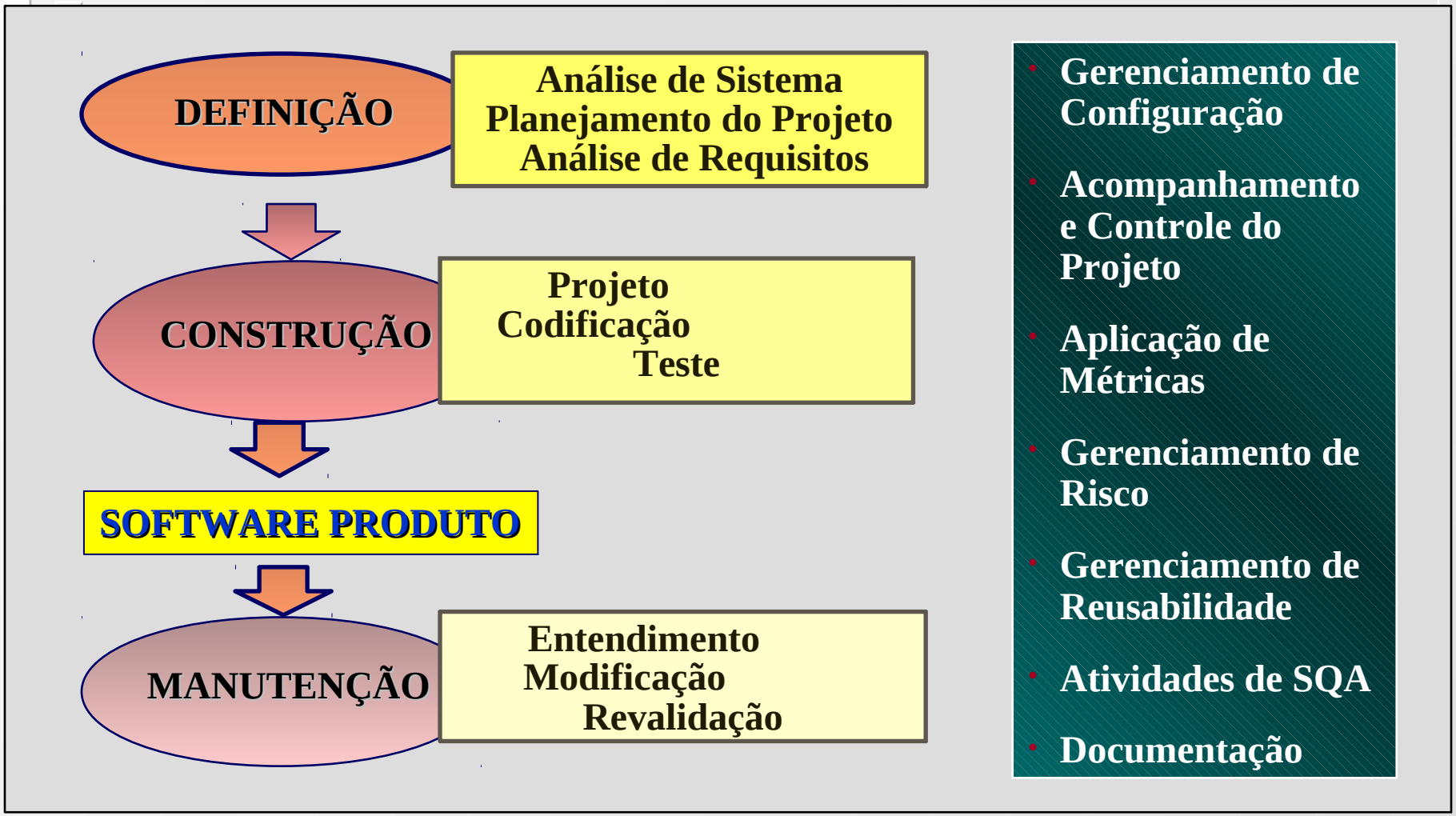
Modelos de Processo de Software

Profa. Ellen Francine

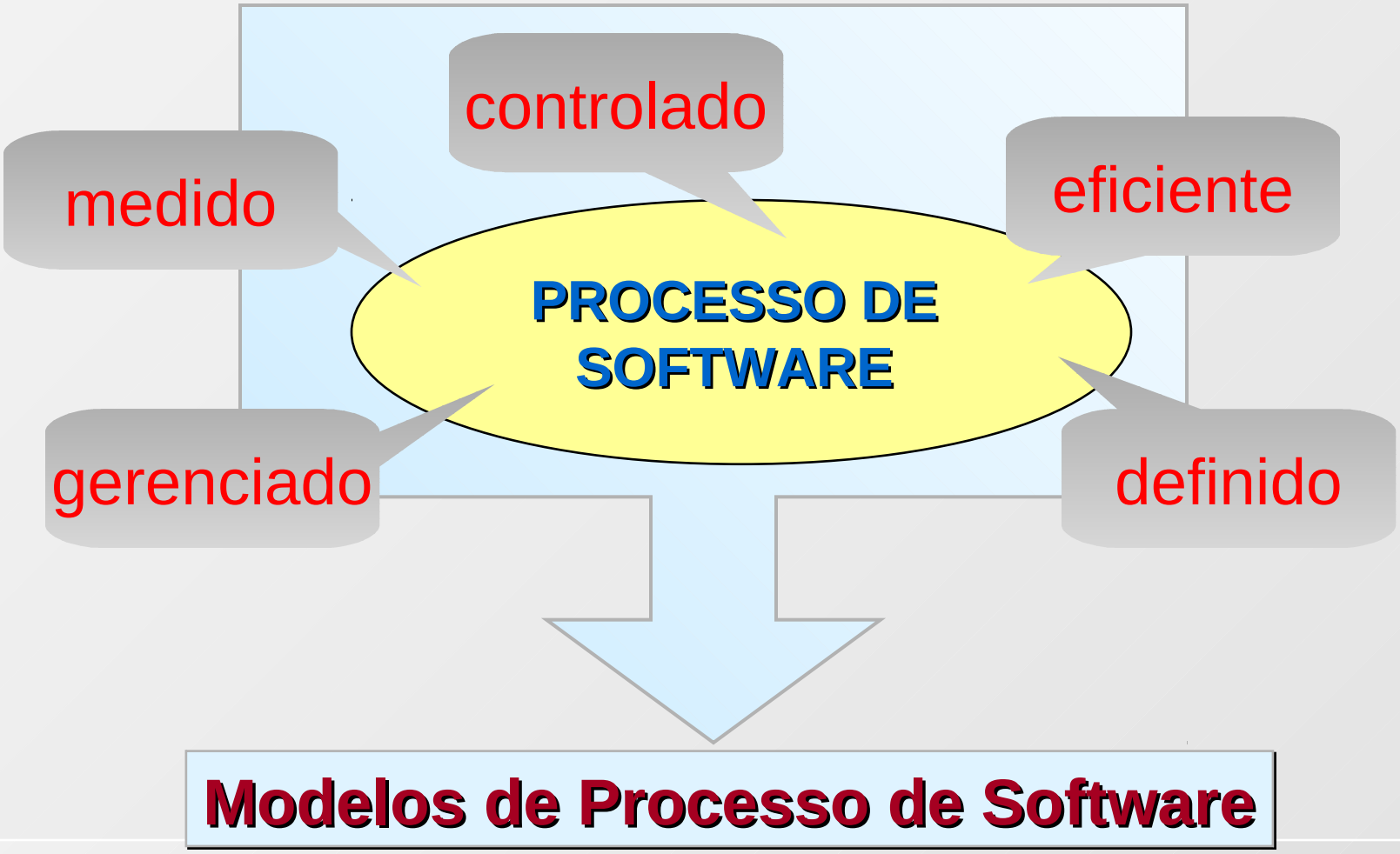
(*francine@icmc.usp.br*)

Processo de Software

Fases Genéricas



Um Processo de Software com Qualidade



Modelos de Processo de Software

- Existem vários modelos de processo de software.
  - Paradigmas de Engenharia de Software.
- Genéricos, abstrações do processo.
  - Usados para explicar diferentes abordagens para o desenvolvimento de software.
- Tentativa de colocar ordem em uma atividade inerentemente caótica.

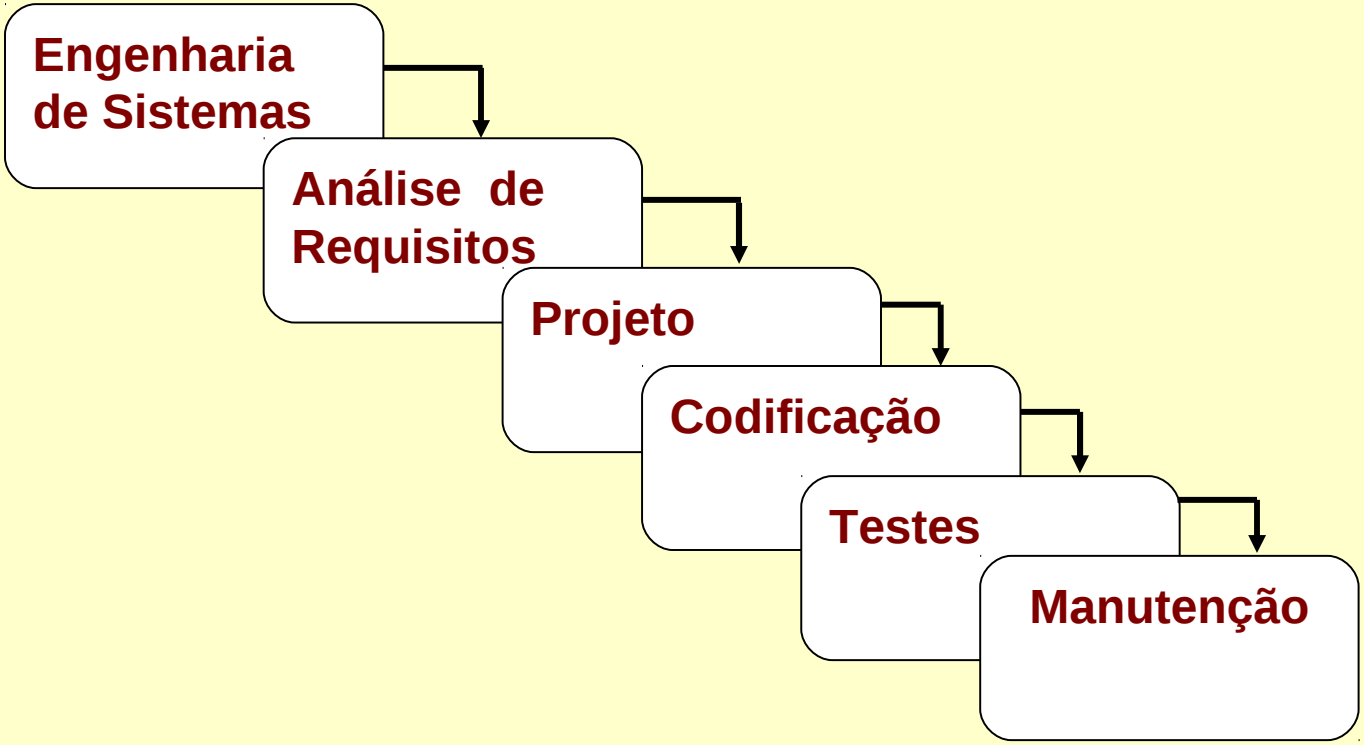
Tópicos da Aula

- Modelos de Processo de Software
  - Cascata
  - Prototipação
  - RAD
  - Evolutivos
    - Incremental
    - Espiral
    - Componentes
  - Métodos Formais
  - Técnicas de Quarta Geração
  - ...

