

PARADIGMAS DE ENGENHARIA DE SOFTWARE

1



AGENDA

- Paradigma?
- Processo?
- Processo de Software?
- Modelos de Processo de Software
 - Sequencial
 - Incremental
 - Iterativo
 - Híbrido
- Desenvolvimento Ágil

PARADIGMA?

Exemplo típico ou modelo de algo. É a representação de um padrão a ser seguido.

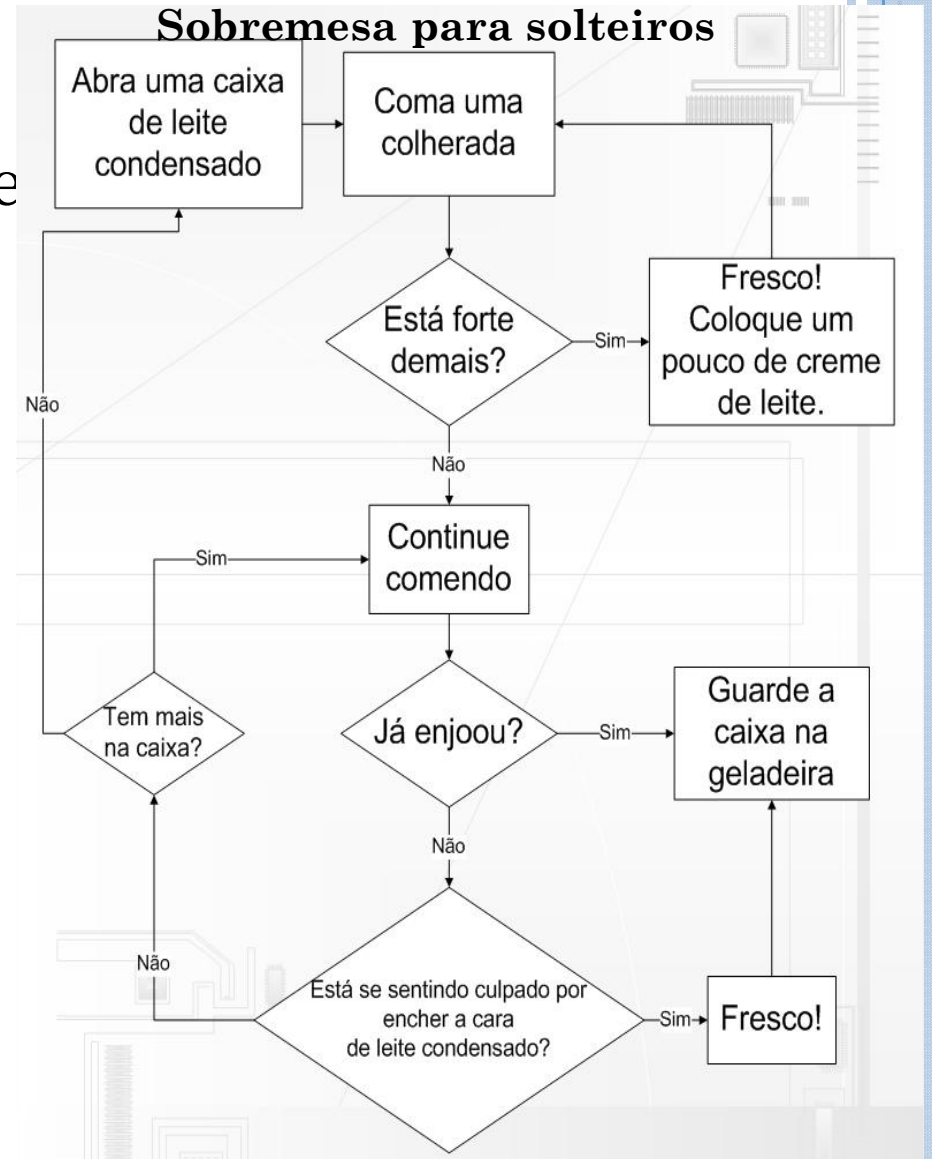
PARADIGMA DE ENGENHARIA DE SOFTWARE?

Recordando...

Engenharia de Software = é uma disciplina de engenharia relacionada com todos os aspectos da produção de software. (*Sommerville, 2008*)

PROCESSO?

- No latim *procedere* é verbo que indica a ação de avançar, ir para frente (pro+cedere)
- Conjunto de manipulações para obter um resultado.
- Modo de fazer alguma coisa.



PROCESSO?

Vamos praticar?

1 – Processo para chupar uma bala que está no bolso da calça.

2 – Processo de inicialização do seu dia.

De: Acordar

Até: Chegar ao trabalho ou faculdade.

PROCESSO DE SOFTWARE?

“Combinação de **atividades**, **ferramentas** e **procedimentos** visando o desenvolvimento ou evolução de um software”. (*Sommerville, 2011*)

“**Roteiro**, ou conjunto de **passos**, **previsível** que ajuda a criar a tempo um software de alta qualidade (*Pressman, 2011*)

PROCESSO DE SOFTWARE?

- Um processo de software
 - prescreve a **ordem** e a **frequência** de cada fase/atividade
 - especifica **critérios** para mudar de uma fase para outra
 - define o que tem que ser **entregue** ao final de cada fase
- Um processo de software **NÃO** significa
 - “sobrecarga”, “papelada desnecessária”, “perda de tempo”
- Um processo de software tem efeito **positivo**
 - para atender ao **cronograma** e obter software com mais **qualidade** e mais **fácil de manter**

PROCESSO DE SOFTWARE?

- Como escolher um processo
 - As **CARACTERÍSTICAS DA APLICAÇÃO** (domínio do problema, tamanho, complexidade etc);
 - A **TECNOLOGIA** a ser adotada na sua construção (paradigma de desenvolvimento, linguagem de programação, mecanismo de persistência etc), a organização;
 - **ONDE** o produto será desenvolvido;
 - O **PERFIL DA EQUIPE** de desenvolvimento.

Quando se escolhe um processo define-se um **MODELO DE PROCESSO (CICLO DE VIDA)**.

MODELOS DE PROCESSO DE SOFTWARE


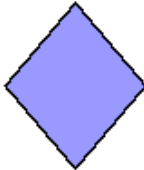
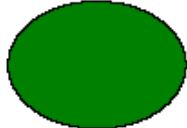
“É uma representação **simplificada** de um processo de software. Cada modelo de processo representa um processo sob determinada perspectiva e, desta forma, fornece somente informações parciais sobre esse processo”. (*Sommerville, 2008*)

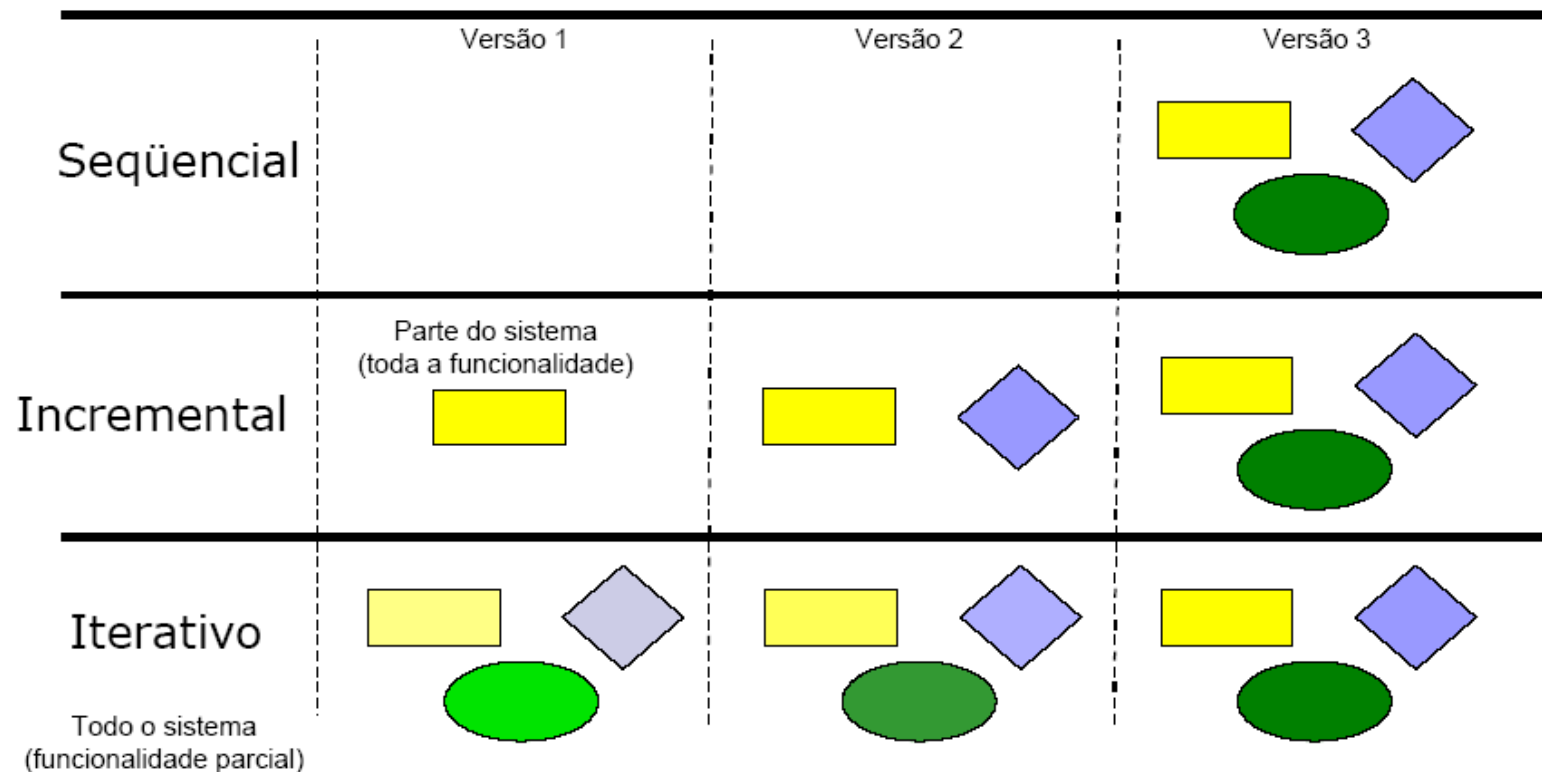
MODELOS DE PROCESSO DE SOFTWARE

- Sugerem um roteiro de atividades, ações, tarefas, marcos e produtos de trabalho necessários para desenvolver um software com qualidade.
- Engenheiros de software e gerentes adaptam um modelo prescritivo (genérico) a suas necessidades.

