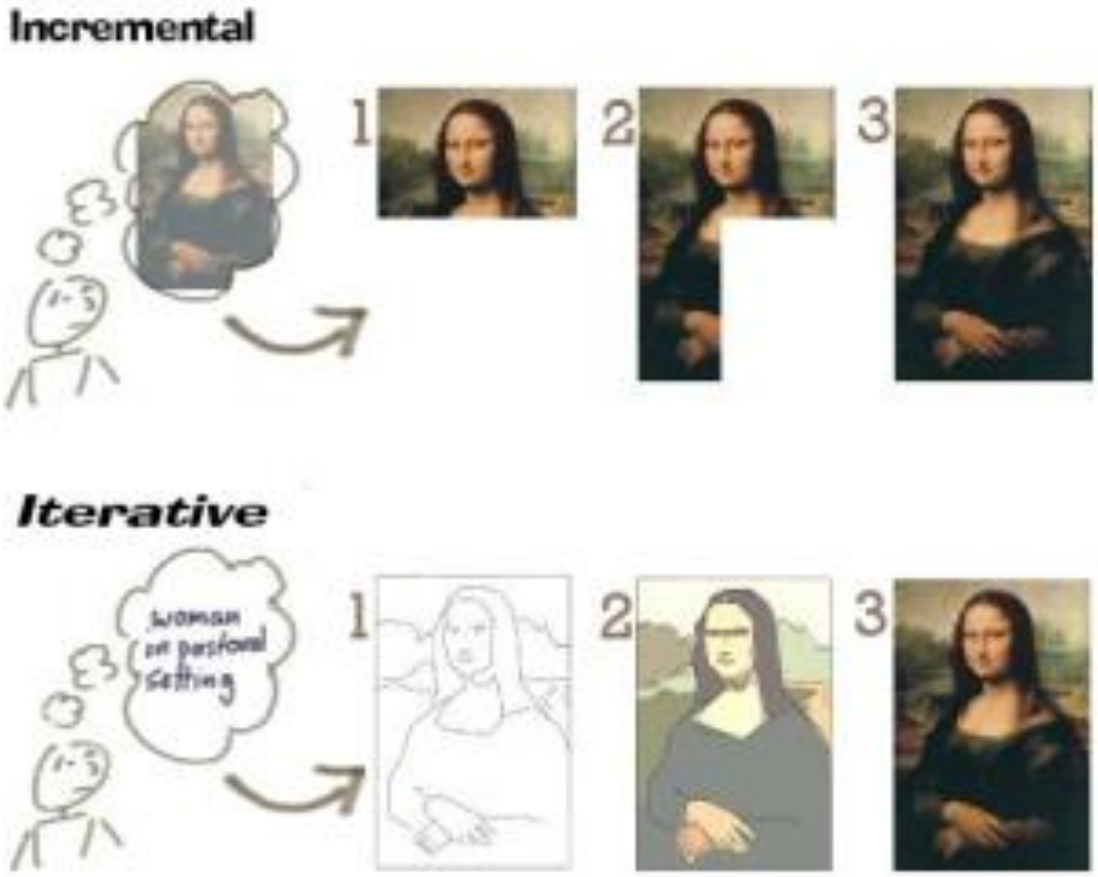
Características Principais do RUP

- O desenvolvimento de sistemas seguindo o RUP é
  - Iterativo e incremental
  - Guiado por casos de uso (use cases)
  - Baseado na arquitetura do sistema

Iterativo e Incremental

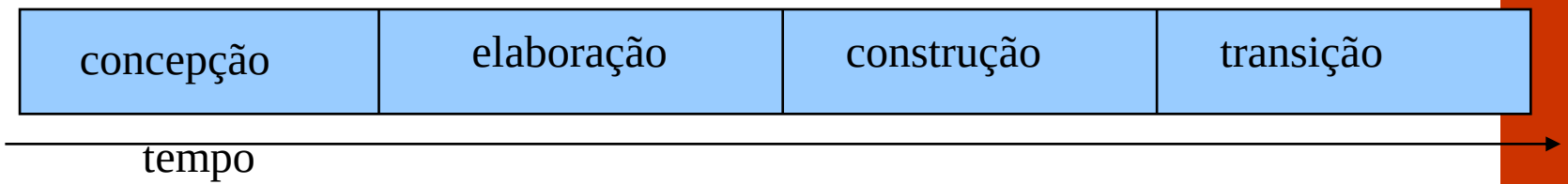


Iterativo e Incremental



O RUP é iterativo e incremental

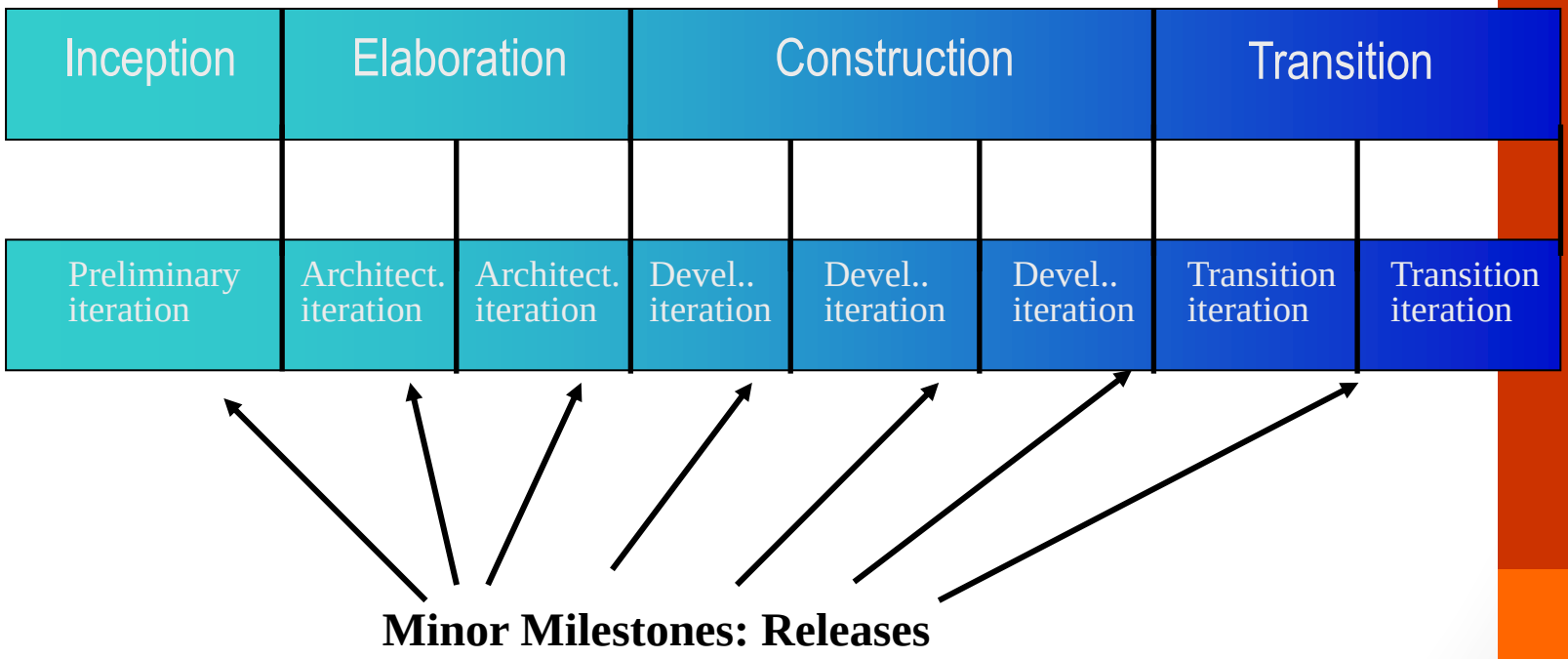
- O ciclo de vida de um sistema consiste de quatro fases:



- Concepção (define o escopo do projeto)
- Elaboração (detalha os requisitos e a arquitetura)
- Construção (desenvolve o sistema)
- Transição (implanta o sistema)

O RUP é iterativo e incremental

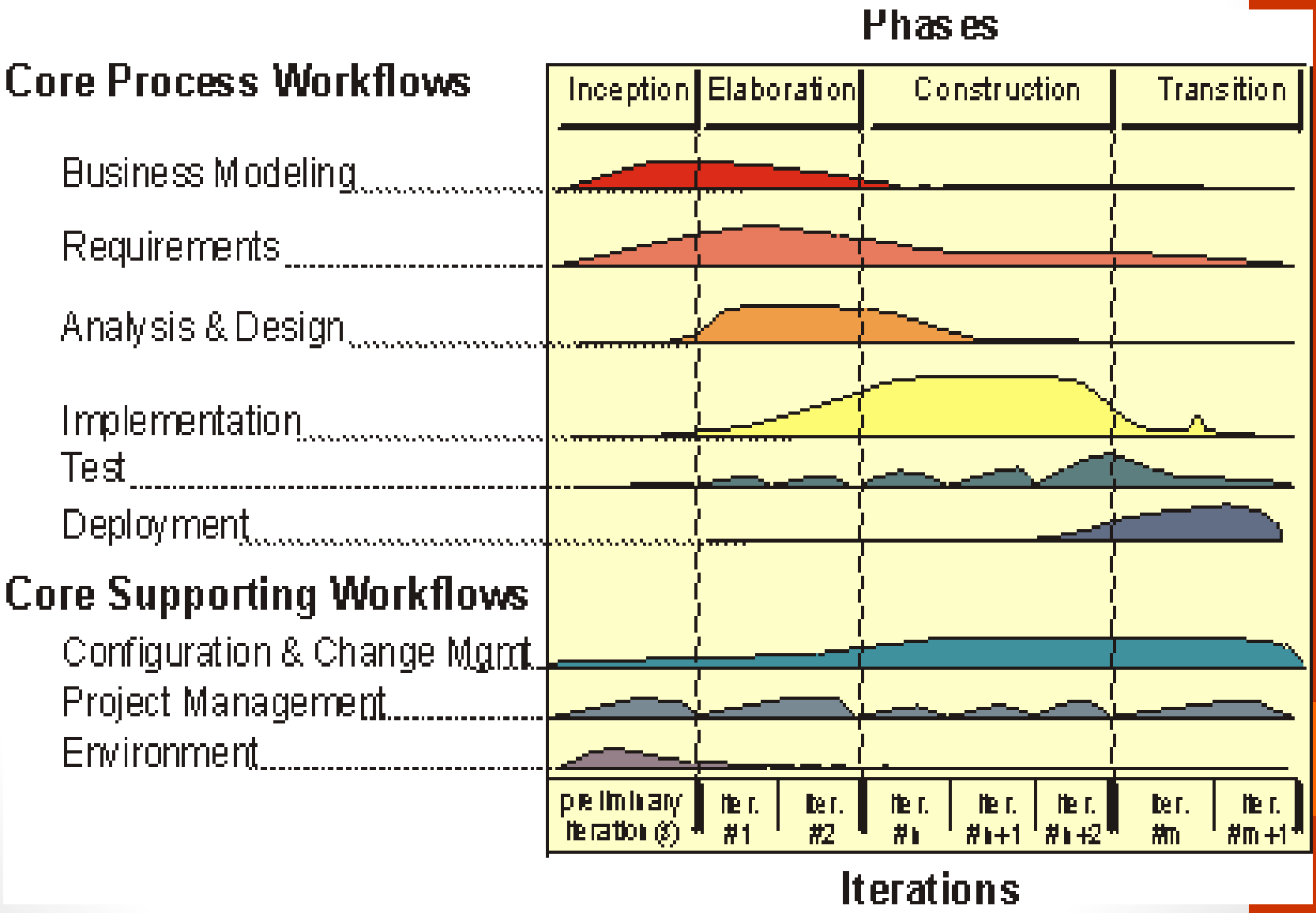
- Cada fase é dividida em iterações:



O RUP é iterativo e incremental

- Cada iteração
  - é planejada
  - realiza uma seqüência de atividades (de elicitação de requisitos, análise e projeto, implementação, etc.) distintas
  - geralmente resulta em uma versão executável do sistema
  - é avaliada segundo critérios de sucesso previamente definidos

O RUP é iterativo e incremental



O RUP é guiado por casos de uso

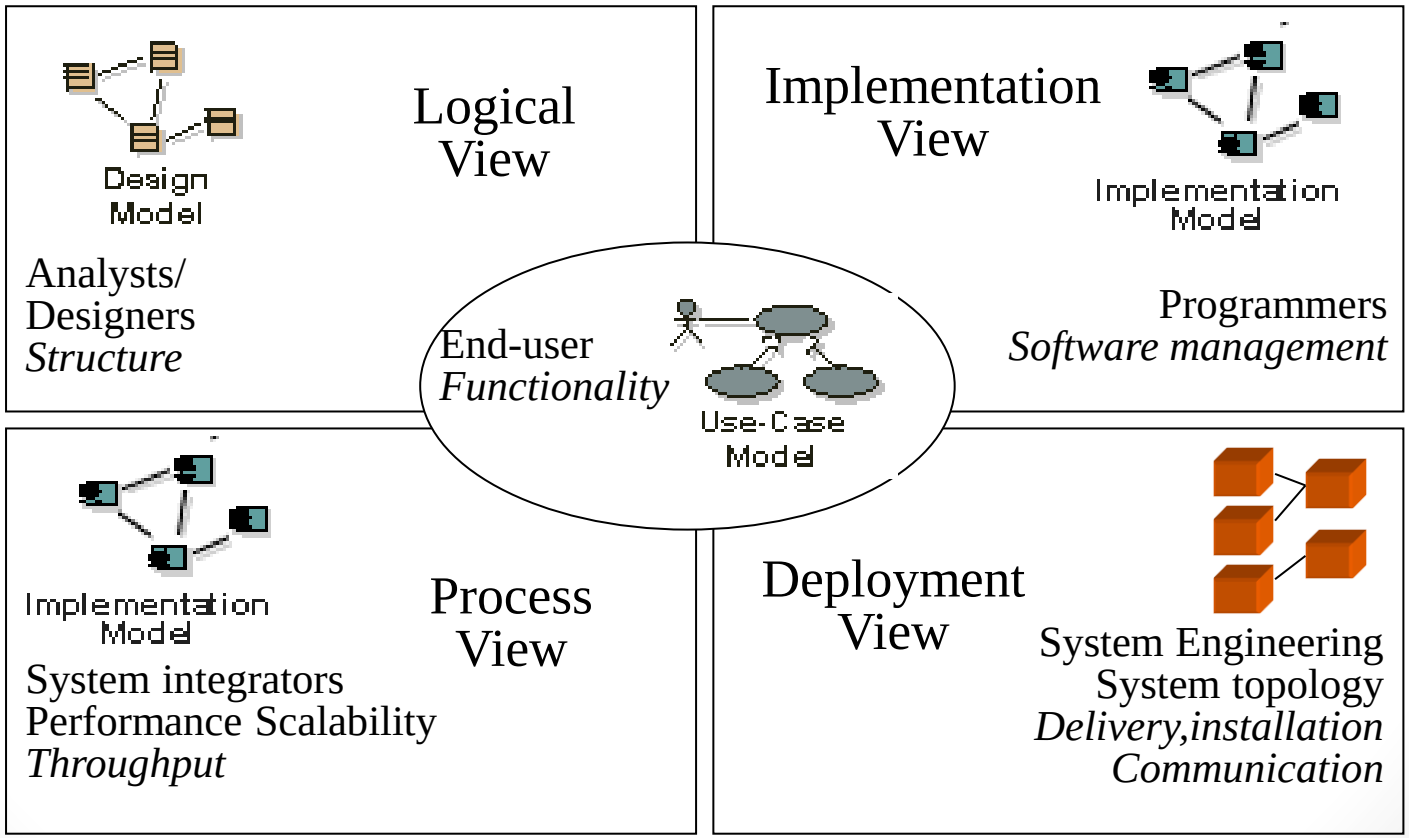
- Os casos de uso não servem apenas para definir os requisitos do sistema
- Várias atividades do RUP são guiadas pelos casos de uso:
  - planejamento das iterações
  - criação e validação do modelo de projeto
  - planejamento da integração do sistema
  - definição dos casos de teste