O Modelo Cascata

- Modelo mais antigo e mais amplamente utilizado da Engenharia de Software.
  - Paradigma do Ciclo de Vida Clássico.
  - Modelo Seqüencial Linear.
- Sugere uma abordagem sistemática seqüencial para o desenvolvimento de software.
  - Tem início em nível de sistema e avança ao longo da análise, projeto, codificação, teste e manutenção.

O Modelo Cascata



O Modelo Cascata

Engenharia de Sistemas

- Estabelecimento de requisitos para todos os elementos do sistema e atribuição de um subconjunto desses requisitos para o software.
  - Visão essencial quando o software deve fazer interface com outros elementos (hardware, pessoas, banco de dados, etc.).
- Coleta de requisitos em nível do sistema, com uma pequena quantidade de projeto e análise de alto nível.

O Modelo Cascata

Análise de Requisitos de Software

- O processo de coleta dos requisitos é intensificado e concentrado especificamente no software.
- Para entender a natureza do programa a ser construído:
  - Deve-se compreender o domínio da informação do software, a função, o desempenho e a interface exigidos.
- Os requisitos (do sistema e do software) são documentados e revistos com o cliente.

O Modelo Cascata

Projeto

- Tradução dos requisitos do software para um conjunto de **representações** que podem ser avaliadas quanto à qualidade, antes que a codificação se inicie.
- Concentra-se em quatro atributos distintos do programa: ***estrutura de dados***, ***arquitetura do software***, ***representações da interface*** *e* ***detalhes procedimentais***.
- Assim como os requisitos, o projeto deve ser documentado e torna-se parte da configuração de software.

O Modelo Cascata

Codificação (Geração de Código)

- Tradução das representações do projeto para uma linguagem de programação, resultando em instruções executáveis pelo computador.
  - Se o projeto for executado detalhadamente, a codificação pode ser executada mecanicamente.

O Modelo Cascata

Testes

- Concentram-se nos aspectos lógicos internos do software, garantindo que todas as instruções tenham sido testadas.
- Concentram-se nos aspectos funcionais externos a fim de descobrir erros e garantir que as entradas definidas produzam resultados reais que estejam em conformidade com os resultados esperados.

O Modelo Cascata

Manutenção

- Provavelmente o software deverá sofrer mudanças depois que for entregue ao cliente.
- Causas das mudanças:
  - Erros.
  - Adaptação do software para acomodar mudanças em seu ambiente externo.
  - Exigência do cliente para melhoramentos funcionais e de desempenho.
- Reaplica cada uma das fases precedentes a um programa existente.

Modelo Cascata: Problemas

- Projetos reais raramente seguem o fluxo seqüencial que o modelo propõe.
  - Modificações podem causar confusão à medida que o projeto prossegue.
- É difícil para o cliente estabelecer explicitamente todos os requisitos logo no início do projeto.
  - Dificuldade do modelo em acomodar a incerteza natural que existe no começo do projeto.

Modelo Cascata: Problemas

- O cliente deve ter paciência. Uma versão executável do software só fica disponível em uma etapa avançada do desenvolvimento.
  - Um erro grosseiro pode ser desastroso, se não for detectado até que o programa executável seja revisto.

Embora o Modelo Cascata tenha fragilidades, ele é significativamente melhor do que uma abordagem casual ao desenvolvimento de software.

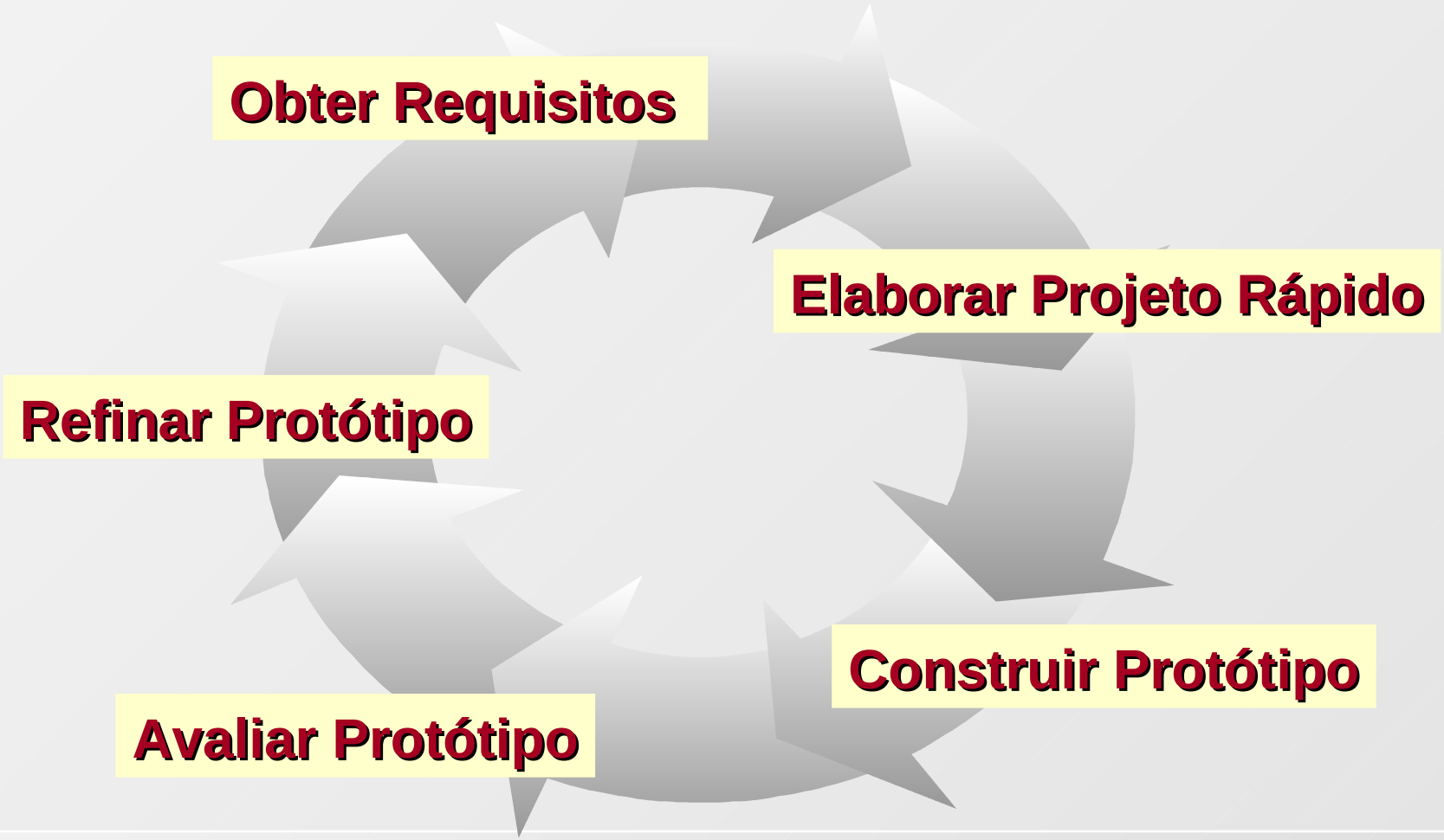
O Modelo Cascata

- Contribuições importantes do modelo ao processo de desenvolvimento de software:
  - Produz um padrão no qual os métodos para análise, projeto, codificação, testes e manutenção podem ser situados.
  - Impõe disciplina, planejamento e gerenciamento.
  - A implementação do produto deve ser postergada até que os objetivos tenham sido completamente entendidos.

O Modelo de Prototipação

- Possibilita que o desenvolvedor crie um modelo (protótipo) do software que deve ser construído.
- O objetivo é entender os requisitos do usuário e, assim, obter uma melhor definição dos requisitos do sistema.
  - Apropriado para quando o cliente não definiu detalhadamente os requisitos.

O Modelo de Prototipação



O Modelo de Prototipação

Obtenção dos Requisitos

- Desenvolvedor e cliente:
  - Definem os objetivos gerais do software.
  - Identificam quais requisitos são conhecidos.
  - Identificam as áreas que necessitam de definições adicionais.

O Modelo de Prototipação

Projeto Rápido

- Representação dos aspectos do software que são visíveis ao usuário.
  - Abordagens de entrada.
  - Formatos de saída.

O Modelo de Prototipação

Construção do Protótipo

- Implementação rápida do protótipo.

O Modelo de Prototipação

Avaliação do Protótipo

- Cliente e desenvolvedor avaliam o protótipo.

O Modelo de Prototipação

Refinamento do Protótipo

- Cliente e desenvolvedor refinam os requisitos do software a ser desenvolvido.

O Modelo de Prototipação

Construção do Produto

- Identificados os requisitos, o protótipo deve ser descartado e a versão de produção deve ser construída considerando os critérios de qualidade.

Prototipação: Problemas

- O cliente não sabe que o software que ele vê não considerou, durante o desenvolvimento:
  - Qualidade global.
  - Manutenibilidade a longo prazo.
- O desenvolvedor freqüentemente faz uma implementação comprometida.
  - Utilizando o que está disponível com o objetivo de produzir rapidamente um protótipo.