MODELOS DE PROCESSO DE SOFTWARE

Três abordagens principais

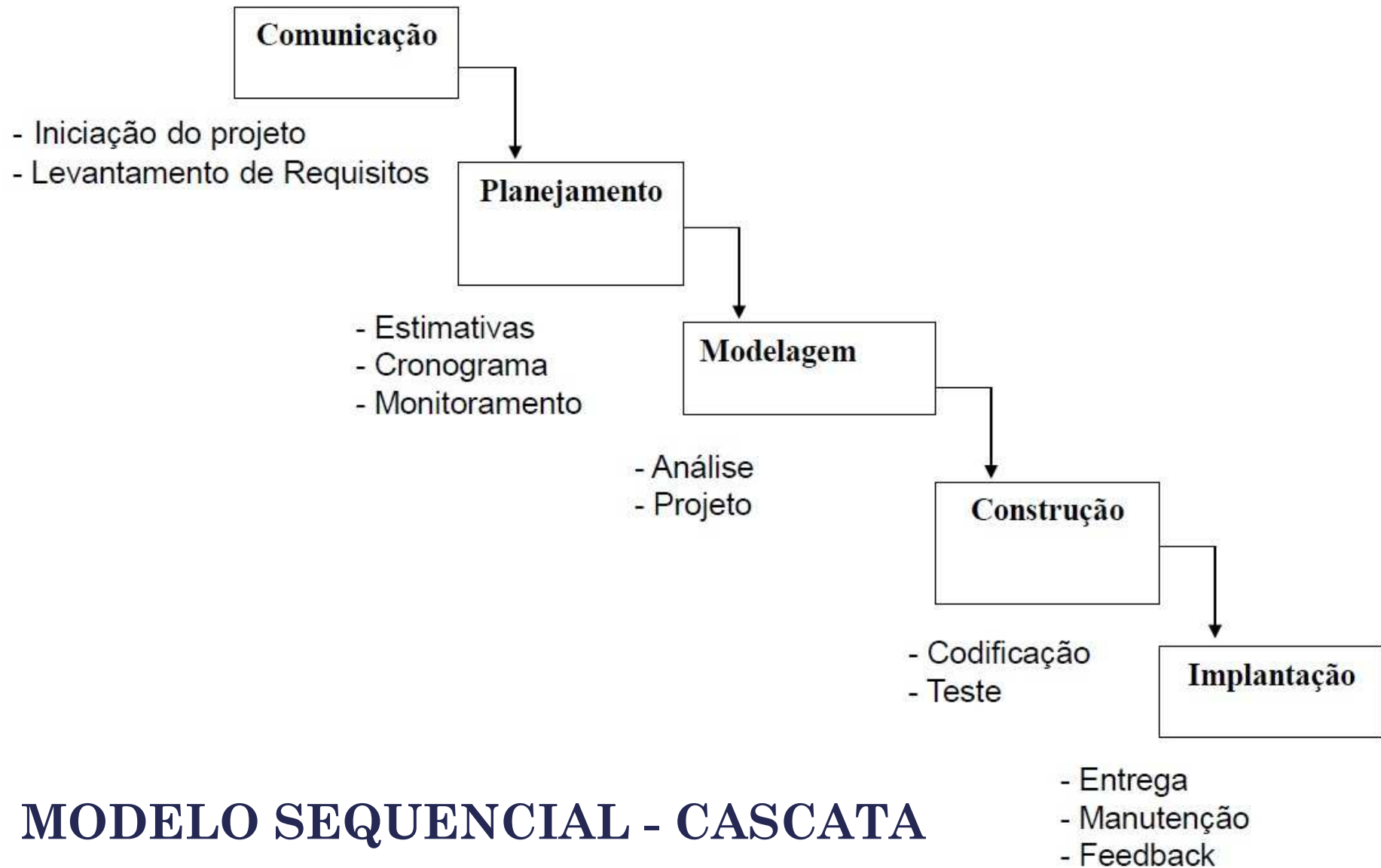
Software desejado =  +  + 



MODELO SEQUENCIAL

MODELO SEQUENCIAL - CASCATA

- Clássico, requer abordagem **sistemática** e **sequencial** ao desenvolvimento de software
- Principal característica
 - **“O resultado de uma fase é a entrada da próxima”**



MODELO SEQUENCIAL - CASCATA

○ Problemas

- Em projetos reais, é difícil estabelecer **todos os requisitos** no início de um processo (incertezas)
- Difícil acomodar **mudanças** com o processo em andamento, pois uma fase deve estar **completa** para passar para a próxima (sem paralelismo)
 - Inflexibilidade em estágios distintos dificulta resposta aos requisitos de mudança do **cliente**
- Cliente **paciente**
 - Uma versão executável só fica disponível em uma etapa **avançada** do desenvolvimento

MODELO SEQUENCIAL - CASCATA

○ Quando usá-lo?

- Apenas quando os **requisitos** são muito bem **compreendidos**, e quando as **mudanças** forem bastante **limitadas** durante o desenvolvimento
 - Poucos sistemas de negócio têm requisitos estáveis
- “... é significativamente **melhor** do que uma abordagem casual de desenvolvimento de software”
 - Modelo Cascata é simples e fácil de usar e gerenciar

MODELO SEQUENCIAL - CASCATA

○ Contribuições

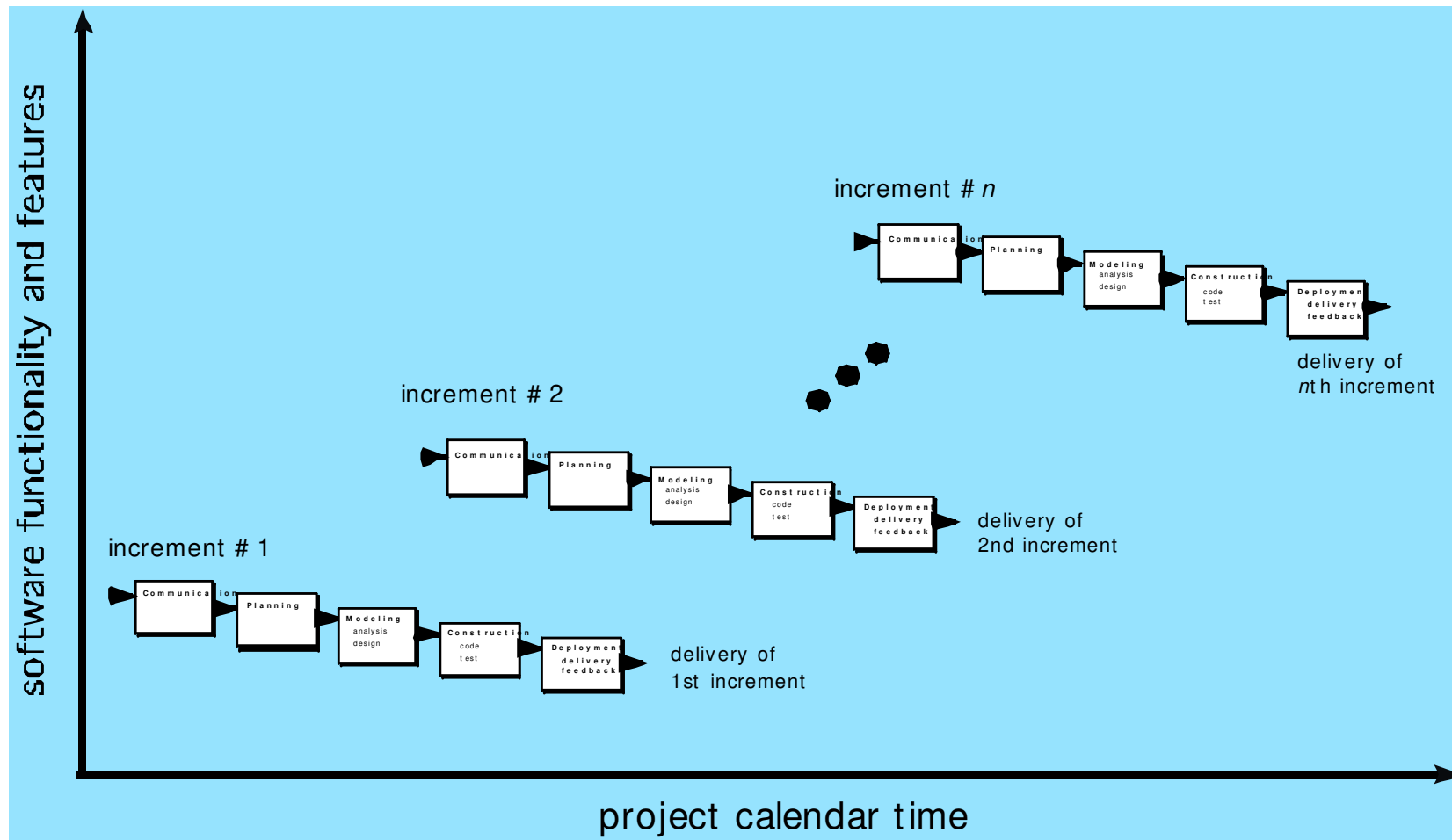
- O processo de desenvolvimento de software é submetido a **disciplina**, planejamento, gerenciamento e documentação
- A implementação do produto de software é postergada até que os requisitos tenham sido completamente entendidos
- Modelo **mais antigo** e foi amplamente usado na Engenharia de Software

MODELO INCREMENTAL

MODELO INCREMENTAL

- Ao invés de o sistema sofrer uma única entrega, o **desenvolvimento** e a **entrega** são **separados** em **incrementos**, sendo que cada incremento fornece **parte da funcionalidade** solicitada
- Requisitos de usuário são **priorizados** (primeiros são o núcleo básico) e os requisitos de prioridade mais alta são incluídos nos incrementos iniciais
 - Se o desenvolvimento de um incremento é iniciado, seus requisitos são **congelados**, embora os requisitos para incrementos posteriores possam continuar evoluindo

MODELO INCREMENTAL



MODELO INCREMENTAL

○ Quando usá-lo?

- Quando não há **mão de obra disponível** para uma **implementação completa** dentro do prazo de entrega estabelecido
- Quando o **núcleo do software** a ser desenvolvido é **bem conhecido**, mas não os requisitos em sua totalidade (mais realista)

MODELO INCREMENTAL

◦ Exemplo Prático

- Software de processamento de texto
 - Incremento inicial (núcleo)
 - gestão básica de arquivos, edição e produção de documentos (ex: novo, abrir, salvar, imprimir, formatação com tipo e tamanho de fonte, N, I, S, ...)
 - Incrementos posteriores
 - edição e produção de documentos mais sofisticados (ex: salvar como, gerar PDF,)
 - verificação ortográfica e gramatical
 - ...