O RUP é baseado na arquitetura do sistema

- Arquitetura
  - visão geral do sistema em termos dos seus subsistemas e como estes se relacionam
- A arquitetura é prototipada e definida logo nas primeiras iterações
- O desenvolvimento consiste em complementar a arquitetura
- A arquitetura serve para definir a organização da equipe de desenvolvimento e identificar oportunidades de reuso

O RUP é baseado na arquitetura do sistema

- Idealmente, tem-se 5 visões da arquitetura



Organização do RUP

- Fluxos de atividades
- Atividades
  - passos
  - entradas e saídas
  - guias (de ferramentas ou não), *templates*
- Responsáveis (papel e perfil, não pessoa)
- Artefatos

Fases do RUP

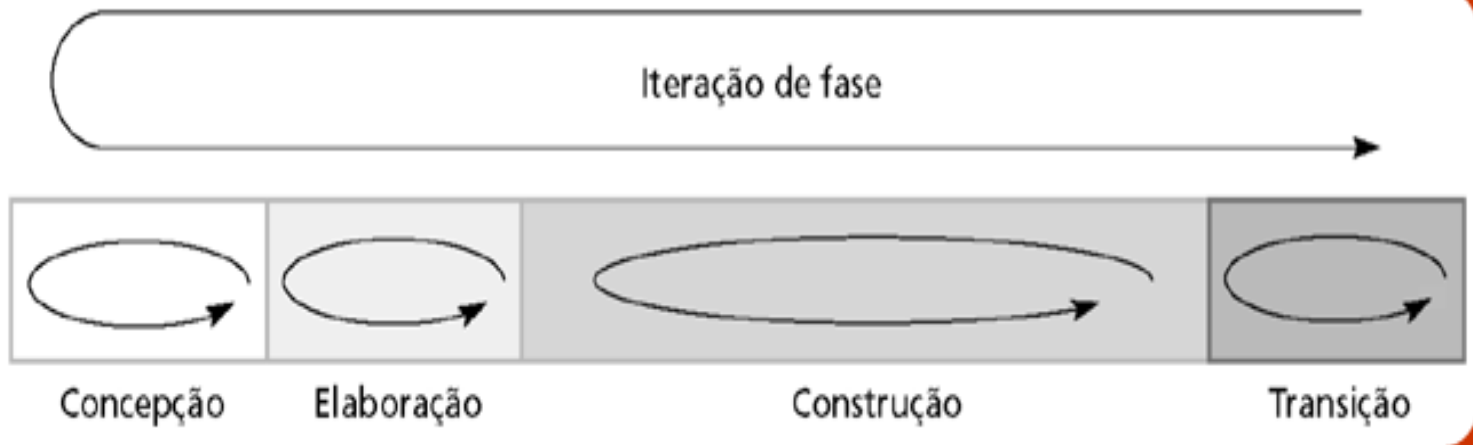
- **Concepção**
  - **Estabelecer o business case para o sistema.**
- **Elaboração**
  - **Desenvolver um entendimento do domínio do problema e a arquitetura do sistema.**
- **Construção**
  - **Projeto, programação e teste de sistema.**
- **Transição**
  - **Implantar o sistema no seu ambiente operacional.**