O Modelo de Prototipação

- Ainda que possam ocorrer problemas, a prototipação é um ciclo de vida eficiente.
- *Definir as regras do jogo logo no começo.*
  - O cliente e o desenvolvedor devem concordar que o protótipo será construído para servir como um mecanismo para definição dos requisitos.
    - Deve ser descartado (ao menos em parte).
    - O software real é submetido à engenharia visando à qualidade e à manutenibilidade.

O Modelo RAD

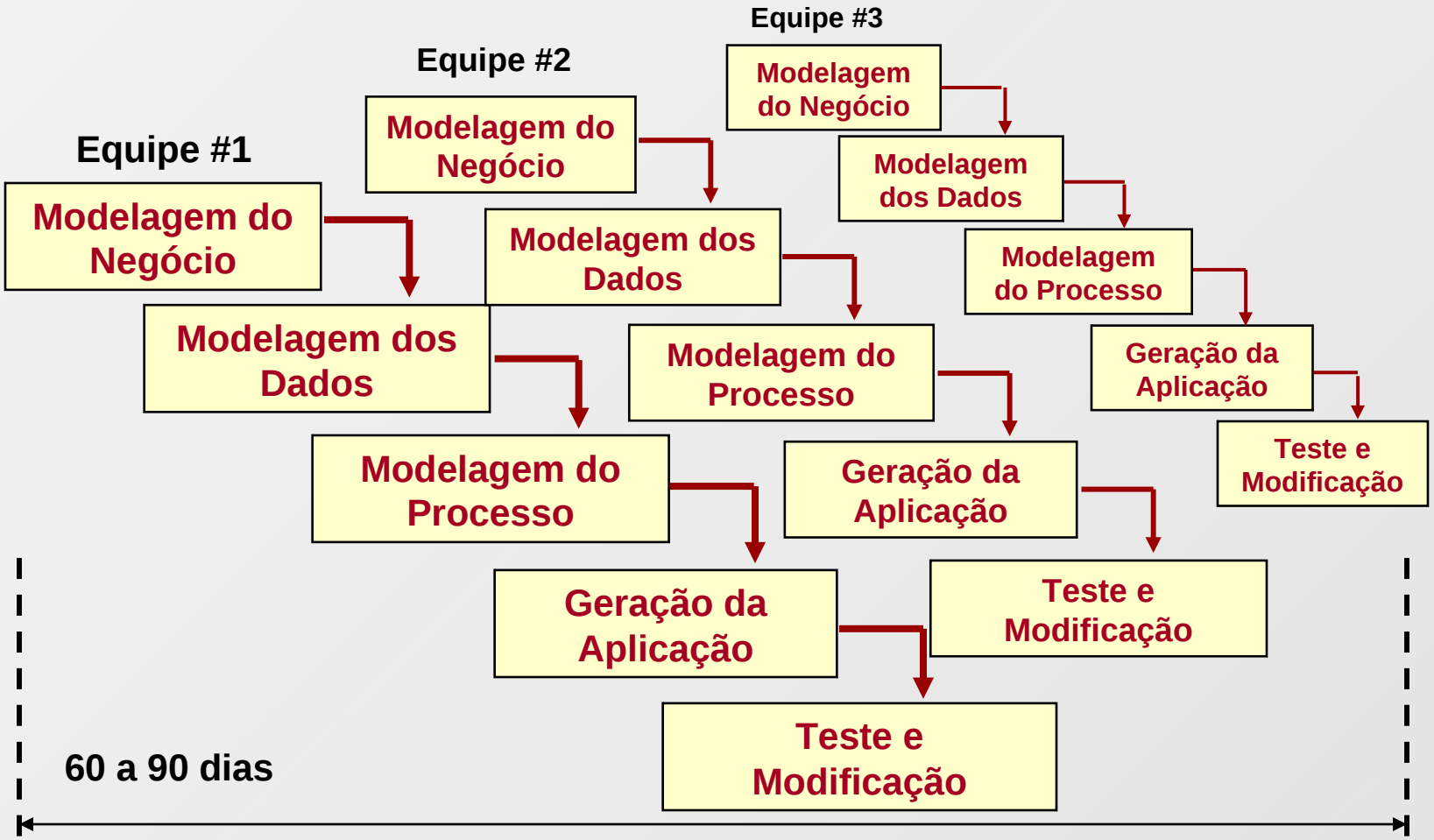
- RAD (*Rapid Application Development*) é um modelo seqüencial linear que enfatiza um ciclo de desenvolvimento extremamente curto.

- Trata-se de uma adaptação de “alta velocidade” do modelo seqüencial linear.

O Modelo RAD

- A abordagem engloba as seguintes fases:
  1. Modelagem do Negócio
  2. Modelagem dos Dados
  3. Modelagem do Processo
  4. Geração da Aplicação
  5. Teste e Modificação

O Modelo RAD


Equipe #1
Modelagem do Negócio
Modelagem dos Dados
Modelagem do Processo
Geração da Aplicação
Teste e Modificação
Equipe #2
Modelagem do Negócio
Modelagem dos Dados
Modelagem do Processo
Geração da Aplicação
Teste e Modificação
Equipe #3
Modelagem do Negócio
Modelagem dos Dados
Modelagem do Processo
Geração da Aplicação
Teste e Modificação
60 a 90 dias

O Modelo RAD

1- Modelagem do Negócio

- O fluxo de informação entre as funções de negócio é modelado de modo que responda as seguintes questões:
  - Que informação direciona o processo de negócio?
  - Que informação é gerada?
  - Quem a gera?
  - Para onde vai a informação?
  - Quem a processa?

O Modelo RAD

2- Modelagem dos Dados

- O fluxo de informação definido como parte da fase de modelagem dos negócios é refinado em um conjunto de objetos de dados necessários para apoiar os negócios.

- As características (atributos) de cada objeto são identificadas e o relacionamento entre esses objetos é definido.

O Modelo RAD

3- Modelagem do Processo

- Os objetos de dados definidos na fase de modelagem dos dados são transformados para obter o fluxo de informação necessário para implementar uma função de negócio.

- As descrições de processamento são criadas adicionando, modificando, excluindo ou recuperando objetos de dados.

O Modelo RAD

4- Geração da Aplicação

- *RAD* assume o uso de técnicas de 4ª geração (ferramentas automatizadas para facilitar a construção do software).
  - Delphi, Visual Basic, Asp.net, etc.
- O processo *RAD* trabalha para reutilizar componentes de programa existentes (quando possível) ou criar componentes reutilizáveis.

O Modelo RAD

5- Teste e Modificação

- Como o processo RAD enfatiza a reutilização, muitos dos componentes do programa já foram testados.
  - Isso reduz o tempo de teste global.
  - Entretanto… os novos componentes devem ser testados e todas as interfaces devem ser exaustivamente exercitadas.

O Modelo RAD

- Se uma aplicação de negócio puder ser modularizada, de modo que possibilite que cada função principal seja completada em um curto prazo (por ex: 30 a 90 dias), ela é candidata ao RAD.
  - Cada função principal pode ser tratada por uma equipe RAD distinta e depois integrada para formar o todo.

Modelo RAD: Problemas

- Para projetos grandes (mas escaláveis), RAD exige recursos humanos suficientes para criar um número adequado de equipes.
- Exige que desenvolvedores e clientes estejam comprometidos com atividades continuamente rápidas, necessárias para obter um sistema completo em um período muito curto.

Modelo RAD: Problemas

- Nem todos os tipos de aplicação são apropriadas para o RAD.
  - Deve ser possível a modularização efetiva da aplicação.
    - Se o sistema não puder ser adequadamente modularizado, a construção dos componentes será problemática.

Modelo RAD: Problemas

- Quando riscos técnicos forem elevados, o RAD não é adequado.
  - A nova aplicação faz uso intenso de novas tecnologias.
  - O novo software exige um alto grau de interoperabilidade com programas existentes.

Modelos Evolucionários (Evolutivos)

- Existem situações em que a engenharia de software necessita de um modelo de processo que possa acomodar um produto que evolui com o tempo.

Modelos Evolucionários (Evolutivos)

- Requisitos de produto e de negócio mudam conforme o desenvolvimento prossegue.
- Requisitos básicos são bem conhecidos, mas os detalhes ainda devem ser definidos.
- Prazos reduzidos de mercado tornam impossível a conclusão de um produto completo.
  - Uma versão reduzida pode ser elaborada face à competitividade ou às pressões do negócio.

Modelos Evolucionários (Evolutivos)

- Modelo Cascata: desenvolvimento retilíneo.
  - O sistema completo estará pronto depois que a seqüência linear for concluída.
- Prototipação: entendimento dos requisitos.
  - Em geral, não é projetado para ajudar um sistema em produção.

A natureza evolutiva do software não é considerada.

Modelos Evolucionários (Evolutivos)

- Modelos evolutivos são iterativos.
  - Possibilitam o desenvolvimento de versões cada vez mais completas do software.
    - *Modelo Incremental*
    - *Modelo Espiral*
    - *Modelo de Desenvolvimento Concorrente*

O Modelo Incremental

- O Modelo Incremental combina:
  - Elementos do Modelo Cascata (aplicado repetidamente).
  - A filosofia iterativa da Prototipação.

O Modelo Incremental

- Objetivo: trabalhar junto ao cliente para descobrir seus requisitos, de maneira incremental, até que o produto final seja obtido.
  - A evolução acontece quando novas características são adicionadas à medida que são sugeridas pelo cliente.