MODELO ITERATIVO/EVOLUCIO NÁRIO

MODELO ITERATIVO

- Modelos de processo **iterativos** que permitem desenvolver **versões cada vez mais completas** de um software
 - Desenvolve-se uma implementação **inicial**, expondo-a aos comentários do **usuário**
 - Depois, refina-se esse resultado por meio **de várias versões** até que seja desenvolvido um sistema adequado

MODELO ITERATIVO RAD

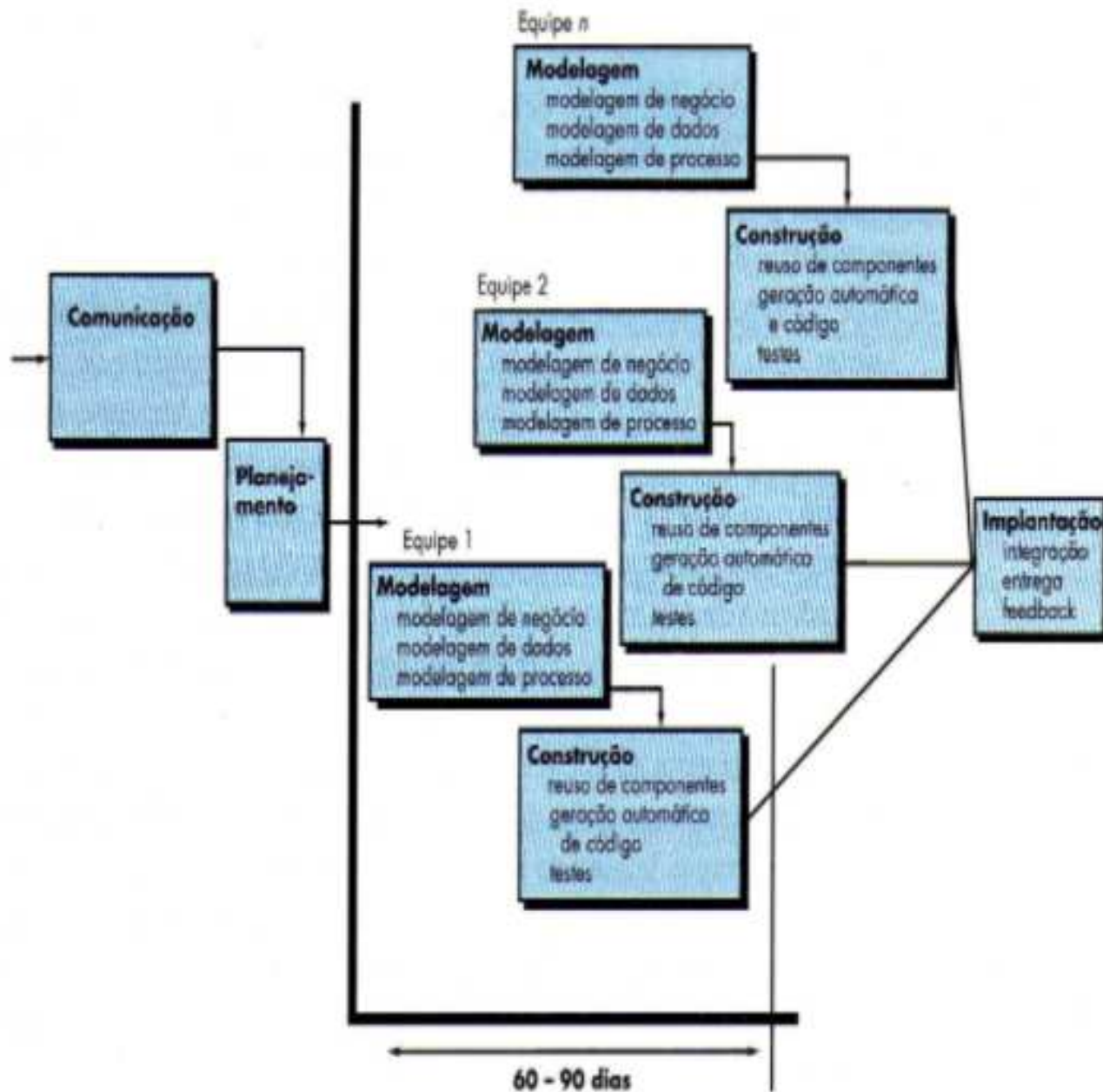
MODELO ITERATIVO - RAD

- O RAD(Rapid Application Development, Desenvolvimento Rápido de Aplicação) é um modelo de processo de software incremental que enfatiza um ciclo de desenvolvimento curto.
- O Modelo RAD é uma adaptação de “alta velocidade” do modelo em cascata, no qual o desenvolvimento rápido é conseguido com o uso de uma abordagem de construção baseada em componentes.
- No modelo RAD a comunicação trabalha para entender os problemas do negócios e as características informais que o software precisa acomodar.

MODELO ITERATIVO - RAD

- O planejamento é essencial, porque várias equipes de software trabalham em paralelo em diferentes funções do sistema.
- A construção enfatiza o uso de componentes de software preexistentes e a aplicação de geração de códigos automática.
- A implantação estabelece a base das iterações subsequentes se necessárias.
- Se uma aplicação comercial pode ser modularizada de modo a permitir que cada função principal possa ser completada em menos de três meses é uma candidata ao RAD.

MODELO ITERATIVO - RAD



MODELO ITERATIVO – RAD - DESVANTAGENS

- Para projetos grandes, mas passíveis de sofrer aumento, o RAD exige recursos humanos suficientes para criar um número adequado de equipes RAD.
- Se desenvolvedores e clientes não estiverem comprometidos com as atividades continuamente rápidas, para completar o sistema em curtíssimo espaço de tempo, o projeto RAD falhará.
- Se o sistema não puder ser adequadamente modularizado, as construções dos componentes necessários ao RAD serão problemáticas.
- O RAD pode não ser ajustado quando os riscos técnicos são altos

MODELO ITERATIVO PROTOTIPAGEM

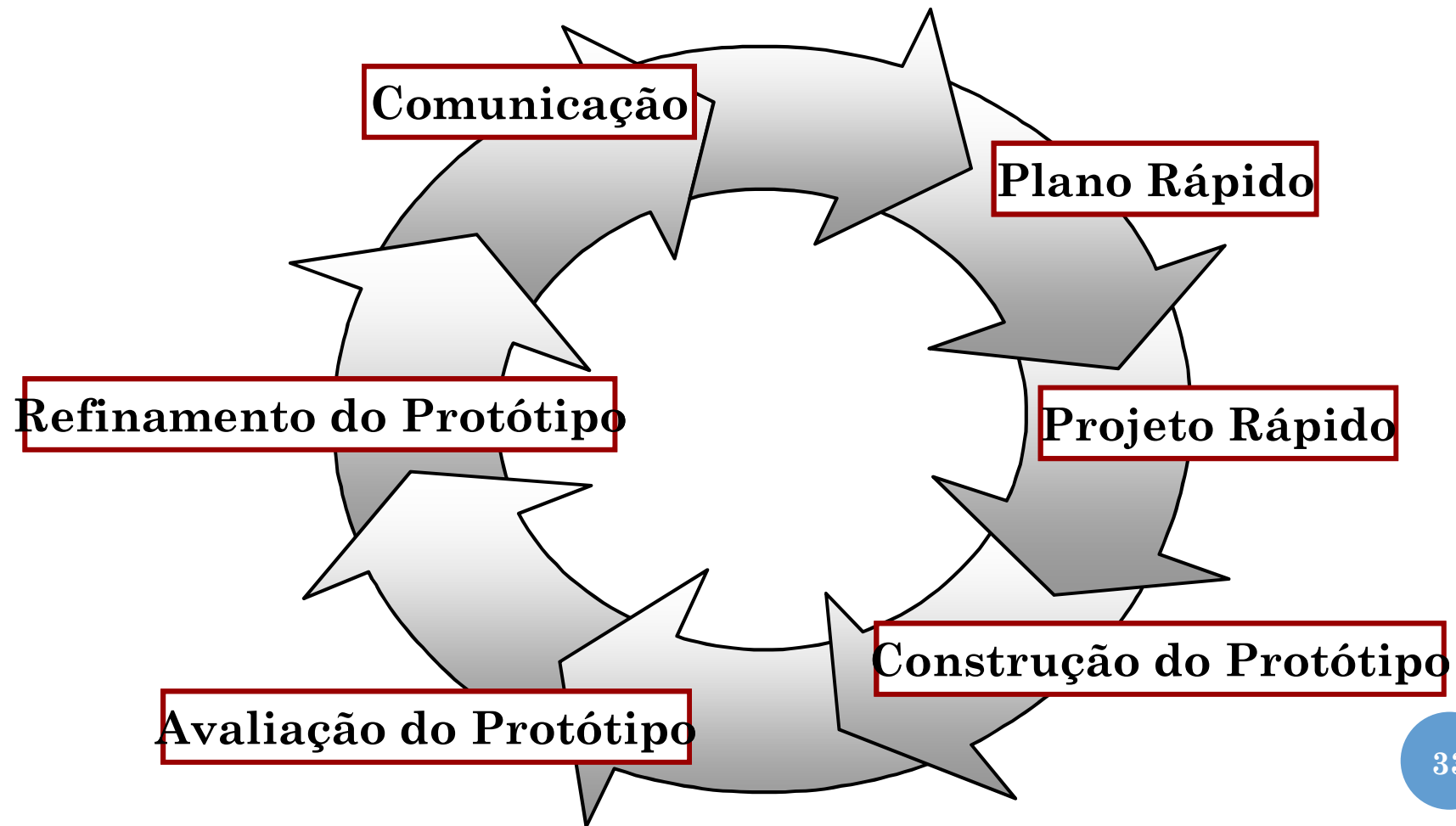
MODELO ITERATIVO - PROTOTIPAGEM

- O que é isso?
 - Produto ainda **não comercializado**, mas em fase de **testes** ou de **planejamento**. Pode ser um avião, automóvel, turbina, etc. geralmente testados antes em modelos físicos, laboratórios ...
 - É um **sistema/modelo** sem todos os elementos funcionais, mas apenas com as funcionalidades **gráficas** e algumas **básicas**. É utilizado para **aprovação** de quem solicita o software

