Introdução à Engenharia de Software

INF-301

Instituto de Computação

Unicamp

Ariadne M. B. R. Carvalho

Conceitos básicos

Sistema:

- conjunto de elementos concretos ou abstratos entre os quais se pode encontrar alguma relação.

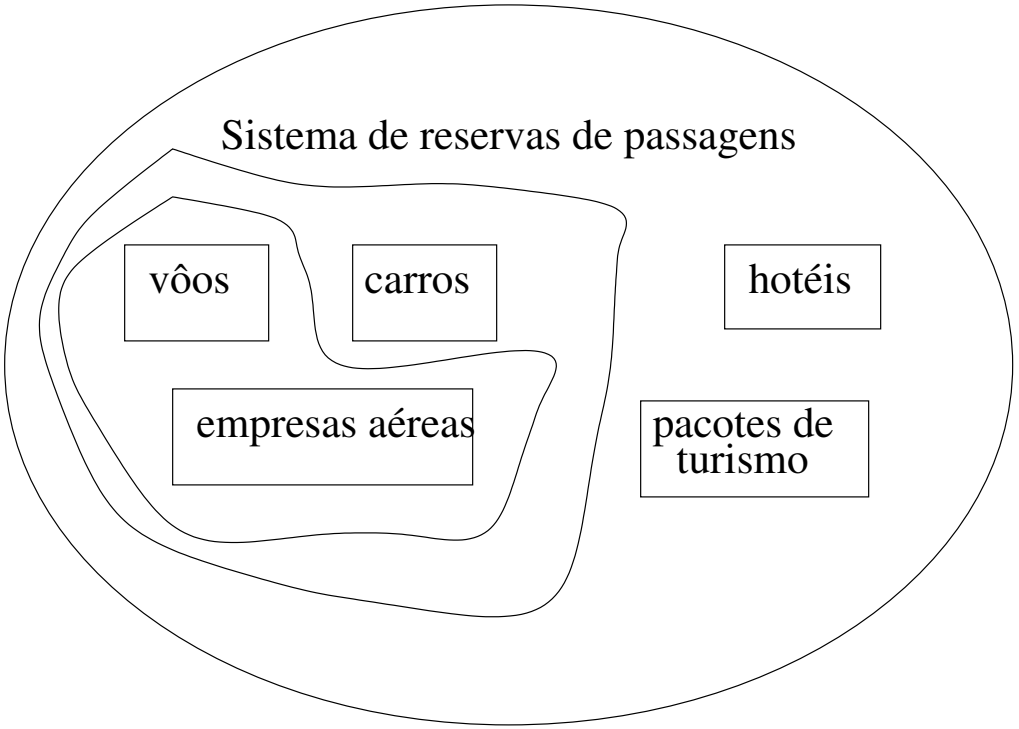
Elementos do sistema

- Estão dentro da fronteira conceitual
- Possuem interações fortes entre si
- Possuem interações fracas com os elementos externos ao sistema

Fronteira conceitual

- Separa o que está dentro do sistema do resto
- Elementos pertencentes ao sistema devem ser detalhados
- Elementos do ambiente externo $\rightarrow$ verificar sua interação

Exemplos de fronteira conceitual
Sistema de reservas de passagem aérea



Produtos de software

- sistemas de software entregues ao usuário com a documentação que descreve como instalar e usar o sistema

Duas categorias

- sistemas genéricos, produzidos e vendidos no mercado a qualquer pessoa que possa comprá-los;
- sistemas específicos, encomendados especialmente por um determinado cliente.

Produto de software

- *instruções* (programas de computador) que, quando executadas, realizam as funções e têm o desempenho desejados;
- *estruturas de dados* que possibilitam às instruções manipular as informações de forma adequada;
- *documentos* que descrevem as operações e uso do produto.