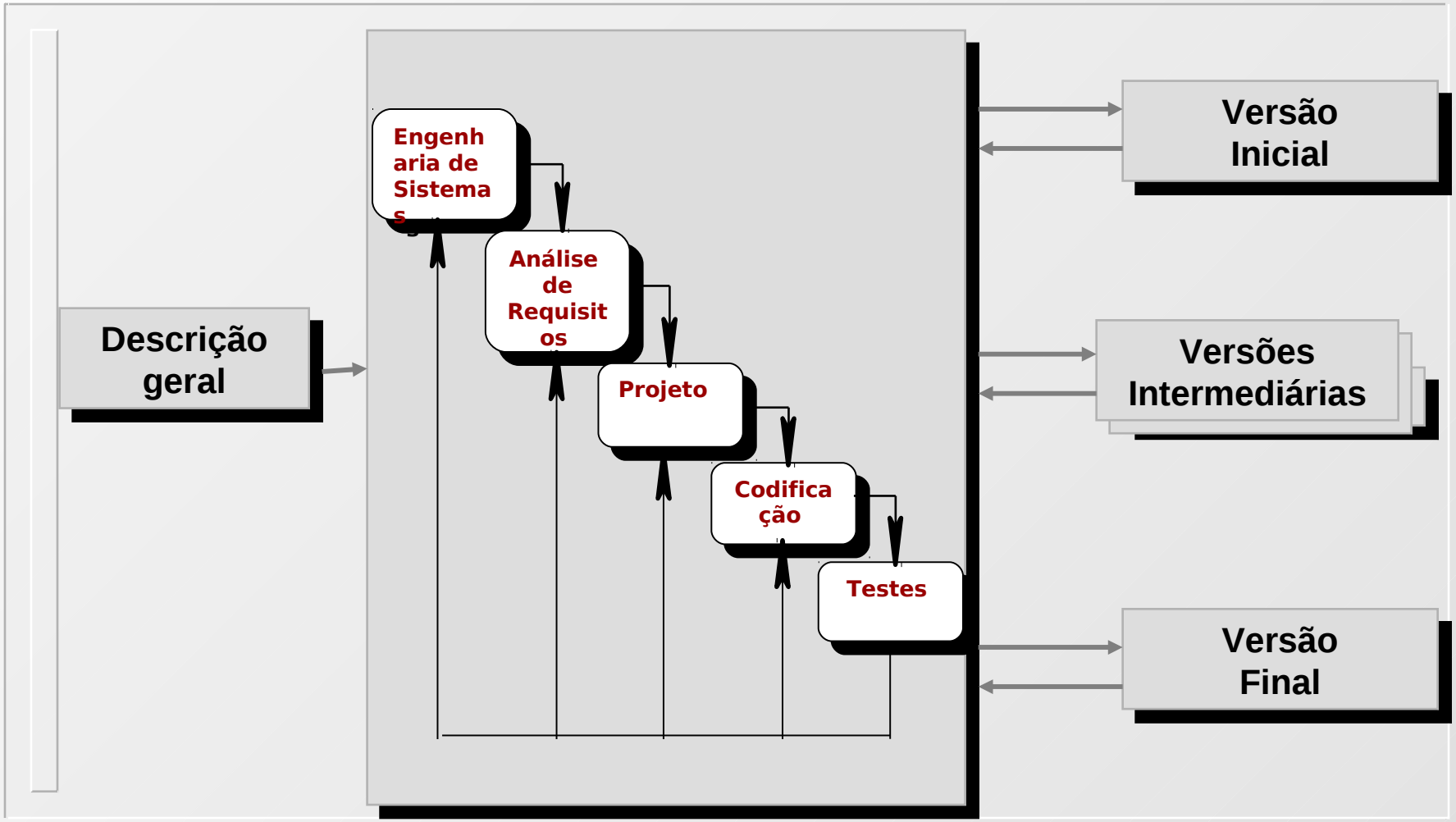
O Modelo Incremental

- Clientes identificam, em um esboço, as funções a serem fornecidas pelo sistema.
  - Funções mais e menos importantes.
- Definem-se estágios de entrega.
  - Cada estágio fornece um subconjunto das funcionalidades do sistema.
    - Incremento.

O Modelo Incremental

- Os incrementos são elaborados utilizando-se o processo de desenvolvimento mais apropriado.
- Incrementos concluídos são entregues ao cliente e colocados em operação.

O Modelo Incremental



O Modelo Incremental

- A versão inicial é frequentemente o **núcleo** do produto (a parte mais importante).
- A funcionalidade do sistema **melhora** a cada novo estágio que é entregue.
  - Os incrementos são **versões simplificadas** do produto final.
    - Oferecem **capacidades** que servem ao usuário.
    - Constituem uma plataforma para **avaliação**.

O Modelo Incremental

- Importante quando é difícil estabelecer *a priori* uma especificação detalhada dos requisitos.
- Útil quando não há mão-de-obra disponível para uma implementação completa, dentro do prazo de entrega estabelecido.
- Os incrementos podem ser planejados de modo que os riscos técnicos possam ser gerenciados.

Modelo Incremental: Problemas

- Incrementos devem ser relativamente pequenos e produzir alguma funcionalidade para o sistema.
  - Difícil mapear os requisitos do cliente dentro de incrementos de tamanho correto.
- Difícil identificar facilidades comuns, exigidas por todos os incrementos.

Programação Extrema (*eXtreme Programming*)

- Evolução da abordagem incremental.

  (*Kent Beck, 1999*)

  - Voltada para equipes de até 20 pessoas, engajadas no desenvolvimento de software cujos requisitos são vagos ou se encontram em constante mudança.

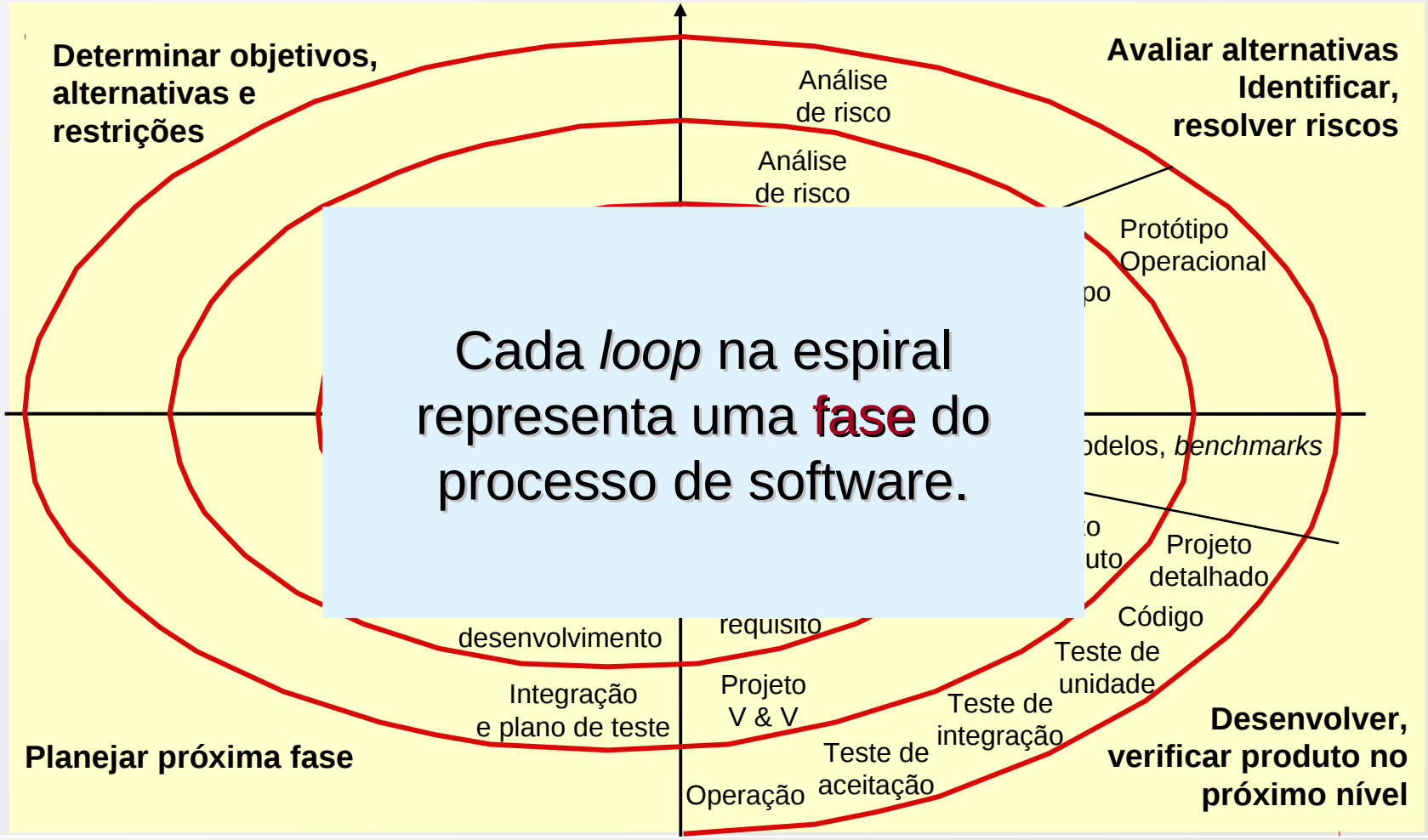
Programação Extrema (*eXtreme Programming*)

- Desenvolvimento e entrega de incrementos de funcionalidade muito pequenos.
- Envolvimento do cliente no processo.
- Constante melhoria de código.
- Programação em pares.
- 40 horas de trabalho.
- *Refactoring*
  - Abordagem disciplinada para tornar o código de um software mais claro e de fácil manutenção, minimizando a probabilidade de inclusão de erros.
- *Test first, code later.*