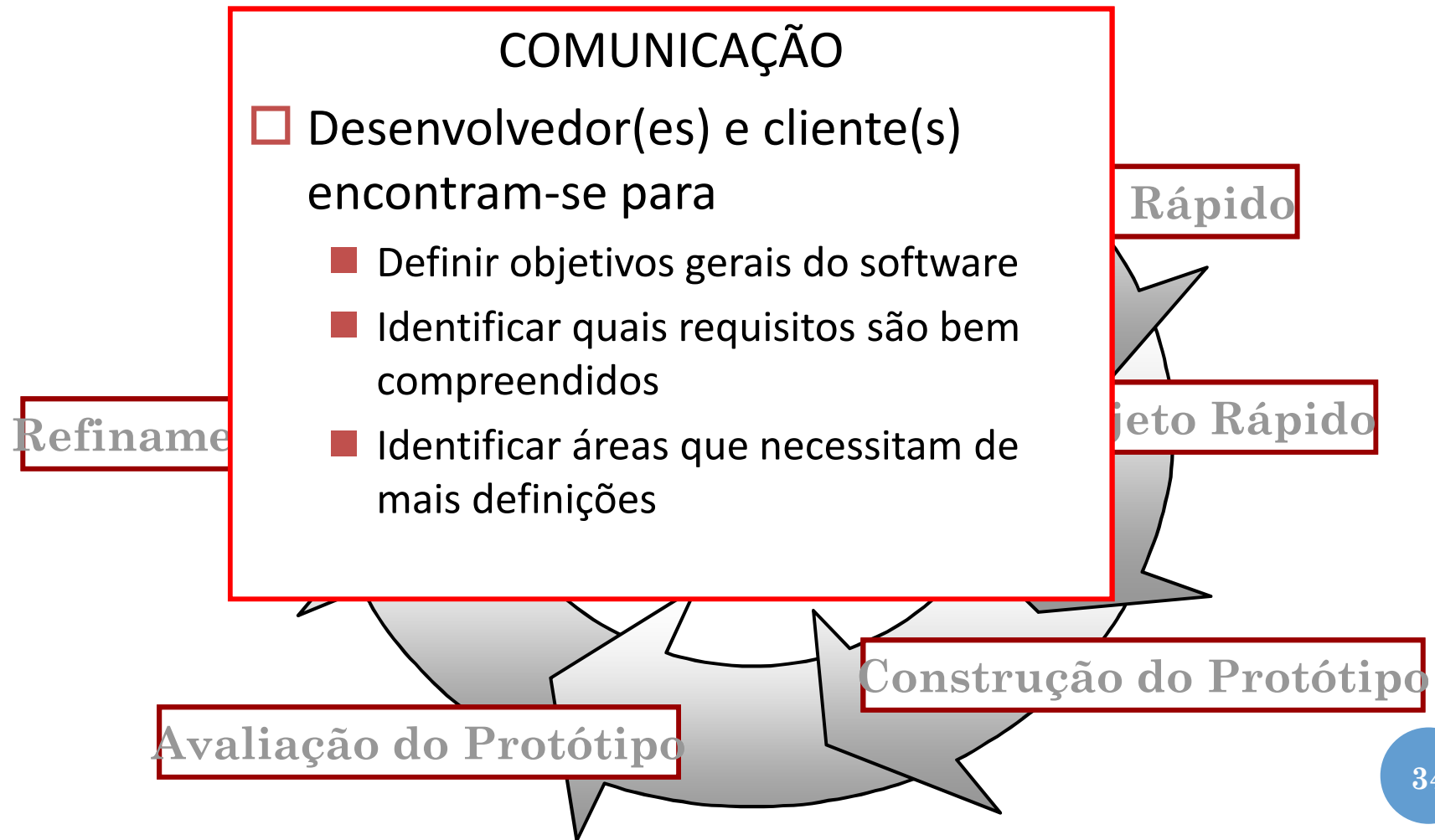
MODELO ITERATIVO - PROTOTIPAGEM

- Entendi, mas para quê?
 - Entender os **requisitos** do **usuário** e, daí obter uma melhor definição dos **requisitos** do **sistema**
 - Quando o cliente definiu **objetivos gerais** para o software, mas ainda não os **detalhou**, ou quando os requisitos estão **confusos**
 - Quando o engenheiro de software pode se sentir inseguro quanto à **eficiência** dos algoritmos, da adaptação a **mudanças**, da interação com **hardware específico** e outros **softwares**, etc.