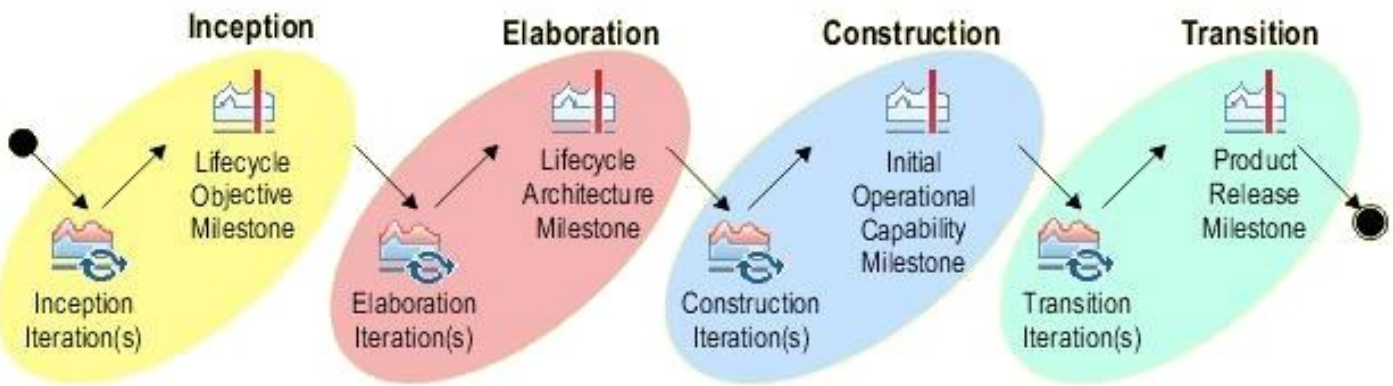
Modelo de fases do RUP

- **Centrado no gerenciamento de projetos**

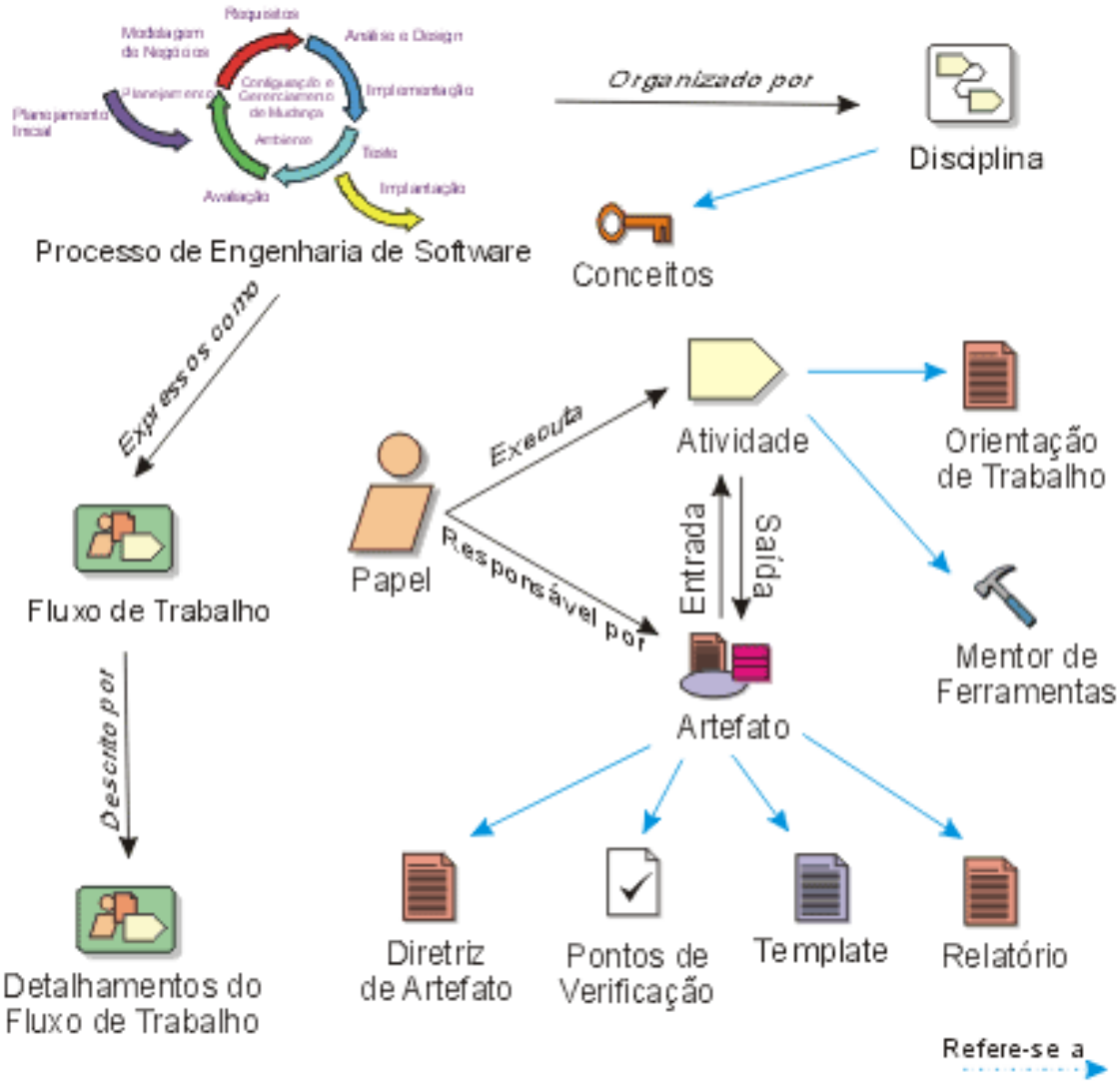
Figura 4.12

Fases no Rational Unified Process.