Tipos de sistemas de software

- Sistemas legados
- Sistemas de tempo real
- Sistemas embarcados (embutidos)

Ciclo de vida do produto de software

- começa na concepção do problema (solicitação do usuário)
- termina quando o sistema sai de uso

Processo de desenvolvimento de software

- O que é
  - Conjunto de atividades + resultados associados
- Objetivo
  - Construir o produto de software
- Atividades principais
  - *desenvolvimento*: as funcionalidades e as restrições são especificadas, e o software é produzido
  - *validação*: o software é validado
  - *manutenção*: o software sofre correções, adaptações e ampliações

Engenharia de software

- Uma disciplina que reúne
  - metodologias
  - métodos
  - ferramentas
- Metodologias → seguem métodos → que utilizam ferramentas
- Visam resolver problemas inerentes
  - ao processo; e
  - ao produto de software

Princípios da engenharia de software

- Formalidade: produtos mais confiáveis, controlar seu custo e mais confiança no seu desempenho.
- Abstração: identificar os aspectos importantes ignorando os detalhes.
- Decomposição: subdividir o processo em atividades específicas, atribuídas a diferentes especialistas.
- Generalização: sendo mais geral, é bem possível que a solução possa ser reutilizada.
- Flexibilização: modificação com facilidade.

Modelos de processos

- especificam as atividades que, de acordo com o modelo, devem ser executadas, assim como a ordem em que devem ser executadas.
- produtos de software podem ser construídos utilizando-se diferentes modelos de processos.
- alguns modelos são mais adequados que outros para determinados tipos de aplicação.
- a opção por um determinado modelo deve ser feita levando-se em consideração o produto a ser desenvolvido.

Objetivo dos modelos

- auxiliar no processo de produção $\rightarrow$ produtos de alta qualidade, produzidos mais rapidamente e a um custo cada vez menor.
- atributos: complexidade, visibilidade, aceitabilidade, confiabilidade, manutenibilidade, segurança etc.
- possibilitam:
  1. ao gerente: controlar o processo de desenvolvimento de sistemas de software
  2. ao desenvolvedor: obter a base para produzir, de maneira eficiente, software que satisfaça os requisitos pré-estabelecidos.

Exemplos

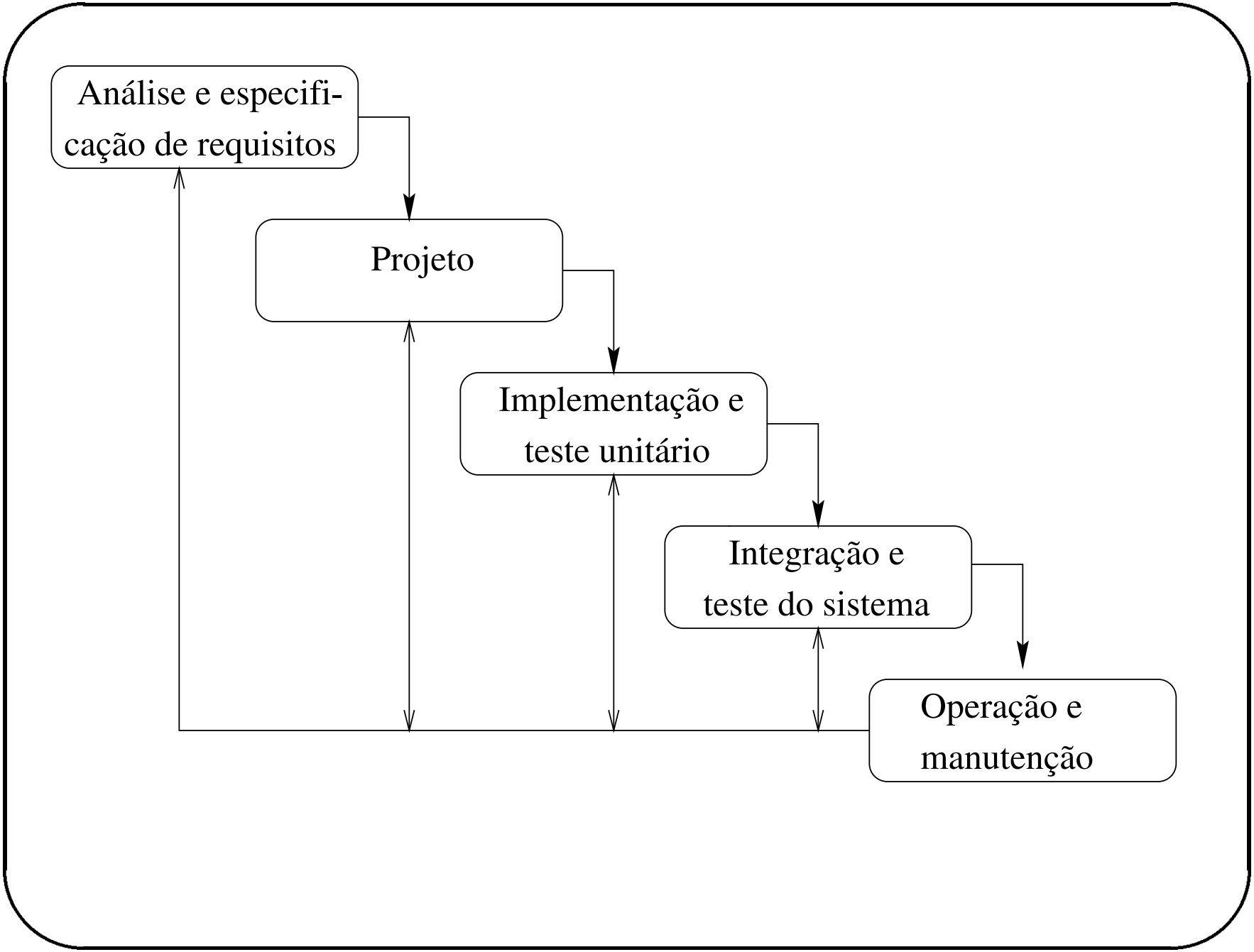
- Sistemas de controle de tráfego aéreo e ferroviário $\rightarrow$ confiabilidade.
- Controladores embutidos em aparelhos eletrodomésticos, como lavadoras de roupa e videocassetes $\rightarrow$ desempenho.
- Três dos modelos mais utilizados são:
  - **ciclo de vida clássico**
  - **evolutivo**
  - **espiral**