MODELO ITERATIVO - PROTOTIPAGEM



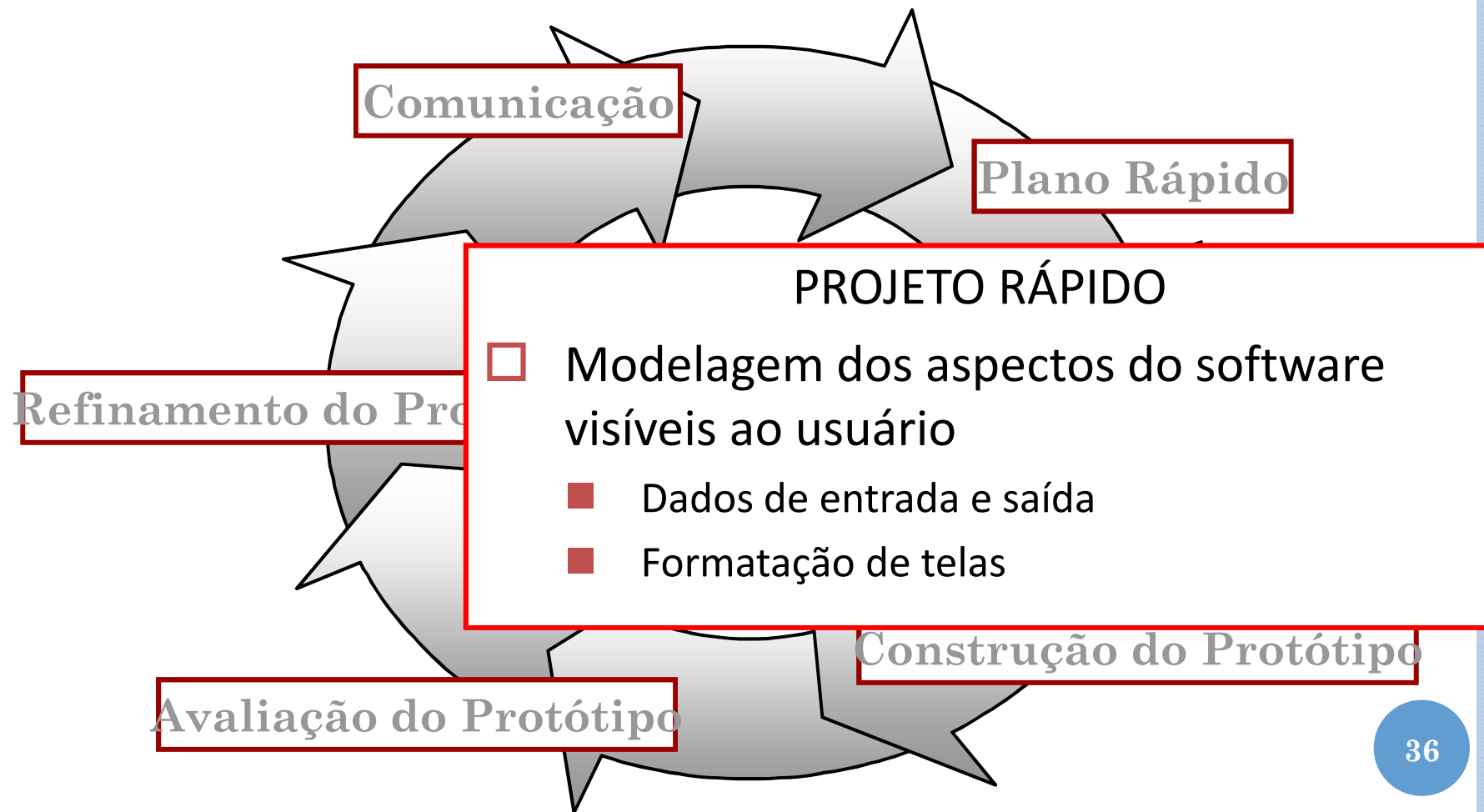
MODELO ITERATIVO - PROTOTIPAGEM



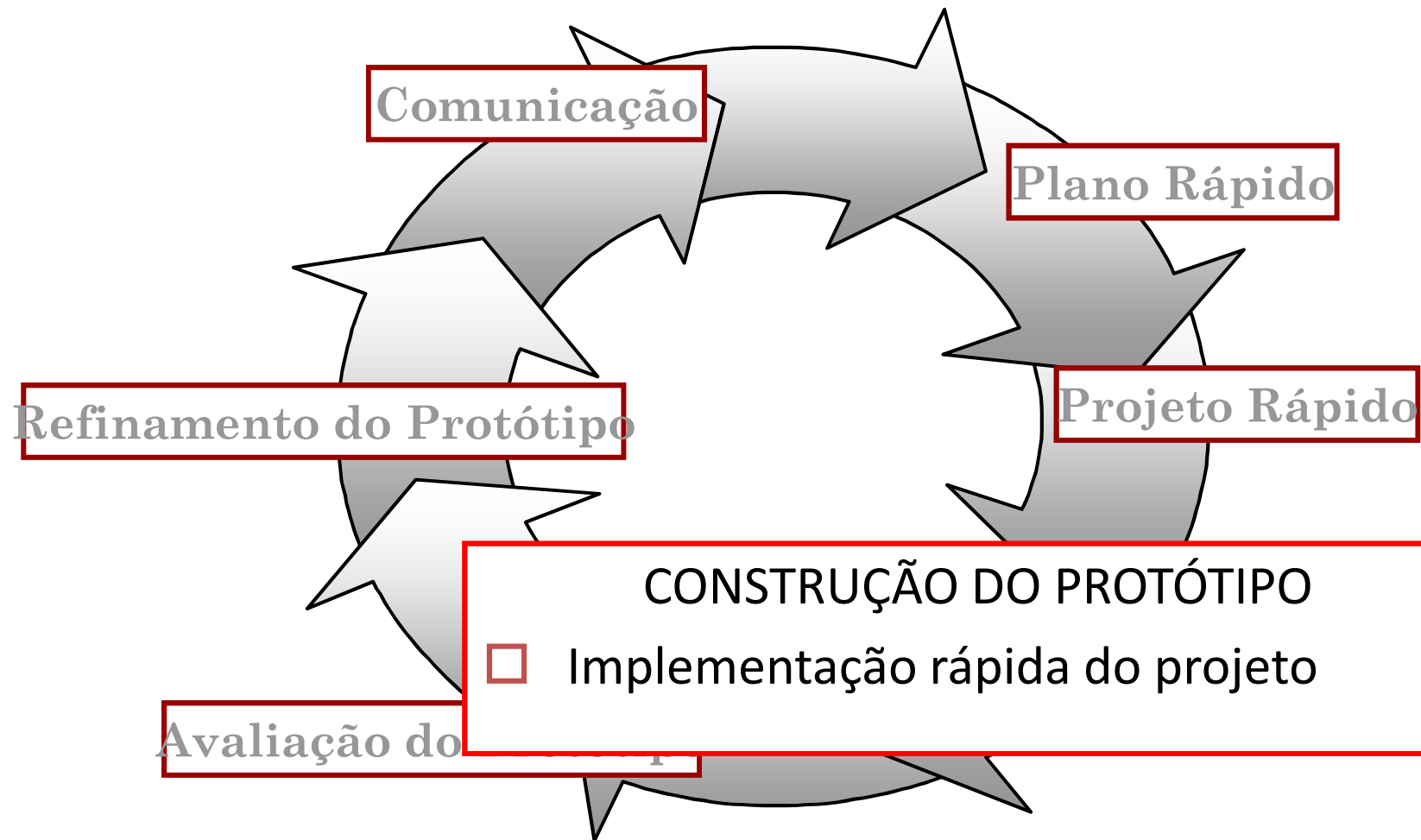
MODELO ITERATIVO - PROTOTIPAGEM



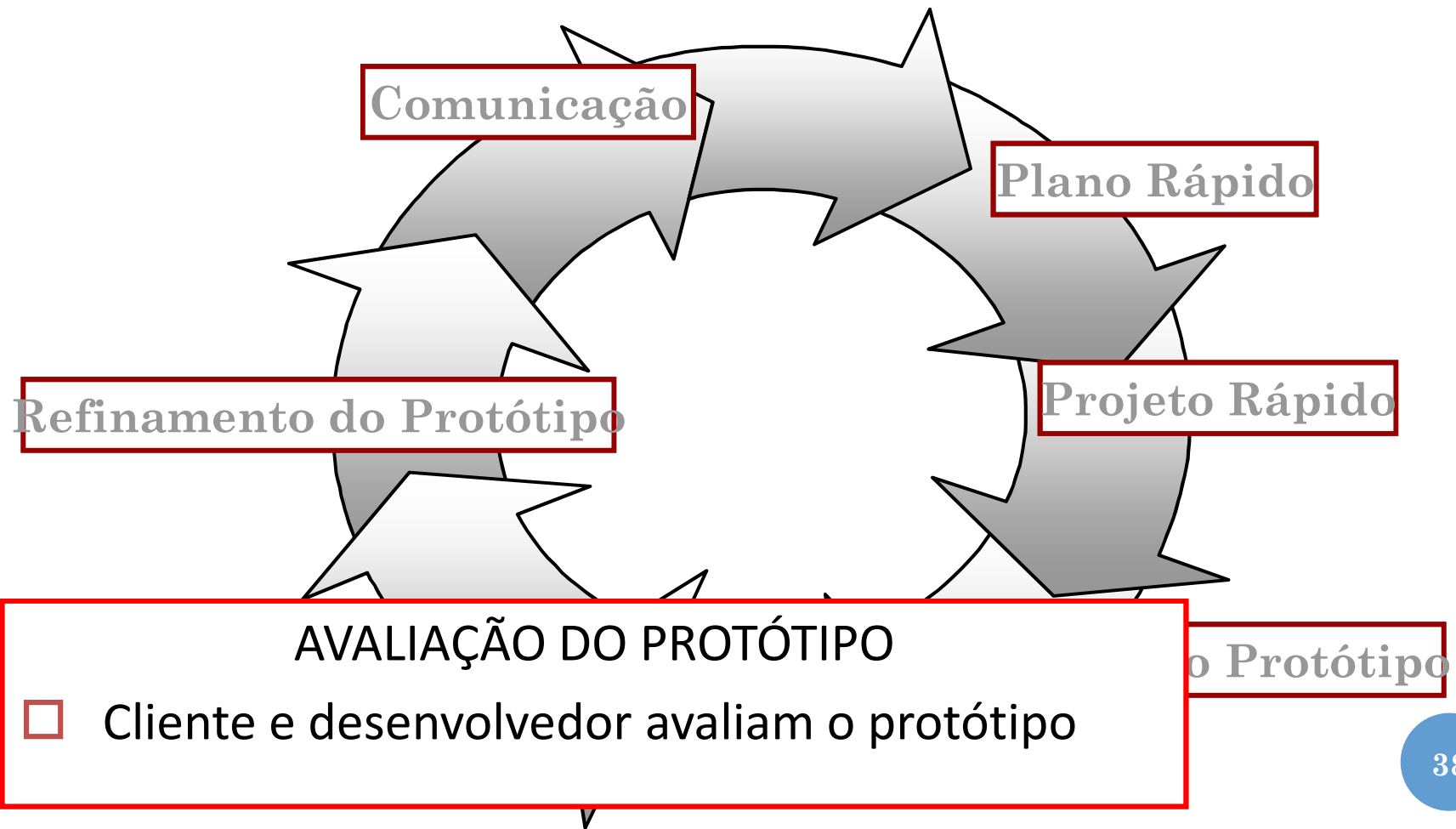
MODELO ITERATIVO - PROTOTIPAGEM



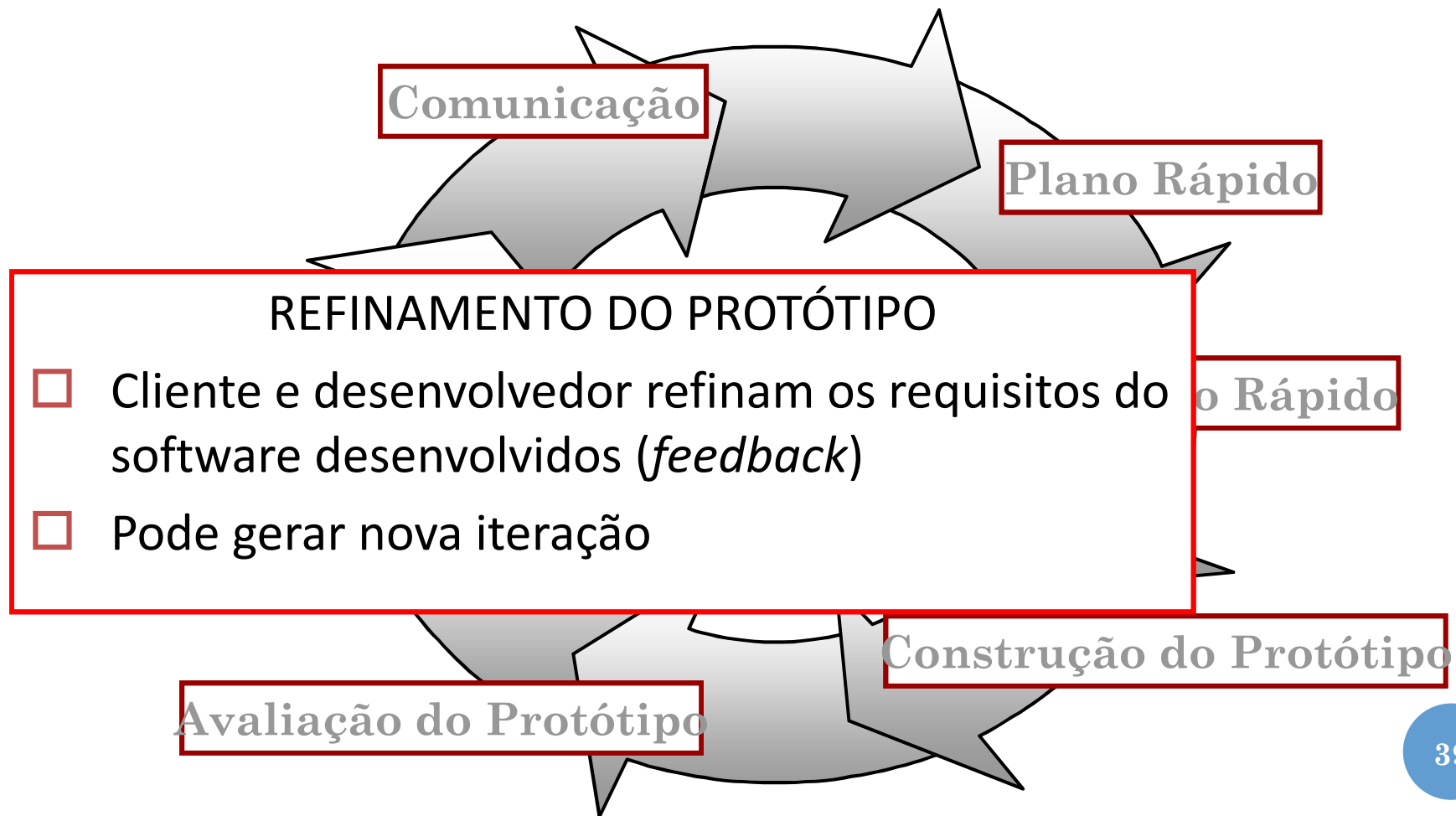
MODELO ITERATIVO - PROTOTIPAGEM



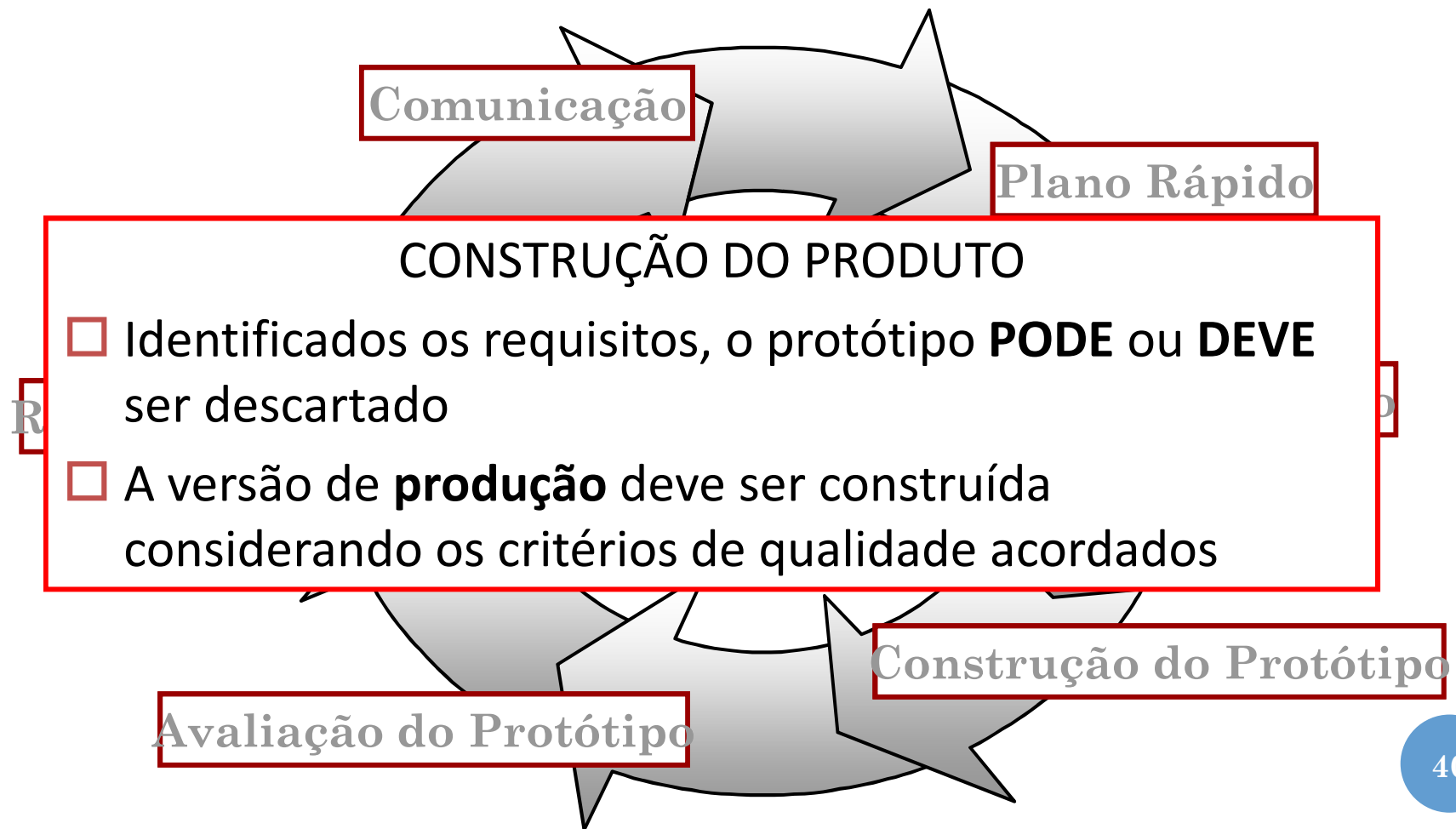
MODELO ITERATIVO - PROTOTIPAGEM



MODELO ITERATIVO - PROTOTIPAGEM



MODELO ITERATIVO - PROTOTIPAGEM



MODELO ITERATIVO - PROTOTIPAGEM

○ Vantagens

- Usuários têm o “**sabor**” de um sistema real precocemente
- Desenvolver “**algo**” em prazo curto os desenvolvedores conseguem “**entender**” o sistema e construir