Conceitos do RUP

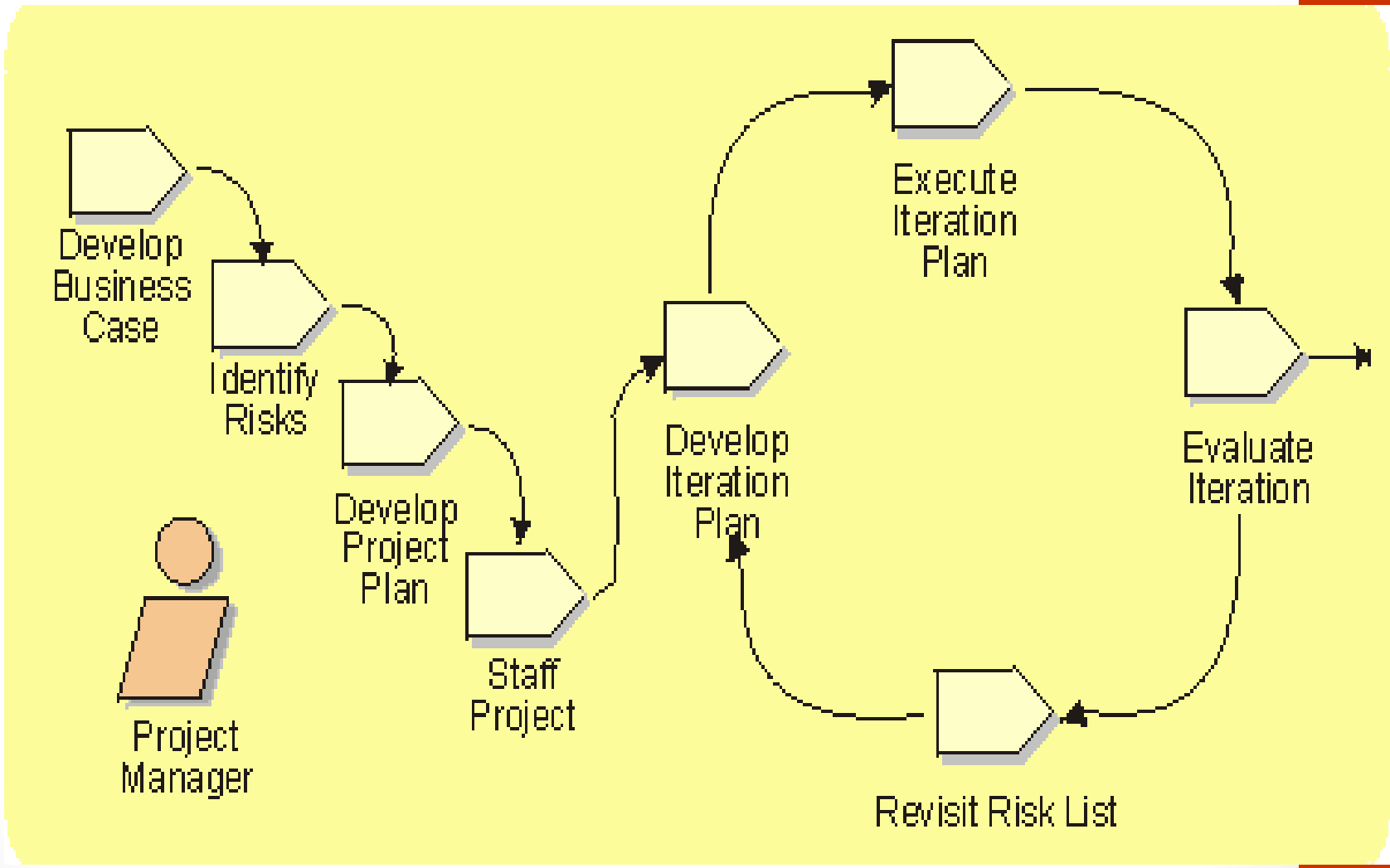


Workflows estáticos

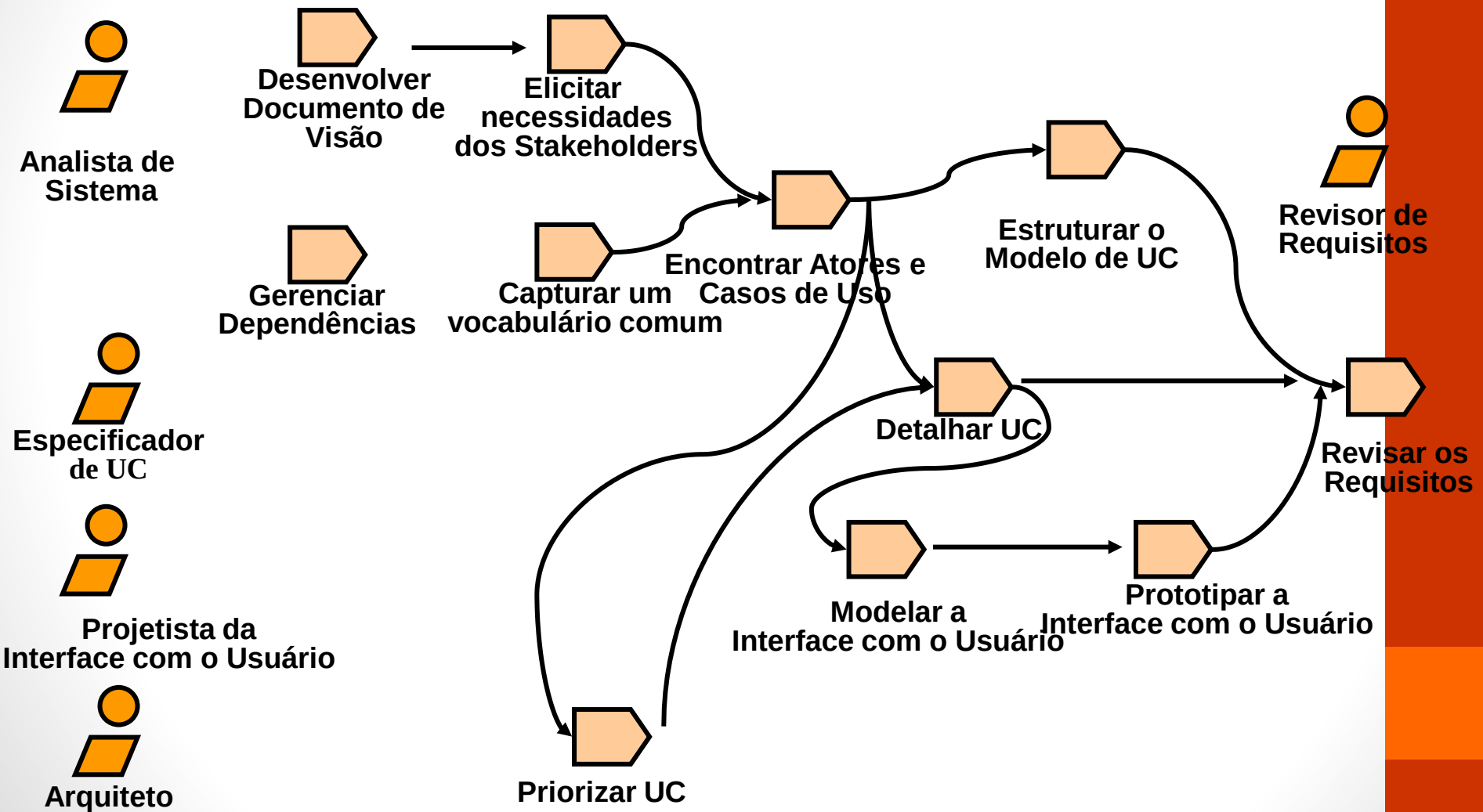
Tabela 4.1 Workflows estáticos no Rational Unified Process.

Workflow	Descrição
Modelagem de negócios	Os processos de negócios são modelados usando casos de uso de negócios.
Requisitos	Os agentes que interagem com o sistema são identificados e os casos de uso são desenvolvidos para modelar os requisitos de sistema.
Análise e projeto	Um modelo de projeto é criado e documentado usando modelos de arquitetura, modelos de componente, modelos de objeto e modelos de seqüência.
Implementação	Os componentes de sistema são implementados e estruturados em subsistemas de implementação. A geração automática de código com base nos modelos de projeto ajuda a acelerar esse processo.
Teste	O teste é um processo iterativo realizado em conjunto com a implementação. O teste de sistema segue o término da implementação.
Implantação	Uma versão do produto é criada, distribuída aos usuários e instalada no local de trabalho.
Gerenciamento de configuração e mudanças	Este workflow de apoio gerencia as mudanças do sistema (veja o Capítulo 29).
Gerenciamento de projetos	Este workflow de apoio gerencia o desenvolvimento do sistema (veja o Capítulo 5).
Ambiente	Este workflow está relacionado à disponibilização de ferramentas apropriadas de software para a equipe de desenvolvimento.