Ciclo de vida clássico

- método sistemático e seqüencial.
- o resultado de uma fase se constitui na entrada de outra.
- também conhecido como *cascata*.
- cada fase é estruturada como um conjunto de atividades que podem ser executadas por pessoas diferentes, simultaneamente.

Atividades do ciclo de vida clássico

1. análise e especificação dos requisitos;
2. projeto do sistema;
3. implementação e teste unitário;
4. integração e teste do sistema; e
5. operação e manutenção.


Análise e especifi-
cação de requisitos
Projeto
Implementação e
teste unitário
Integração e
teste do sistema
Operação e
manutenção

1. Análise e especificação de requisitos

- são identificados:
  - os serviços e as metas a ser atingidas, assim como as restrições a ser respeitadas.
  - a qualidade desejada em termos de funcionalidade, desempenho, facilidade de uso, portabilidade etc.
  - *quais* os requisitos do produto de software, sem especificar *como* esses requisitos serão obtidos.

- resultado dessa fase $\rightarrow$ documento de especificação de requisitos, com dois objetivos:
  1. deve ser analisado e confirmado pelo usuário para verificar se ele satisfaz todas as suas expectativas;
  2. deve ser usado pelos desenvolvedores de software para obter um produto que satisfaça os requisitos.

Qualidades do documento

- deve ser inteligível, preciso, completo, consistente e sem ambigüidades.
- deve ser facilmente modificável.
- outro produto da fase de análise e especificação de requisitos $\longrightarrow$ *plano de projeto*, baseado nos requisitos do produto a ser desenvolvido.

Documento de especificação dos requisitos

- *funcionais*: o que o produto de software faz, usando notações informais, semiformais, formais ou uma combinação delas;
- *não funcionais*: confiabilidade (disponibilidade, integridade, segurança), acurácia dos resultados, desempenho, problemas de interface, restrições físicas e operacionais, questões de portabilidade etc.;
- *de desenvolvimento e manutenção*: procedimentos de controle de qualidade (teste, prioridades das funções desejadas, mudanças prováveis nos procedimentos de manutenção do sistema etc).

2. Projeto dos sistema

- É definida a solução do problema
  - Decomposição do produto (sub-sistemas, componentes etc.)
  - representação das funções do sistema em uma forma que possa ser transformada em programas.
- Definição de "como" o produto deve ser implementado
- Pode ser dividido em:
  - Projeto de alto nível (Projeto geral)
  - Projeto detalhado
- Resultado $\rightarrow$ *documento de especificação do projeto*

3. Implementação e teste unitário

- Implementação: o projeto é transformado em um programa, ou unidades de programa.
- Teste unitário → cada unidade satisfaz suas especificações (plano e casos de teste pré-estabelecidos)
- Resultado → coleção de programas implementados e testados.