○ Desvantagens

- Cliente "pensa" estar usando uma versão **operacional**
- Concessões **equivocadas** do desenvolvedor para entregar logo o protótipo (Algoritmo ineficiente, SO ou linguagem inapropriados)
- O descarte do protótipo pode ser visto com perda de tempo para o cliente

MODELO ITERATIVO - PROTOTIPAGEM

- Definir as “regras do jogo” no início
 - Cliente e desenvolvedor devem estar de acordo
 - que o protótipo deve servir como mecanismo de **definição de requisitos**
 - que o protótipo **pode ser descartado** (*prototipação throwaway*), ou apenas **parte** dele
 - e que o **software real** será desenvolvido visando qualidade

MODELO ITERATIVO - PROTOTIPAGEM

○ Problemas?

- Falta de **visibilidade** de processo
 - Inviabilidade econômica de se produzir **documentação** para cada versão desenvolvida rapidamente
 - Falta de **produtos regulares** inviabilizam a medição do progresso do projeto
- Software freqüentemente **mal estruturado**
 - Mudanças contínuas tendem a corromper a **estrutura** do software, tornando-a cada vez mais onerosa e difícil
- Habilidades **especiais** podem ser solicitadas
 - Exemplos, linguagens para **prototipação rápida**

MODELO ITERATIVO - PROTOTIPAGEM

○ Quando Usá-la?

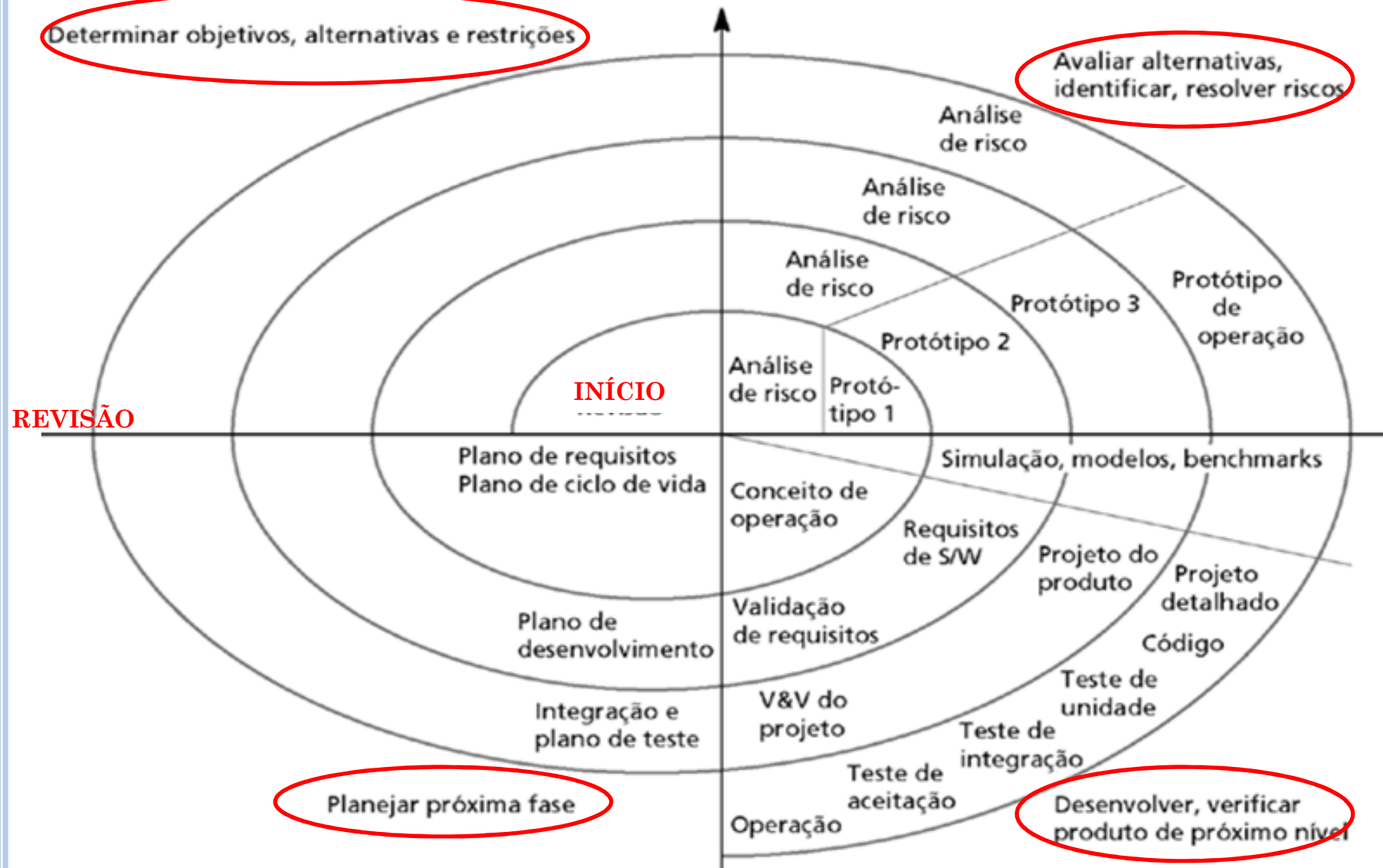
- Sistemas interativos de **pequeno/médio** porte
 - Mais fácil e rápido para estabelecer uma **arquitetura estável** do sistema
 - Sistemas de grande porte e longo ciclo de vida costumam a estabelecer uma arquitetura estável
- Em partes de um sistema de **grande** porte
 - Por exemplo, a interface com o usuário
- Normalmente, é utilizada como **técnica auxiliar** em vez de modelo de processo independente

MODELO ITERATIVO ESPIRAL

MODELO ITERATIVO - ESPIRAL

- Processo representado como uma espiral e cada volta na espiral representa uma iteração
 - Sem fases definidas (ex: comunicação, projeto) as voltas na espiral são escolhidas com base no que é requisitado
- Processo direcionado a riscos e planejamento
 - Riscos são explicitamente avaliados e resolvidos ao longo do processo

MODELO ITERATIVO - ESPIRAL



MODELO ITERATIVO - ESPIRAL

1. Identificar objetivos críticos e restrições
2. Avaliar alternativas de projeto e de processo
3. Identificar riscos
4. Resolver um subconjunto de riscos (pensando nos custos) usando análise, modelos, protótipos, simulações
5. Desenvolver “entregáveis” do projeto: especificação de requisitos, de projeto, implementação e testes
6. Planejar os próximos ciclos – atualizar o plano de projeto incluindo cronograma, custo e o número de iterações subsequentes
7. Revisar junto com os interessados no projeto se o trabalho está sendo seguido como esperado

MODELO ITERATIVO - 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
 - Escolha de linguagem inadequada, que pode gerar atraso na entrega e aumento de custo
- Desenvolvimento e validação
 - Um modelo de desenvolvimento para o sistema é escolhido (qualquer um dos modelos genéricos apresentados)
- Planejamento
 - O custo do projeto é revisitado e revisado e a próxima fase da espiral é planejada

MODELO ITERATIVO - ESPIRAL

○ Vantagens?

- Riscos são gerenciados cedo e ao longo do processo – reatividade a riscos, que são reduzidos antes de se tornarem problemáticos
- Usa prototipação para reduzir riscos
- Software evolui enquanto o projeto prossegue – erros/alternativas não atrativas são eliminadas cedo
- Planejamento é construído sobre o processo – cada ciclo inclui um passo de planejamento para auxiliar o monitoramento do projeto

MODELO ITERATIVO - ESPIRAL

○ Desvantagens?

- Complicado de usar e controlar – análise de riscos requer experiência e, conseqüentemente, \$\$\$
 - Se um risco importante não for detectado ou bem gerenciado, ocorrerão problemas
- Pode ser inadequado para pequenos projetos – não faz sentido se o custo da análise de riscos é grande parte do custo do projeto como um todo

MODELO ITERATIVO - ESPIRAL

○ Quando usá-lo?

- Abordagem mais realista para sistemas de software de **grande porte**
- **Equipe de projeto experiente** em relação a planejamento e, principalmente, análise de **riscos**

MODELOS ESPECIALIZADOS DE PROCESSOS

- Os modelos especializados tendem a ser aplicados quando uma abordagem de engenharia de software estreitamente definida é escolhida.

DESENVOLVIMENTO BASEADO EM COMPONENTES

- Os componentes de software comercial de prateleira, desenvolvidos por vendedores que os oferecem como produtos, podem ser usados quando o software precisa ser construído. Esses componentes fornecem funcionalidades-alvo com interfaces bem definidas que permitem ao componente ser integrado no software.
- O modelo compõe aplicações a partir de componentes de software previamente preparados. As atividades de modelagem e construção começam com a identificação dos componentes candidatos.
- O modelo de desenvolvimento baseado em componentes incorpora muitas das características do modelo espiral, demanda uma abordagem iterativa para criação de software.

DESENVOLVIMENTO BASEADO EM COMPONENTES

- Independente da tecnologia usada para criar os componentes, o modelo de desenvolvimento baseado em componentes incorpora os seguintes passos:
 1. Produtos baseados em componente disponíveis são pesquisados e avaliados para o domínio da aplicação em questão.
 2. Tópicos de integração de componentes são considerados.
 3. Uma arquitetura de software é projetada para acomodar os componentes.
 4. Componentes são integrados a arquitetura.
 5. Testes abrangentes são realizados para garantir a funcionalidade adequada.