O Modelo Espiral

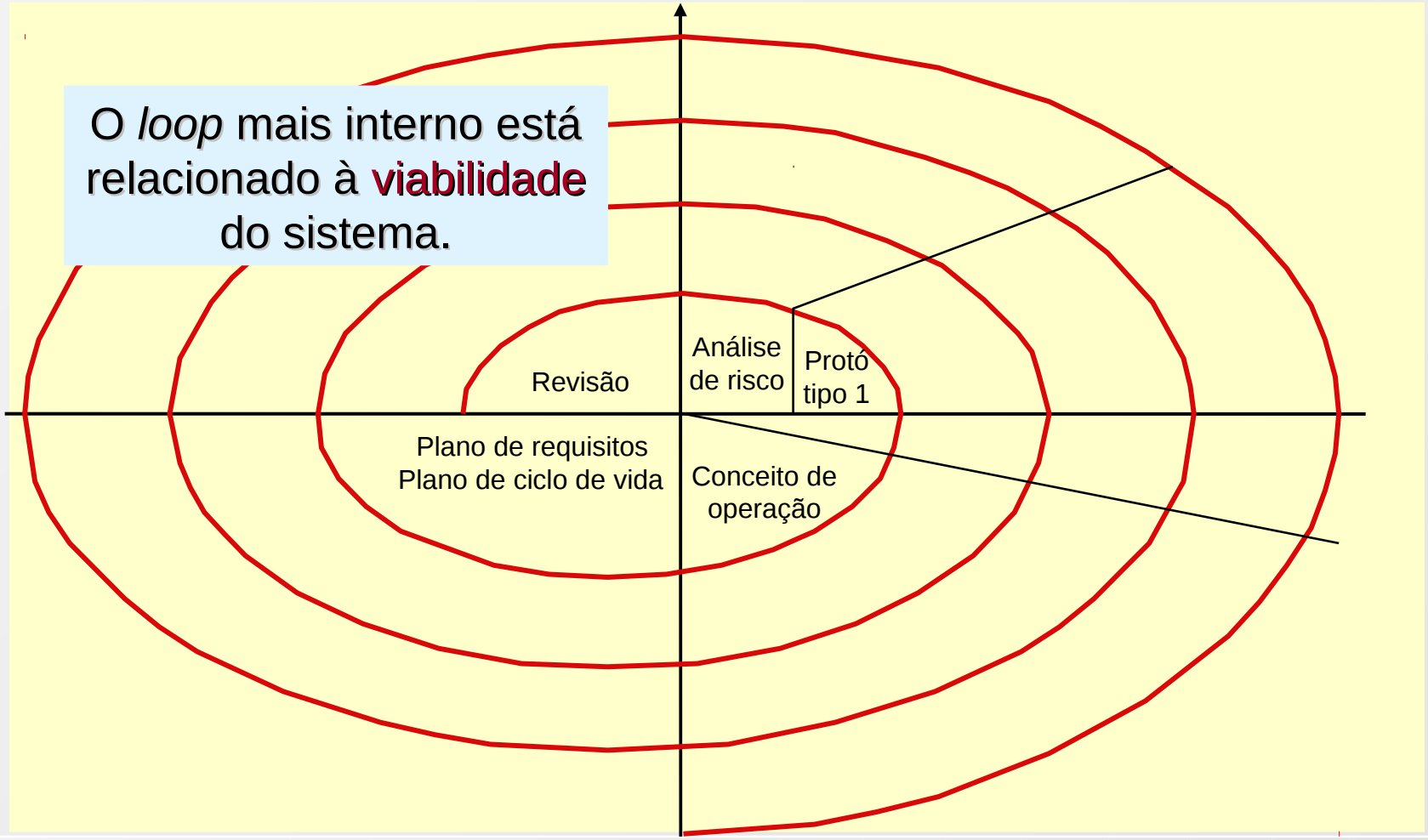
- O Modelo Espiral combina:
  - A natureza iterativa da Prototipação.
  - Os aspectos controlados e sistemáticos do Modelo Cascata.
- Fornece potencial para o desenvolvimento rápido de versões incrementais do software.
- Dividido em um certo número de atividades ou regiões de tarefa.
  - Existem, tipicamente, de 3 a 6 regiões de tarefa.

O Modelo Espiral



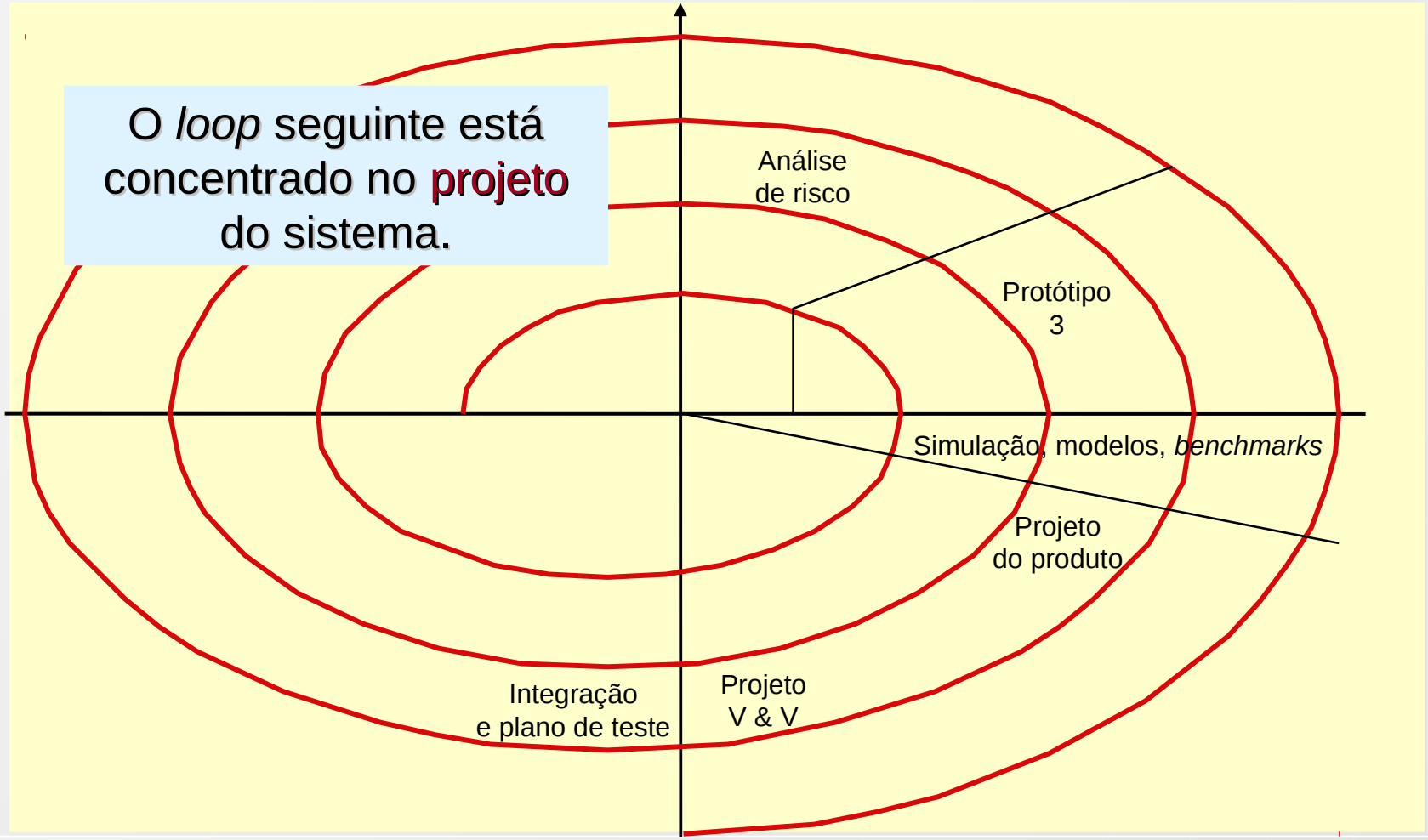
Cada *loop* na espiral representa uma fase do processo de software.