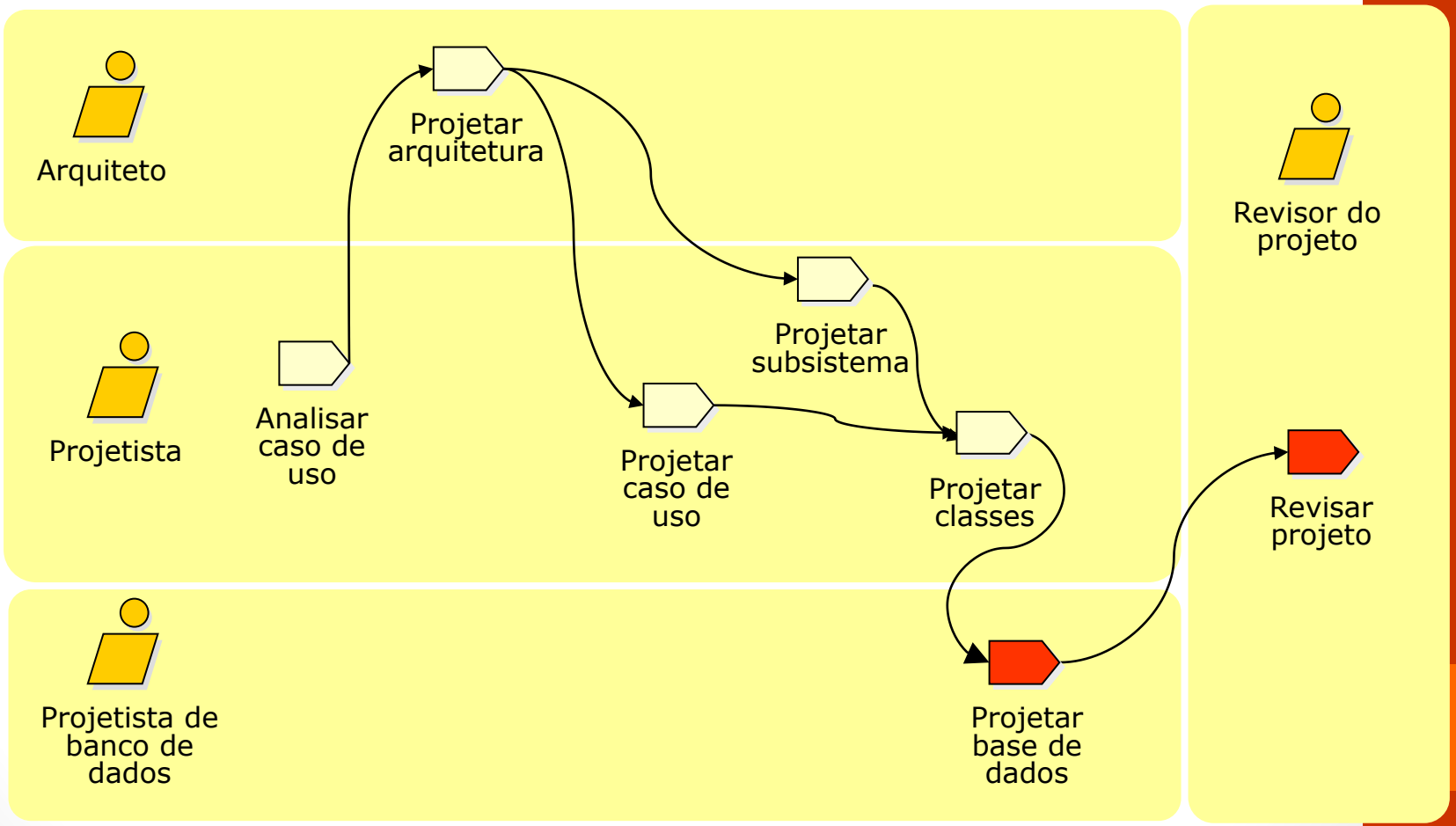
Planejamento e Gerenciamento



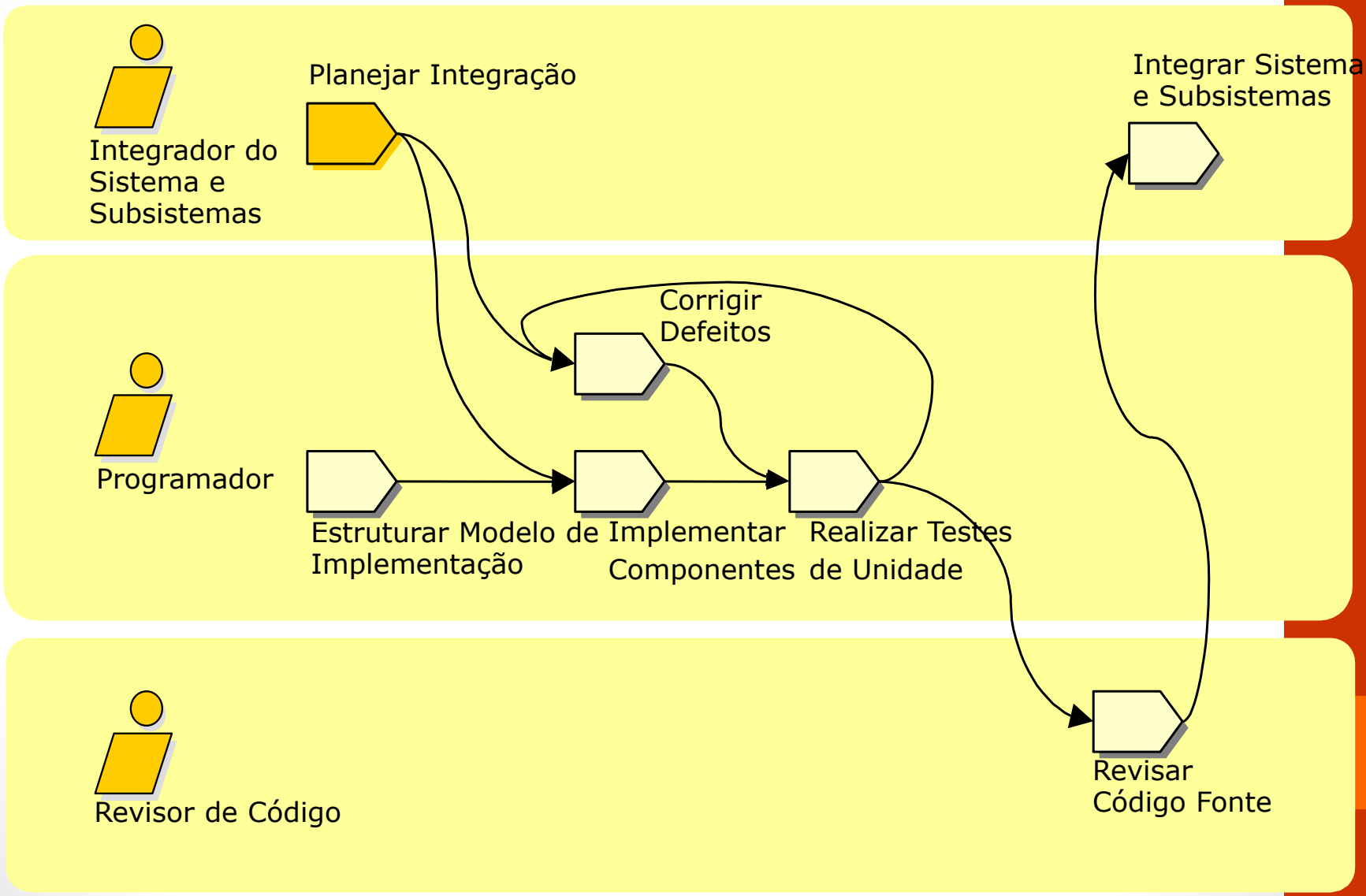
Elicitação de Requisitos



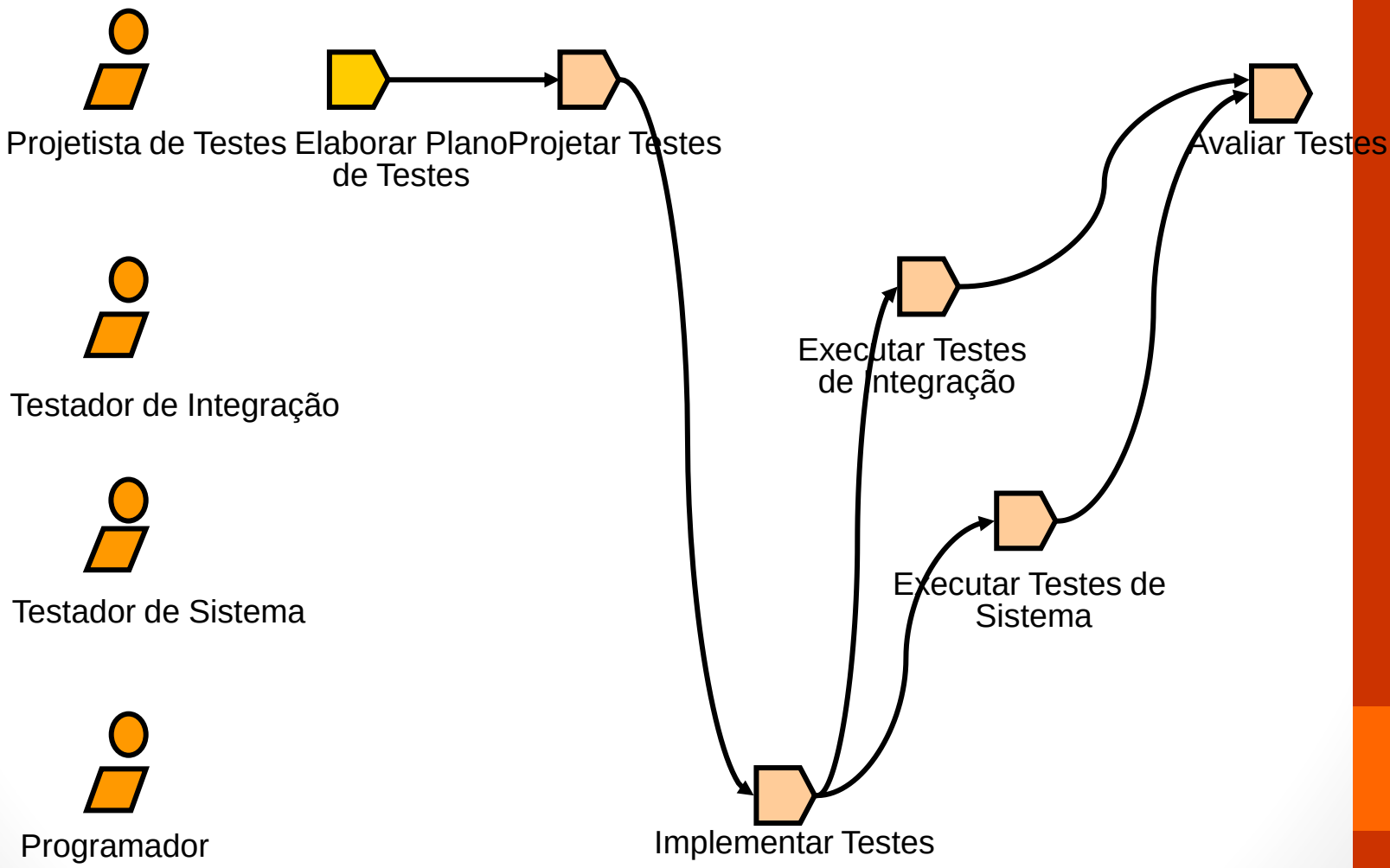
Análise e Projeto (Simplificado)



Implementação



Testes


Projetista de Testes
Elaborar Plano de Testes
Projetar Testes
Avaliar Testes