DESENVOLVIMENTO BASEADO EM COMPONENTES

- O modelo de desenvolvimento baseada em componentes leva ao reuso do software, e a reusabilidade fornece aos engenheiros vários benefícios mensuráveis.
- ✓ Redução de 70% do prazo do ciclo de desenvolvimento.
- ✓ Redução de 84% do custo do projeto
- ✓ Um índice de produtividade de 26,2.

MODELO DE MÉTODOS FORMAIS

- O modelo de métodos formais abrange um conjunto de atividades que levam à especificação matemática formal do software de computador. Os métodos formais permitem ao engenheiro de software especificar, desenvolver e verificar um sistema baseado em computador pela aplicação rigorosa de uma notação matemática.
- Quando métodos formais são usados durante o desenvolvimento, eles fornecem um mecanismo para eliminação de muitos problemas que são difíceis de resolver usando outros paradigmas de engenharia de software.
- Quando usados durante o projeto, métodos formais servem de base para a verificação do programa e, assim, permite que o engenheiro descubra e corrija erros que poderiam passar despercebidos.

MODELO DE MÉTODOS FORMAIS

- As desvantagens desse método são:
 - O desenvolvimento de modelos formais é atualmente muito lento e dispendioso.
 - Como poucos desenvolvedores de software têm o preparo necessário para aplicar métodos formais, torna-se necessário um treinamento extensivo.
 - É difícil usar os modelos como um mecanismo de comunicação, com clientes despreparados tecnicamente.

MODELO HÍBRIDO

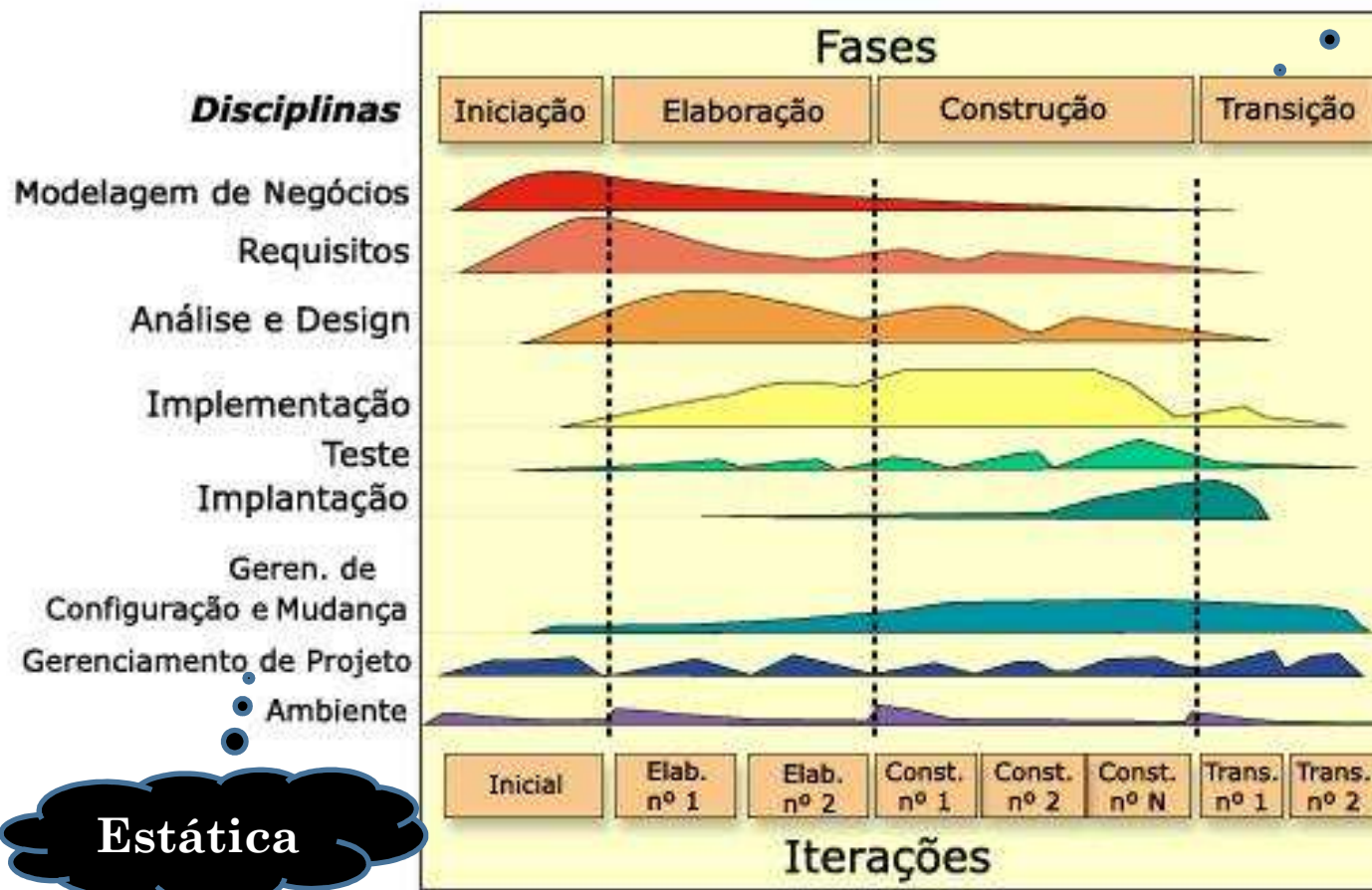
RATIONAL UNIFIED PROCESS (RUP)

- A metodologia RUP utiliza uma abordagem de orientação a objetos em sua concepção e é projetado e documentado utilizando o UML para ilustrar os processos. Tem como principais características ser incremental e iterativo. Incremental significa que aquele software é construído e entregue em pedaços, constituindo um conjunto de funcionalidades completas.
- Através de pequenos ciclos de projetos – que correspondem a uma iteração – o software é melhorado através da adição de mais detalhes, o que resulta em um incremento no software. Iterações referem-se a passos e incrementos a evolução do produto.

MODELO HÍBRIDO - RUP

○ Rational Unified Process (RUP)

Dinâmica



Prática



Principal inovação: separação das fases e disciplinas

RATIONAL UNIFIED PROCESS (RUP)

- **Concepção:** define o escopo do software. É uma fase preliminar, é nessa etapa que se concentra o levantamento de requisitos, define preços e prazos da entrega do sistema e onde se avalia os possíveis riscos.
- **Elaboração:** plano do projeto, especificação de características e arquitetura. Aqui todas as análises de riscos são aprofundadas, como também os custos.
- **Construção:** ocorre a codificação do software.
- **Transição:** implantação do software, assegurando que ele esteja disponível aos usuários finais. Nesta fase está incluída os testes e o treinamento dos usuários.

DESENVOLVIMENTO ÁGIL

DESENVOLVIMENTO ÁGIL

- Manifesto para o Desenvolvimento Ágil
 - Assinado em **2001** por Kent Beck e outros 16 desenvolvedores, autores e consultores de software

“Desenvolvendo e ajudando outros a desenvolver software, estamos desvendando formas melhores de desenvolvimento. Por meio deste trabalho passamos a valorizar:

- *Indivíduos e interações acima de processos e ferramentas*
- *Software operacional acima de documentação completa*
- *Colaboração dos clientes acima de negociação contratual*
- *Respostas a mudanças acima de “seguir um plano”*

DESENVOLVIMENTO ÁGIL

- Surgiram de um esforço para **sanar fraquezas** reais e perceptíveis da engenharia de software convencional
- Oferece **benefícios** importantes

MAS, não é indicado para todos os tipos de projetos, produtos, pessoas e situações

DESENVOLVIMENTO ÁGIL



DESENVOLVIMENTO ÁGIL

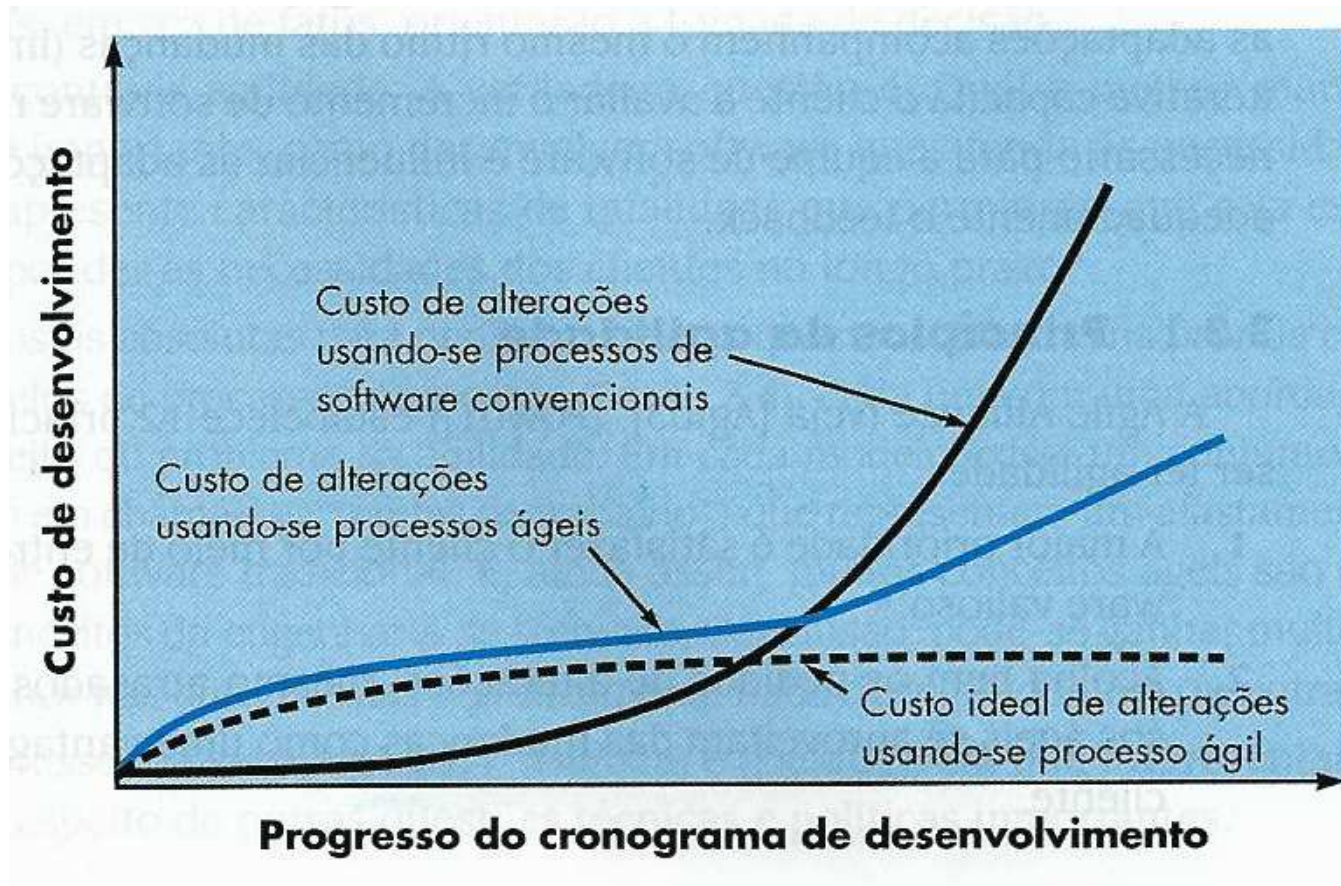
- **AGILIDADE**

- Entregar versões funcionais em **prazos curtos**
- Estar preparado para **requisitos mutantes**
- Pessoal de negócios e desenvolvedores **juntos**
- Cliente é considerado parte da equipe (visão nós e eles)
- Troca de informações através de **conversas diretas**
- Plano de projeto deve ser **flexível**

DESENVOLVIMENTO ÁGIL

- **Agilidade e Custo das Mudanças**
 - Desenvolvimento de software **convencional** afirma que os **custos com mudanças aumentam** de forma não linear conforme o projeto avança
 - Defensores da **agilidade** argumentam que o processo ágil bem elaborado **“achata” o custo da curva de mudança**
 - Processo ágil envolve entregas incrementais
 - Custo das mudanças é atenuado com entrega incremental associada a outras práticas ágeis: testes contínuos de unidade e programação por pares

DESENVOLVIMENTO ÁGIL



DESENVOLVIMENTO ÁGIL

Processo Ágil?