4. Integração e teste do sistema

- programas ou unidades de programa são integrados e testados como um sistema
- integração incremental → programas ou unidades são integrados à medida em que forem sendo desenvolvidos
- Resultado → produto pronto para ser entregue ao cliente

Operação e manutenção

1. Operação
   - atividade que mantém o sistema funcionando
   - inicialmente operado por um grupo selecionado de usuários
   - testes possibilitam correções antes da entrega final

2. Manutenção

   - conjunto de atividades executadas depois que o produto é entregue aos usuários
   - Tipos de manutenção
     - Manutenção corretiva → correção de erros remanescentes
     - Manutenção adaptativa → adaptação do produto às mudanças
     - Manutenção evolutiva → alteração dos requisitos e manutenção da qualidade

Estudo de viabilidade

- analisar o problema, pelo menos em nível global.
- identificar soluções alternativas, seus custos e potenciais benefícios para o usuário.
- simulação do futuro processo de desenvolvimento
- resultado $\rightarrow$ documento com avaliação dos custos e benefícios contendo:
  - a definição do problema;
  - soluções alternativas, com os benefícios esperados;
  - fontes necessárias, custos e datas de entrega para cada solução proposta.

Documentação

- produto ou resultado de grande parte das fases;
- a mudança ou não de uma fase para outra vai depender dos documentos;
- normalmente os padrões organizacionais definem a forma em que eles devem ser entregues.

Verificação

- ocorre em duas fases específicas (teste unitário e teste do sistema).
- realizada em várias outras fases.
- processo de controle de qualidade através de revisões e inspeções.
- objetivo: monitorar a qualidade do produto durante o desenvolvimento e não após a implementação.

Gerenciamento

- controla o processo de desenvolvimento.
  1. *adaptação* do ciclo de vida ao processo
  2. *definição de políticas*: como produtos ou resultados intermediários vão ser armazenados, acessados e modificados, como versões diferentes do sistema são construídas, quais autorizações são necessárias para acessar os componentes do sistema.
  3. *recursos* que afetam o processo de produção de software, particularmente os *recursos humanos*.

Contribuições do paradigma clássico

- o processo de desenvolvimento de software deve ser sujeito à disciplina, planejamento e gerenciamento.
- implementação do produto deve ser postergada até que os objetivos tenham sido completamente entendidos.
- deve ser utilizado especialmente quando os requisitos estão bem claros no início do desenvolvimento.

Problemas do paradigma clássico

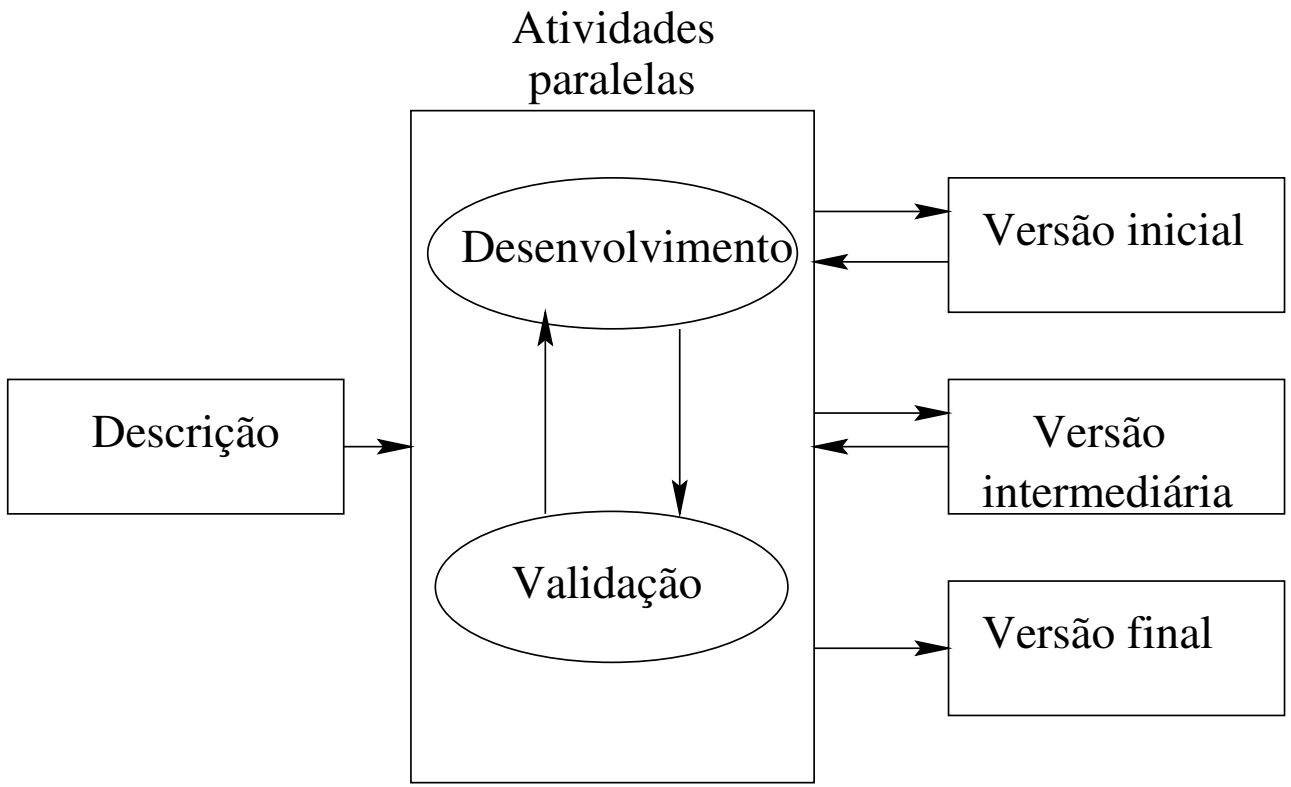
- rigidez, mas o desenvolvimento não é linear.
- a meta continua sendo tentar a linearidade, para manter o processo previsível e fácil de monitorar.
- qualquer desvio é desencorajado, pois vai representar um desvio do plano original e, portanto, requerer um replanejamento.
- todo o planejamento é orientado para a entrega do produto de software em uma data única.
- o processo de desenvolvimento pode ser longo e a aplicação pode ser entregue quando as necessidades do usuário já tiverem sido alteradas.