Testador de Integração
Executar Testes de Integração
Testador de Sistema
Executar Testes de Sistema
Programador
Implementar Testes

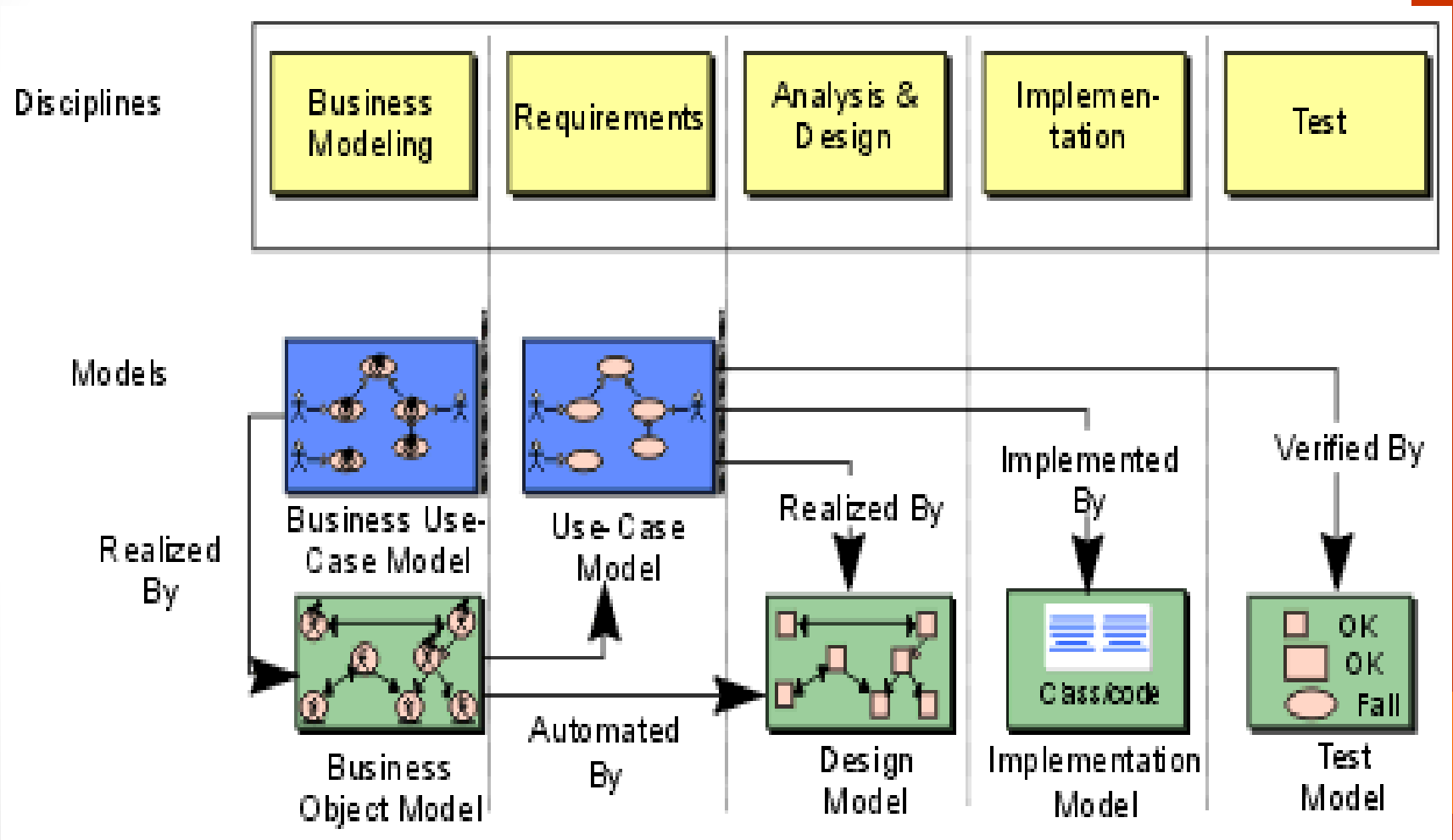
Resumo

- O RUP é:
- iterativo e incremental
- guiado por casos de uso
- baseado na arquitetura do sistema
- organizado em fases, iterações, fluxos, atividades e passos