- Processo capaz de administrar a *imprevisibilidade*
 - Processo facilmente **adaptável**
 - Adaptar **incrementalmente**
 - Equipe precisa de **feedback do cliente** para adaptações incrementais
 - Catalisador para feedback do cliente é um **protótipo operacional** ou parte de um sistema operacional entregues em curtos períodos de tempo

DESENVOLVIMENTO ÁGIL

Abordagens?

- **Extreme Programming – XP – Programação Extrema**
- Industrial XP – IXP – Programação extrema industrial
- **Scrum**
- Adaptive Software Development – ASD – Desenvolvimento de software adaptativo
- Dynamic Systems Development Method – DSDM – Método de Desenvolvimento de Sistemas Dinâmicos
- Crystal

DESENVOLVIMENTO ÁGIL

XP – Extreme Programming

- Amplamente divulgado em 2004 por Kent Beck
- Abordagem mais amplamente usada para desenvolvimento de software ágil (dados de 2011)
- A que se destina
 - Grupos de 2 a 10 programadores
 - Projetos de 1 a 36 meses

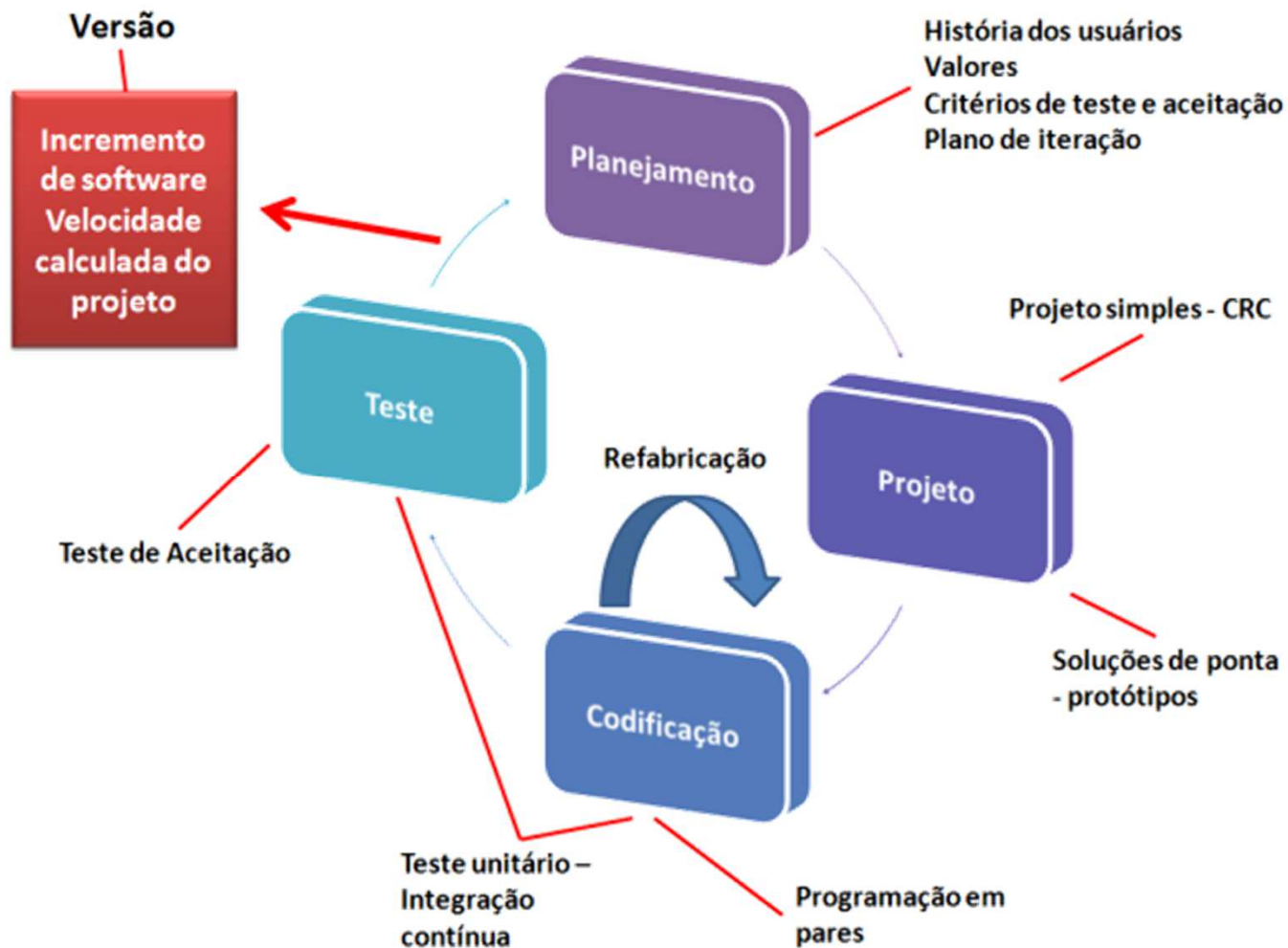
DESENVOLVIMENTO ÁGIL

XP – Extreme Programming

- Valores para trabalhos realizados com a XP
 - **Comunicação** – contínua entre cliente e desenvolvedores
 - **Simplicidade** – projetar apenas para as necessidades imediatas
 - **Feedback** – com base no próprio software implementado
 - **Coragem** – coragem e disciplina para projetar para hoje
 - **Respeito** – Membros da equipe, Clientes e o próprio software desenvolvido (faz bem feito o que deve fazer HOJE)

DESENVOLVIMENTO ÁGIL

XP – Extreme Programming



XP – EXTREME PROGRAMMING

- **Na atividade de Planejamento** temos inicialmente uma atividade de levantamento de requisitos que tem como objetivo capacitar os membros técnicos da equipe XP a entender o ambiente de negócios do software. Nesta atividade teremos um conjunto de histórias (chamadas de histórias de usuário) que tem como objetivo descrever o resultado, as características e a funcionalidade requisitada para o software a ser construído. Cada uma das histórias é descrita pelo cliente e atribuída uma prioridade à história, baseando-se no quanto essa história acrescenta de valor ao negócio do cliente. Por outro lado, os membros da equipe de desenvolvimento avaliam essa história e atribuem um custo a essa história, que é medido em semanas ou horas de desenvolvimento. Se a história durar mais do que três semanas é solicitado ao cliente dividir mais essa tarefa para que ela ocorra no máximo no prazo de três semanas. Novas histórias podem ser escritas a qualquer momento.

XP – EXTREME PROGRAMMING

- **Na atividade de projeto** temos o princípio KIS (Keep It Single) que deve ser seguido rigorosamente afim de sempre mantermos o projeto simples. Nunca devemos acrescentar funcionalidades extras. A XP também encoraja todos da equipe a usarem os cartões CRC (classe-responsabilidade-desenvolvedor) que serve como um mecanismo bastante eficaz para pensar sobre o software em um contexto OO (orientado a objetos). Esses cartões permitem identificarmos e organizarmos as classes OO para o incremento que está sendo desenvolvido. Os cartões CRC são o único artefato de projeto produzido como parte do projeto XP.

XP – EXTREME PROGRAMMING

- **Na atividade de Codificação** temos a implementação do software. Inicialmente a equipe não vai diretamente para a codificação do software propriamente dito. A equipe realizará inicialmente uma série de teste de unidades que exercitarão cada uma das histórias que serão incluídas na versão corrente. Após os testes de unidade terem sido criados, o desenvolvedor poderá focar-se melhor no que deve ser codificado e o que deve ser implementado para passar nos testes. Essa implementação dos testes antes de fazer a implementação do código também é chamada de TDD (Test Driven Development) ou Desenvolvimento guiado por testes. Outro conceito chave na atividade de codificação é o Peer Programming ou programação em pares. A XP recomenda que duas pessoas trabalhem em duplas num mesmo computador para criar código para uma história.

XP – EXTREME PROGRAMMING

- **Na atividade de teste** temos a criação dos testes de unidade que é iniciada antes da codificação. Os testes de unidade devem ser automatizados permitindo que possamos testar novamente a qualquer momento de forma fácil e repetida. Com isso também temos o encorajamento dos testes de regressão toda vez que o código for modificado. Os testes também permitem que possamos encontrar problemas logo no início. Os testes de unidade, integração e validação do sistema podem ocorrer diariamente. Outro teste que temos é o de aceitação que é realizado pelo cliente. Os testes de aceitação são obtidos de histórias de usuários.

DESENVOLVIMENTO ÁGIL

Scrum

- Definição Informal
 - Estratégia em um jogo de rugby onde jogadores colocam uma bola quase perdida novamente em jogo através de trabalho em equipe
- Método de desenvolvimento **ágil** de software
- Concebido por Jeff Sutherland no início de 1990
- Princípios são consistentes com o **manifesto ágil** e usados para **orientar** as atividades de desenvolvimento dentro de um processo que incorpora as atividades de requisitos, análise, projeto, evolução e entrega

DESENVOLVIMENTO ÁGIL

Scrum

- Enfatiza o uso de padrões de processos de software
 - **Backlog**: lista com requisitos pendentes com prioridades
 - **Sprints**: unidades de trabalho com requisitos estabelecidos e prazo de entrega (geralmente 2 à 4 semanas)
 - **Alterações**: não são introduzidas em Sprints já iniciados – membros trabalham em ambiente de curto prazo, mas estável
 - **Reuniões Scrum**: reuniões curtas (15 minutos) realizadas diariamente
 - O que realizou desde a última reunião?
 - Quais obstáculos está encontrando?

DESENVOLVIMENTO ÁGIL