O paradigma evolutivo

- baseado no desenvolvimento e implementação de um produto inicial, que é submetido aos comentários e críticas do usuário.
- o produto vai sendo refinado através de múltiplas versões, até que o produto de software almejado tenha sido desenvolvido.
- as atividades de desenvolvimento e validação são desempenhadas paralelamente, com um rápido *feedback* entre elas.

1. Desenvolvimento exploratório

- o objetivo é trabalhar junto do usuário para descobrir seus requisitos, de maneira incremental, até que o produto final seja obtido.
- o desenvolvimento começa com as partes do produto que são mais bem entendidas.
- a evolução acontece quando novas características são adicionadas à medida que são sugeridas pelo usuário.
- importante quando é difícil, ou mesmo impossível, estabeler uma especificação detalhada dos requisitos do sistema *a priori*.


Atividades
paralelas
Desenvolvimento
Validação
Descrição
Versão inicial
Versão
intermediária
Versão final

Desenvolvimento exploratório (continuação)

- versão inicial do produto, que é submetida a uma avaliação inicial do usuário.
- essa versão é refinada, gerando várias versões, até que o produto almejado tenha sido desenvolvido.

2. Protótipo descartável

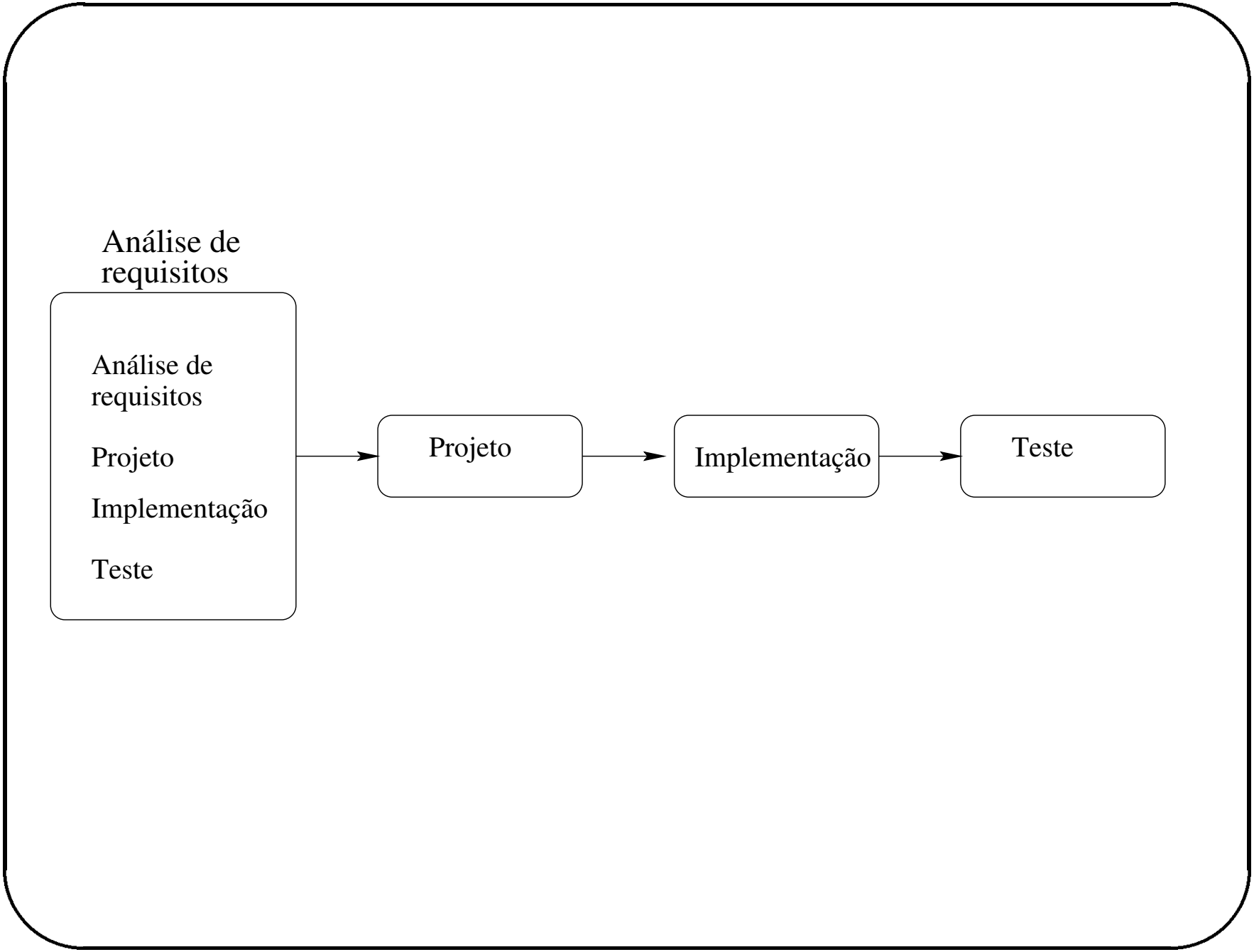
- entender os requisitos do usuário e obter uma melhor definição dos requisitos do sistema.