O Modelo Espiral

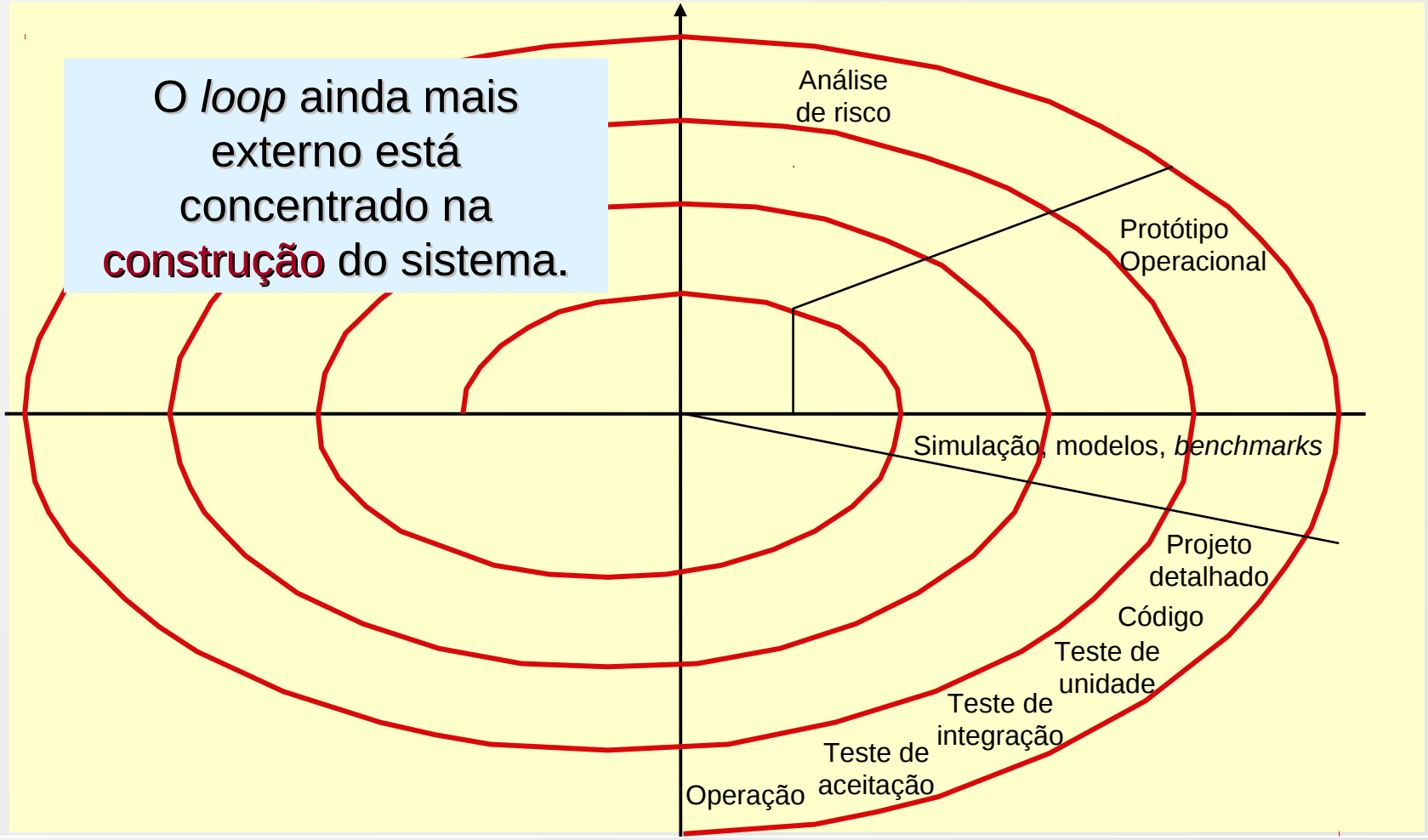


O Modelo Espiral

O Modelo Espiral

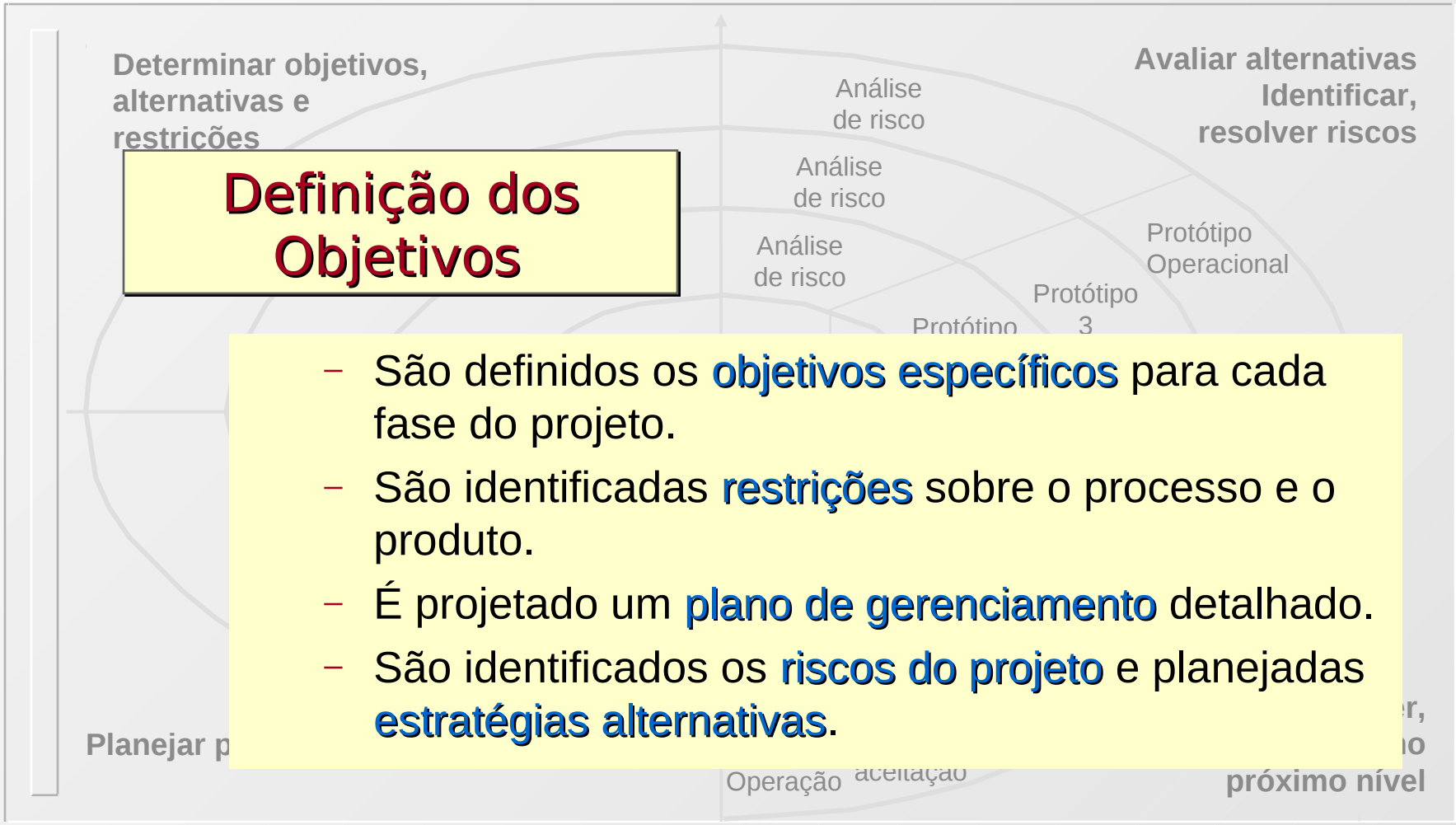


O Modelo Espiral



O Modelo Espiral

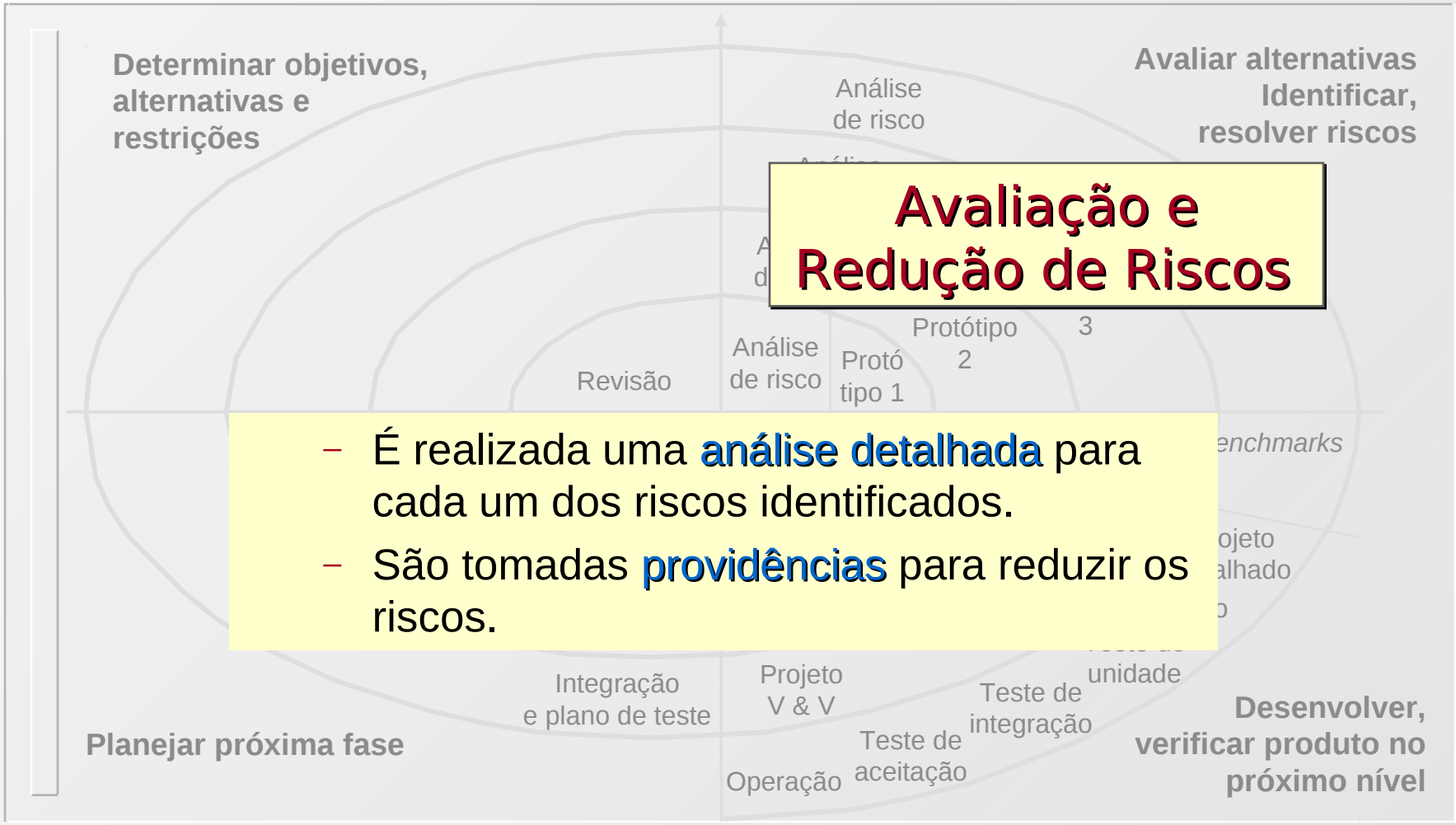
O Modelo Espiral



Definição dos Objetivos

- São definidos os **objetivos específicos** para cada fase do projeto.
- São identificadas **restrições** sobre o processo e o produto.
- É projetado um **plano de gerenciamento** detalhado.
- São identificados os **riscos do projeto** e planejadas **estratégias alternativas**.