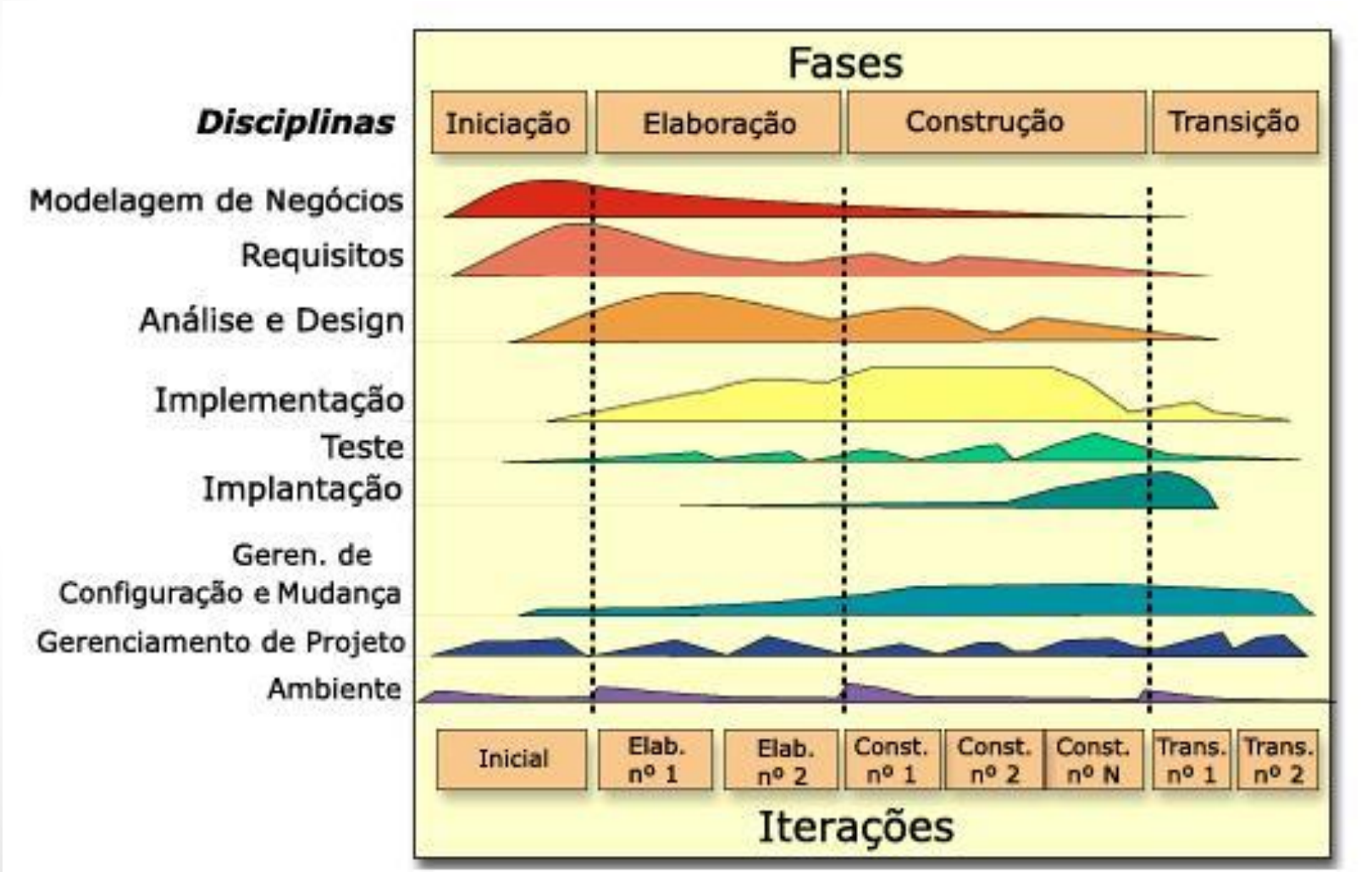
Referências

- Ivar Jacobson, Grady Booch e James Rumbaugh. T*he Unified Software Development Process.* Capítulos 1 a 5.
- Philippe Kruchten. *The Rational Unified Process – an Introduction.*

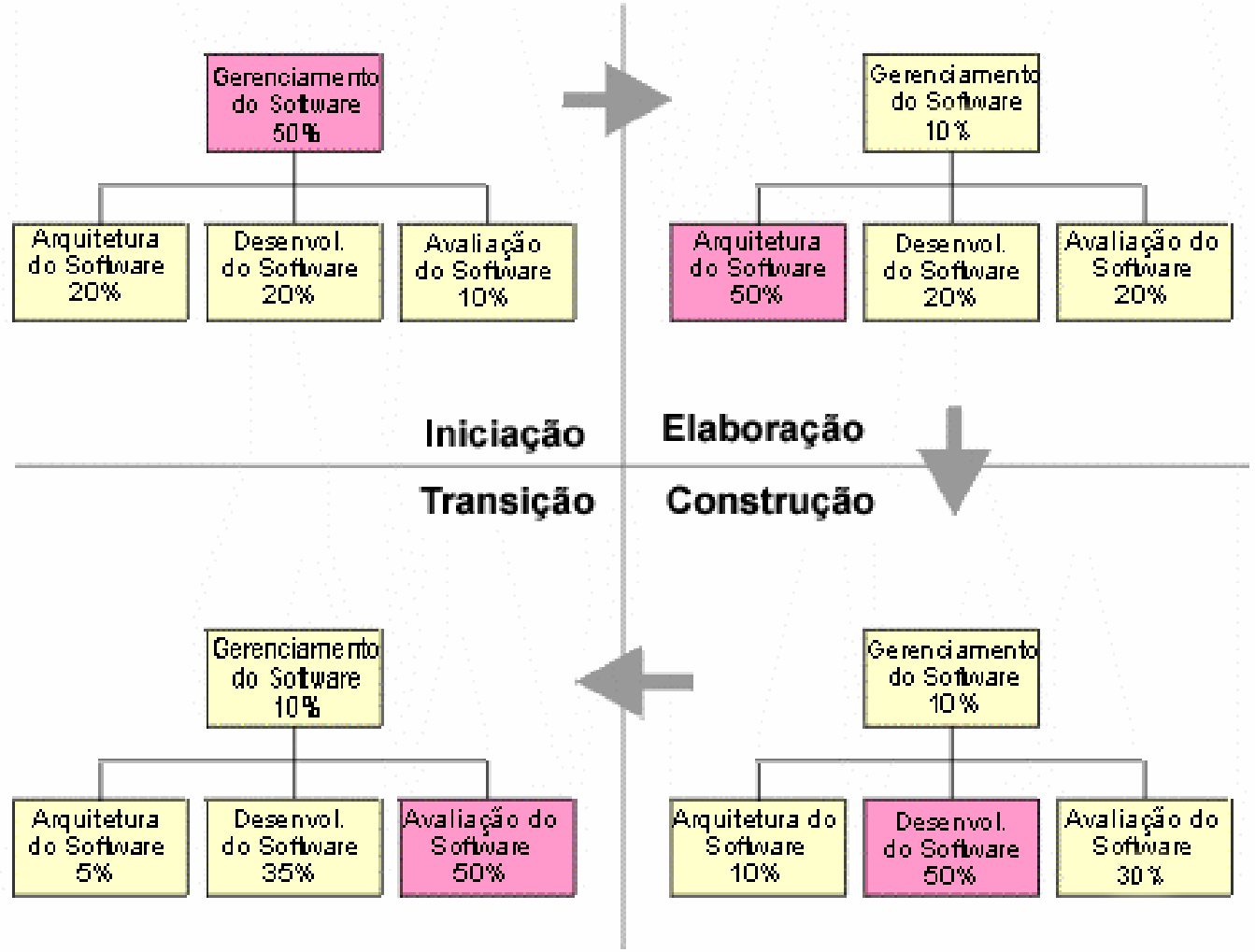
Resumindo o RUP



Resumindo o RUP



Resumindo o RUP



Evolução da equipe de projeto de software no ciclo de vida

Resumindo o RUP