- usado para fazer experimentos com os requisitos do usuário que não estão bem entendidos.
- envolve projeto, implementação e teste, mas não de maneira formal ou completa.

Protótipo descartável (continuação)

- usuário define uma série de objetivos, mas não consegue identificar detalhes de entrada, processamento ou requisitos de saída.
- o desenvolvedor está incerto sobre a eficiência de um algoritmo, a adaptação de um sistema operacional ou ainda sobre a forma da interação homem–máquina.

Protótipo descartável (continuação)

- possibilita ao desenvolvedor criar um modelo do software que será construído.
- o desenvolvedor pode perceber as reações iniciais do usuário e obter sugestões para mudar ou inovar o protótipo.
- o usuário pode relacionar o que vê no protótipo diretamente com os seus requisitos.


Análise de requisitos
Análise de requisitos
Projeto
Implementação
Teste
Projeto
Implementação
Teste

Problemas do paradigma evolutivo

- o processo não é visível: como o desenvolvimento acontece de maneira rápida, não compensa produzir documentos que reflitam cada versão do produto de software.
- os sistemas são pobremente estruturados: mudanças constantes tendem a corromper a estrutura do software.
- o usuário vê o que parece ser uma versão em funcionamento do produto de software.
- o desenvolvedor, muitas vezes, assume certos compromissos de implementação.