O Modelo Espiral



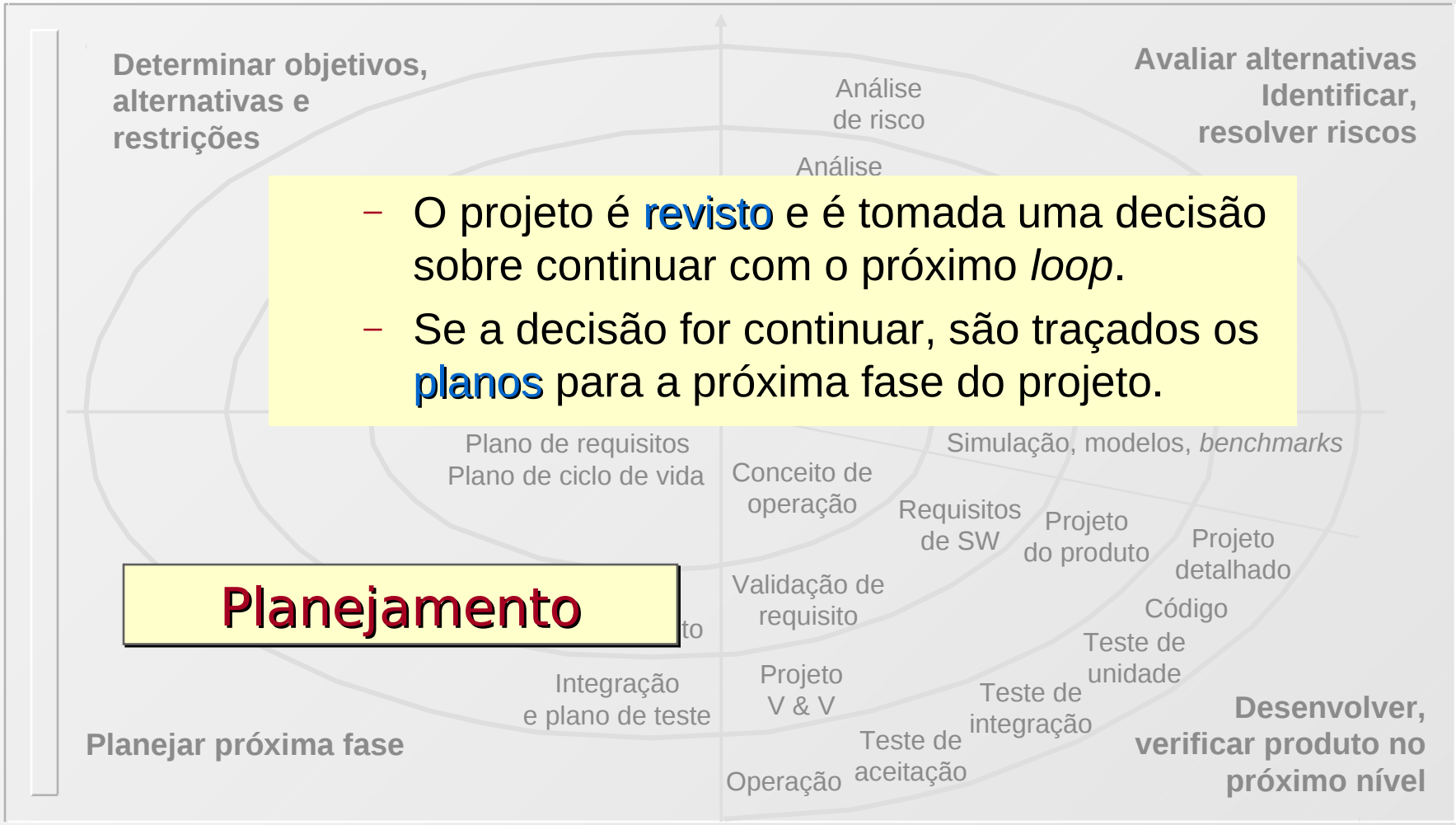
Avaliação e Redução de Riscos

- É realizada uma análise detalhada para cada um dos riscos identificados.
- São tomadas providências para reduzir os riscos.

O Modelo Espiral

- Depois da avaliação dos riscos, é escolhido um modelo de desenvolvimento para o sistema.

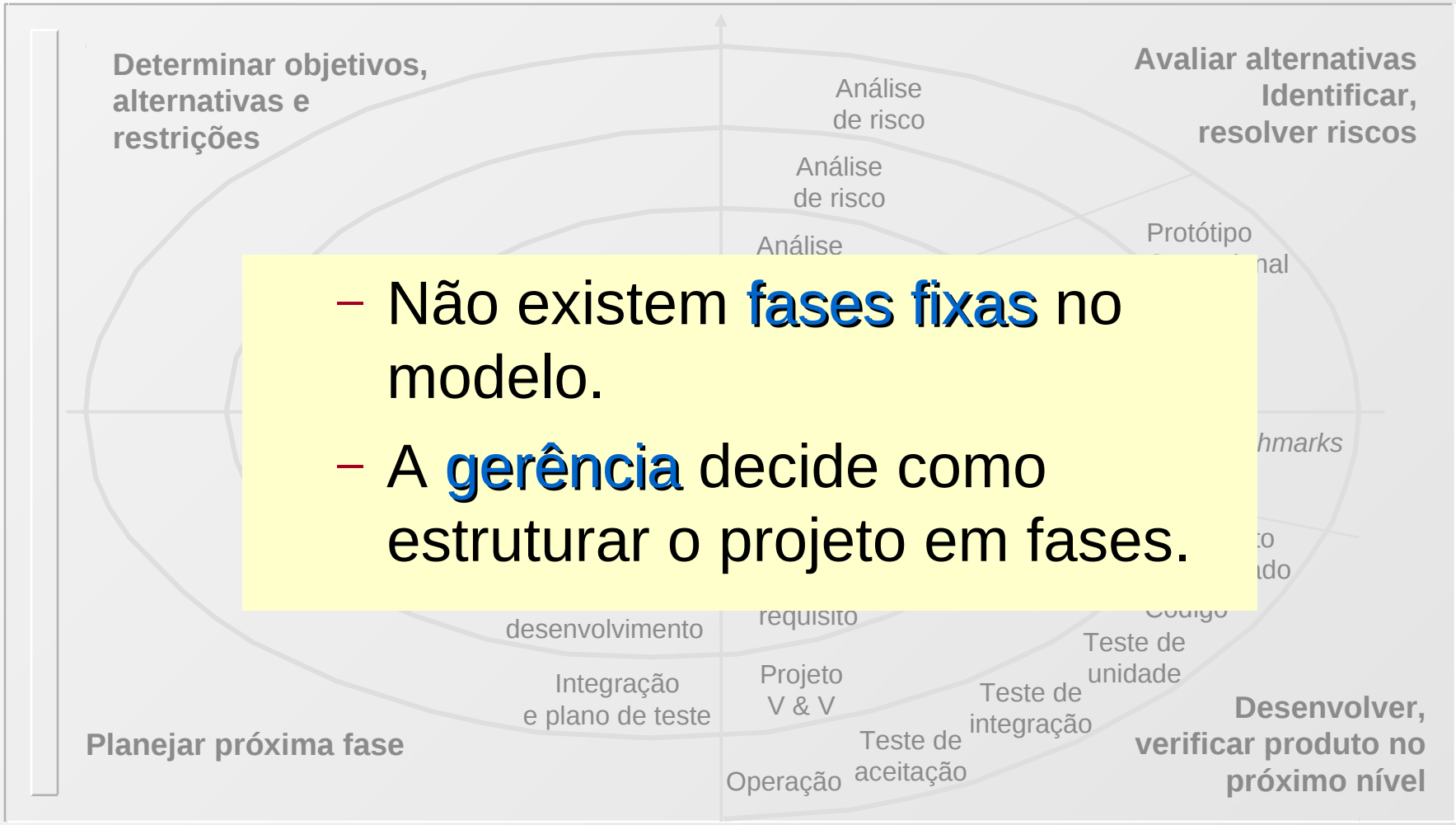
O Modelo Espiral



Planejamento

- O projeto é revisto e é tomada uma decisão sobre continuar com o próximo *loop*.
- Se a decisão for continuar, são traçados os planos para a próxima fase do projeto.

O Modelo Espiral



- Não existem fases fixas no modelo.
- A gerência decide como estruturar o projeto em fases.

O Modelo Espiral

- Engloba as melhores características do Modelo Cascata e da Prototipação, adicionando um novo elemento: a Análise de Riscos.
  - Risco: algo que pode dar errado.
  - Segue a abordagem de passos sistemáticos do Modelo Cascata, incorporando-os numa estrutura iterativa.
  - Usa a Prototipação em qualquer etapa da evolução do produto, como mecanismo de redução de riscos.

Modelo Espiral

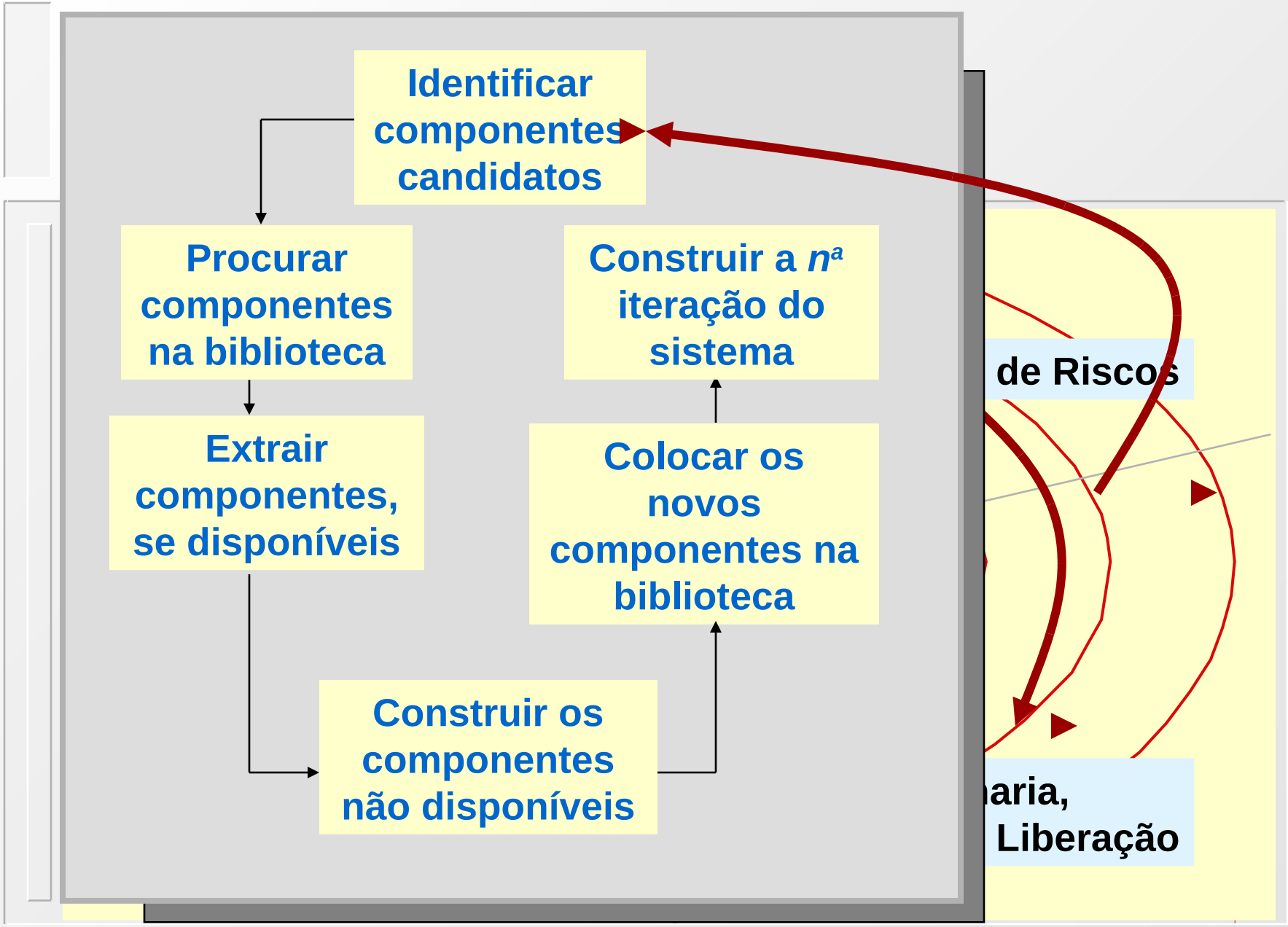
- É, atualmente, a abordagem mais realística para o desenvolvimento de sistemas e software de grande porte.
- Exige a consideração direta dos riscos técnicos em todos os estágios do projeto.
  - Se aplicado adequadamente, pode reduzir os riscos antes que os mesmos fiquem problemáticos.