Quando usar o desenvolvimento evolutivo

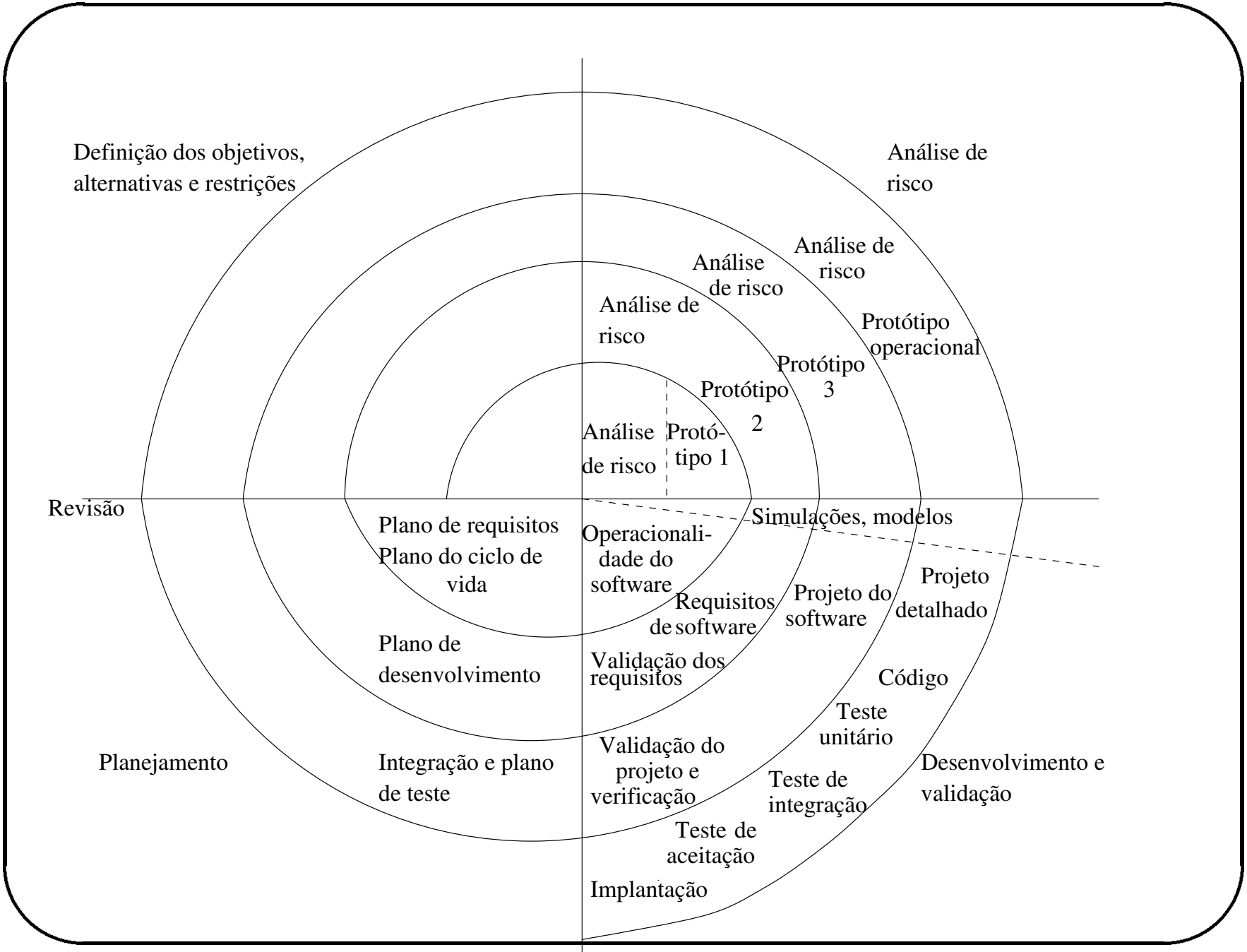
- sistemas pequenos, pois os problemas de mudanças no sistema atual podem ser contornados através da reimplementação do sistema todo quando mudanças substanciais se tornam necessárias.
- testes podem ser mais efetivos, visto que testar cada versão do sistema é provavelmente mais fácil do que testar o sistema todo no final.

O paradigma espiral

- desenvolvido para englobar as melhores características do ciclo de vida clássico e do paradigma evolutivo
- adiciona um novo elemento — a *análise de risco* — que não existe nos dois paradigmas anteriores.
- riscos: circunstâncias adversas que podem atrapalhar o processo de desenvolvimento e a qualidade do produto a ser desenvolvido.

O paradigma espiral (continuação)

- prevê a eliminação de problemas de alto risco através de um planejamento e projeto cuidadosos.
- as atividades podem ser organizadas como uma espiral que tem muitos ciclos.
- cada ciclo na espiral representa uma fase do processo de desenvolvimento do software.
- o primeiro ciclo pode estar relacionado com o estudo de viabilidade e com a operacionalidade do sistema; o próximo ciclo com a definição dos requisitos; o próximo com o projeto do sistema e assim por diante.
- não existem fases fixas.