Modelo Espiral: Problemas

- 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.

O Modelo de Montagem de Componentes

- Incorpora características de tecnologias orientadas a objeto no Modelo Espiral.
- É de natureza evolutiva demandando uma abordagem iterativa para a criação do software.
- O modelo compõe aplicações a partir de componentes de software “empacotados”.
  - Quando projetadas e implementadas apropriadamente, as classes são reutilizáveis em diferentes aplicações e arquiteturas de sistema.


Identificar componentes candidatos
Procurar componentes na biblioteca
Extrair componentes, se disponíveis
Construir os componentes não disponíveis
Colocar os novos componentes na biblioteca
Construir a *nª* iteração do sistema
de Riscos
aria,
Liberação

O Modelo de Montagem de Componentes

- O modelo de montagem de componentes conduz ao reuso do software.

- A reusabilidade pode fornecer uma série de benefícios:
  - Redução no tempo de desenvolvimento.
  - Redução no custo do projeto.
  - Aumento no índice de produtividade.
    - Tais benefícios dependem da robustez da biblioteca de componentes.

O Modelo de Métodos Formais

- Especificar, desenvolver e verificar um sistema pela aplicação de uma rigorosa notação matemática.
  - Abrange um conjunto de atividades que levam à especificação matemática formal do software.
  - Tem como base a transformação matemática formal de uma especificação de sistema em um programa executável.

O Modelo de Métodos Formais

$$
\begin{array}{l}
\textit{LearningOrg} \\
\hline
\textit{FormalOrg} \\
\textit{learningteams} : \mathbb{P}\, \textit{LearningTeam} \\
\textit{sharedvisions} : \mathbb{P}\, \textit{SharedVision} \\
\hline
\textit{sharedvisions} \neq \\
\forall\, lt : \textit{LearningTeam} \mid lt \in \textit{learningteams} \bullet \\
\qquad (\exists_1\, te : \textit{Team} \mid te \in \textit{teams} \bullet \\
\qquad te = (\lambda\, \textit{LearningTeam} \bullet \theta\, \textit{Team})\, lt) \\
\#\textit{teams} = \#\textit{learningteams} \\
\hline
\end{array}
$$

O Modelo de Métodos Formais

- No projeto...
  - Base para a verificação do programa, permitindo que se descubram erros que poderiam passar despercebidos.
- No desenvolvimento...
  - Ambigüidade, não completitude e inconsistência podem ser mais facilmente descobertas pela análise matemática.

O Modelo de Métodos Formais

- Abordagem particularmente adequada ao desenvolvimento de sistemas com rigorosas exigências de segurança, confiabilidade e garantia.
  - Eletrônica de aeronaves.
  - Controle de tráfego aéreo.
  - Dispositivos médicos.
  - Telecomunicações.

Modelo de Métodos Formais: Problemas

- Preocupações quanto à aplicabilidade em ambientes comerciais:
  - O desenvolvimento é lento e dispendioso.
  - Requer treinamento extensivo.
  - Difícil utilizar os modelos como um mecanismo de comunicação.
    - Clientes despreparados tecnicamente.

Cleanroom (Sala Limpa)

- Exemplo mais conhecido do processo de desenvolvimento formal.
  - Desenvolvimento incremental do software.
  - Cada estágio é desenvolvido e sua correção é demonstrada utilizando-se uma abordagem formal.
  - O teste de sistema visa a medir a confiabilidade do sistema.
    - Conduzido como um experimento estatístico.

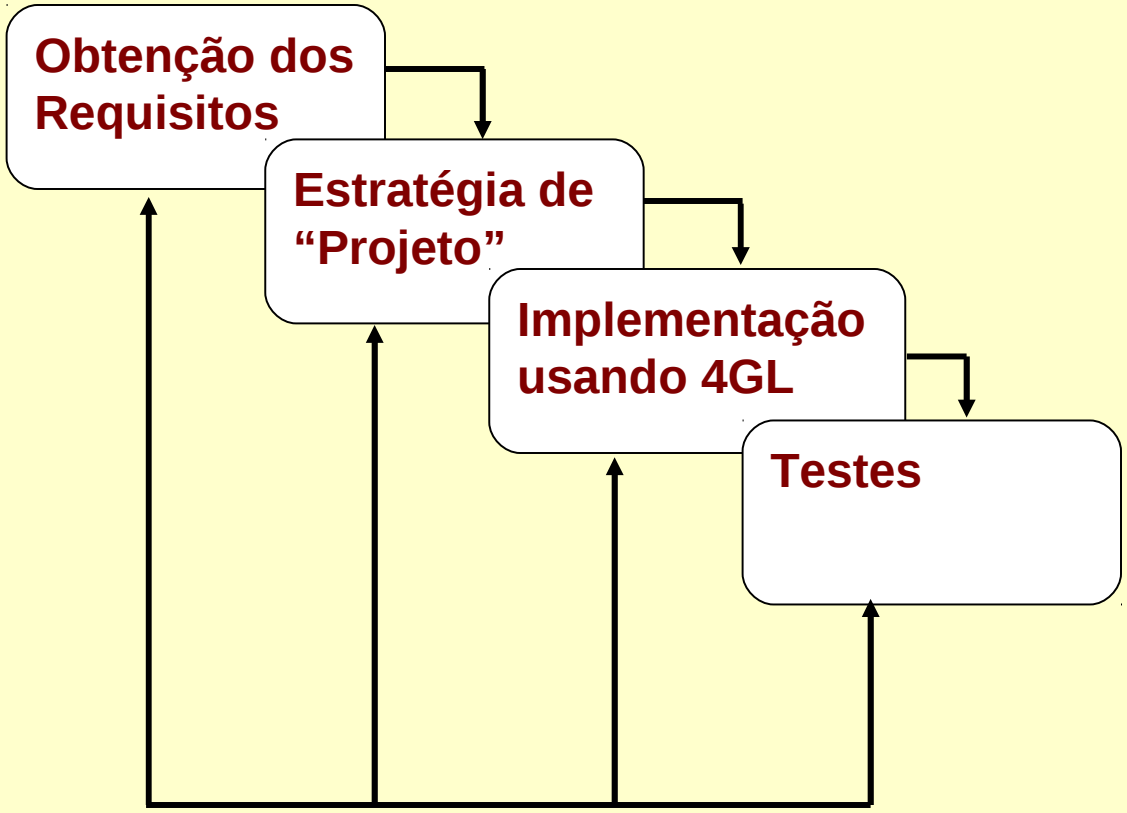
Técnicas de Quarta Geração

- Engloba um conjunto de ferramentas de software possibilitando que:
  - O sistema seja especificado em uma linguagem de alto nível.
  - O código-fonte seja gerado automaticamente a partir dessas especificações.

Técnicas de Quarta Geração

- Ferramentas de Apoio
  - Linguagens não-procedimentais para:
    - Consulta de banco de dados (*SQL*).
    - Geração de relatórios (*PostScript*).
    - Manipulação de dados (*XML*).
    - Interação e definição de telas (*Delphi, Visual Basic*).
    - Geração de código.
  - Capacidade gráfica de alto nível.
  - Capacidade de disposição em planilhas.
  - Geração automática de HTML e linguagens similares para criação de páginas Web.

Técnicas de Quarta Geração



Técnicas de Quarta Geração

Obtenção dos Requisitos

- O cliente descreve os requisitos os quais são traduzidos para um protótipo operacional.
  - O cliente pode estar inseguro quanto aos requisitos.
  - O cliente pode ser incapaz de especificar as informações de um modo que uma ferramenta 4GL possa ser utilizada.
  - As 4GLs atuais não são suficientemente sofisticadas para acomodar a verdadeira linguagem natural.

Técnicas de Quarta Geração

Estratégia de “Projeto”

- Pequenas Aplicações
  - É possível mover-se do passo de Obtenção dos Requisitos para o passo de Implementação usando uma linguagem de quarta geração.
- Grandes Projetos
  - É necessário desenvolver uma estratégia de projeto.
  - De outro modo ocorrerão os mesmos problemas encontrados quando se usa abordagem convencional (baixa qualidade).

Técnicas de Quarta Geração

Implementação usando 4GL

- Os resultados desejados são representados de modo que haja geração automática de código.
- Deve existir uma estrutura de dados com informações relevantes e que seja acessível pela 4GL.

Técnicas de Quarta Geração

Testes

- O desenvolvedor deve efetuar testes e desenvolver uma documentação significativa.
- O software desenvolvido deve ser construído de maneira que a manutenção possa ser efetuada prontamente.

Técnicas de Quarta Geração

- Proponentes
  - Redução dramática no tempo de desenvolvimento do software.
    - Aumento de produtividade.
- Oponentes
  - As 4GL atuais não são mais fáceis de usar do que as linguagens de programação.
  - O código-fonte produzido é ineficiente.
  - A manutenibilidade de sistemas usando técnicas 4GT ainda é questionável.

Metodologias Ágeis

- SCRUM
- Crystal/Clear
- FDD (*Feature Driven Development*)
  - Desenvolvimento Orientado a Funcionalidades
- DSDM (*Dynamic Systems Development Method*)
  - Método Dinâmico de Desenvolvimento de Sistemas
- ASD (*Adaptive Software Development*)
  - Desenvolvimento Adaptável de Software

Metodologias Ágeis

- XP (*eXtreme Programming*)
- Pragmatic Programming
- *Open Source Software Development*

- RUP (*Rational Unified Process*)

Concluindo...

- *Cascata, Prototipação, Evolutivos, Formais, Ágeis, ...*
- Para escolha de um modelo de processo de software:
  - Natureza do projeto e da aplicação.
  - Métodos e ferramentas a serem usados.
  - Controles e produtos que precisam ser entregues.
- Não são mutuamente exclusivos e freqüentemente são usados em conjunto.