Definição dos objetivos, alternativas e restrições
Análise de risco
Análise de risco
Análise de risco
Análise de risco
Protótipo operacional
Protótipo 3
Protótipo 2
Análise de risco
Protó-tipo 1
Revisão
Simulações, modelos
Plano de requisitos
Plano do ciclo de vida
Operacionali-dade do software
Requisitos de software
Projeto do software
Projeto detalhado
Plano de desenvolvimento
Validação dos requisitos
Código
Teste unitário
Validação do projeto e verificação
Teste de integração
Planejamento
Integração e plano de teste
Desenvolvimento e validação
Teste de aceitação
Implantação

Atividades do paradigma espiral

1. *Definição dos objetivos, alternativas e restrições*:
   - os objetivos para a fase de desenvolvimento são definidos, tais como desempenho e funcionalidade;
   - são determinadas alternativas para atingir esses objetivos;
   - as restrições relativas ao processo e ao produto são também determinadas;
   - um plano inicial de desenvolvimento é esboçado e os riscos de projeto são identificados;
   - estratégias alternativas, dependendo dos riscos detectados, podem ser planejadas.

Atividades do paradigma espiral (cont.)

2. *Análise de risco*: para cada um dos riscos identificados é feita uma análise cuidadosa; atitudes são tomadas visando à redução desses riscos.

3. *Desenvolvimento e validação*: após a avaliação dos riscos, um paradigma de desenvolvimento é escolhido.

4. *Planejamento*: o projeto é revisado, e a decisão de percorrer ou não mais um ciclo na espiral é tomada. Se a decisão for percorrer mais um ciclo, então o próximo passo do desenvolvimento do projeto deve ser planejado.

Atividades do paradigma espiral (cont.)

- à medida que a volta na espiral acontece (começando de dentro para fora), versões mais completas do software vão sendo progressivamente construídas.
- durante a primeira volta, são definidos objetivos, alternativas e restrições.
- se a análise de risco indicar que há incerteza nos requisitos, a prototipagem pode ser usada para auxiliar o desenvolvedor e o usuário.

Atividades do paradigma espiral (cont.)

- simulações e outros modelos podem ser usados para melhor definir o problema e refinar os requisitos.
- o processo de desenvolvimento tem início, e o usuário avalia o produto e faz sugestões de modificações.
- com base nos comentários do usuário, ocorre então a próxima fase de planejamento e análise de risco. Cada volta na espiral resulta em uma decisão "continue" ou "não continue".

Atividades do paradigma espiral (cont.)

- se os riscos forem muito grandes, o projeto pode ser descontinuado.
- cada passo leva os desenvolvedores em direção a um modelo mais completo do sistema e, em última instância, ao próprio sistema.

Atividades do paradigma espiral (cont.)

- a diferença mais importante entre o paradigma espiral e os outros paradigmas é a análise de risco.
- o paradigma espiral possibilita ao desenvolvedor entender e reagir aos riscos em cada ciclo.
- usa prototipagem como um mecanismo de redução de risco e mantém o desenvolvimento sistemático sugerido pelo ciclo de vida clássico.

Engenharia de software influenciando e sendo influenciada por outras áreas

- teoria da computação
- semântica formal
- linguagens de programação
- compiladores
- sistemas operacionais
- bancos de dados
- sistemas multiagentes
- sistemas especialistas

Outras áreas

- ciência do gerenciamento: estimativas de projeto, cronograma, planejamento dos recursos humanos, decomposição e atribuição de tarefas e acompanhamento do projeto, questões pessoais envolvendo contratação e atribuição da tarefa certa à pessoa certa,
- engenharia de sistemas: estuda sistemas complexos, partindo da hipótese de que certas leis governam o comportamento de qualquer sistema complexo, que por sua vez é composto de muitos componentes com relações complexas.

Comentários finais

- em muitos casos os paradigmas podem ser combinados de forma que os "pontos fortes" de cada um possam ser utilizados em um único projeto.
- o paradigma espiral já faz isso diretamente, combinando prototipagem e elementos do ciclo de vida clássico em um paradigma evolutivo.
- qualquer um desses pode servir como alicerce no qual outros paradigmas podem ser integrados.