

# **Introdução à Engenharia de Software**

Ariadne Carvalho

modificado por C.M.F.R em agosto/2010

## CONCEITOS BÁSICOS

### **Sistema:**

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

### **Elementos do sistema**

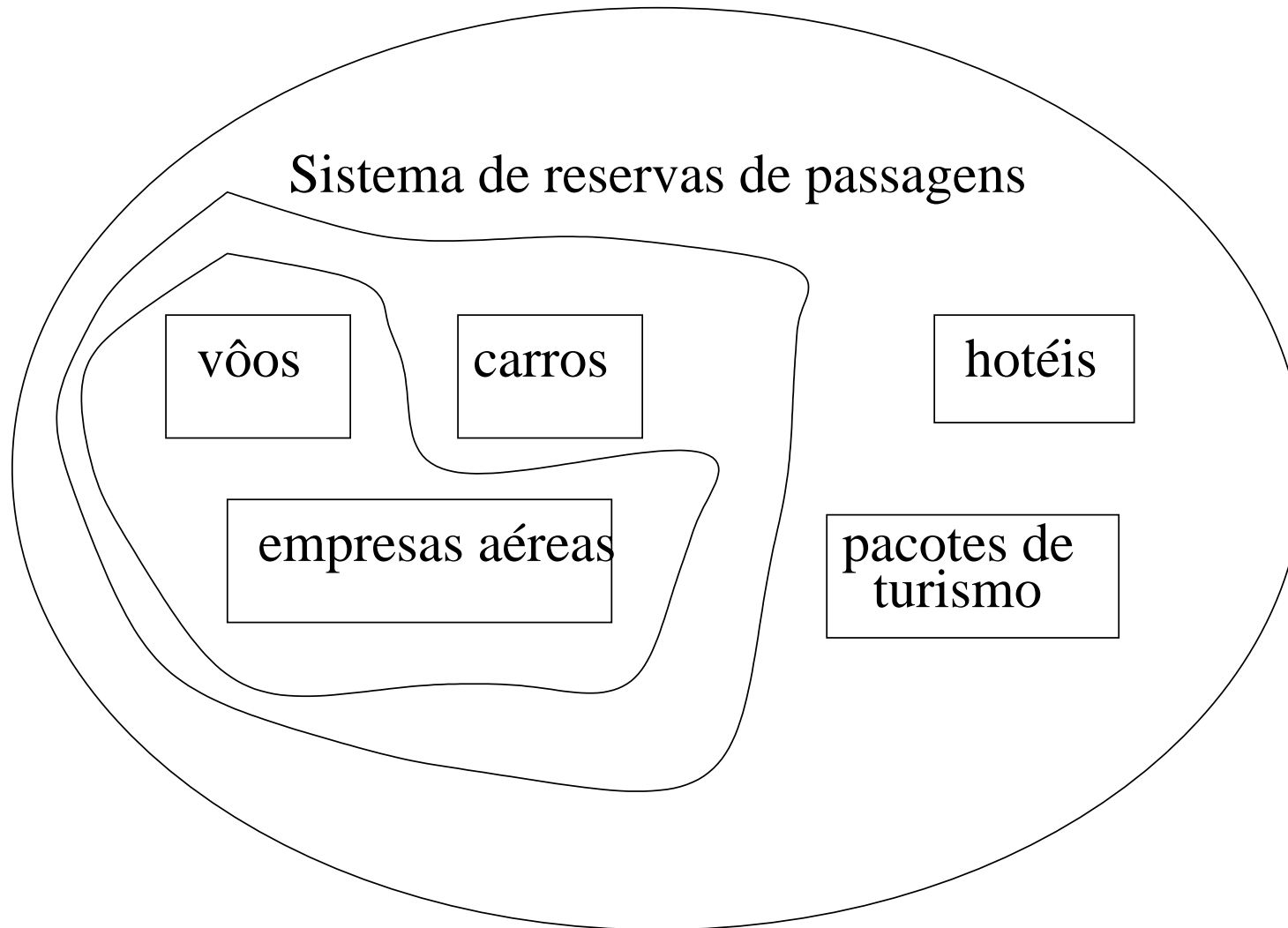
- Estão dentro da fronteira conceitual.
- Possuem interações fortes entre si.
- Possuem interações fracas com os elementos externos ao sistema.

## Fronteira conceitual

- Separa o que está dentro do sistema do resto.
- Elementos pertencentes ao sistema devem ser detalhados.
- Elementos do ambiente externo → 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 de informação, Sistemas de controle, Sistemas embarcados (embutidos) etc

## **Ciclo de vida do produto de software**

- começa na concepção do problema (solicitação do cliente).
- 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 sistema é produzido.
  - **validação**: o sistema é validado.
  - **manutenção**: o sistema sofre correções, adaptações e extensões.



## Engenharia de software

- Uma disciplina que reúne
  - metodologias,
  - métodos,
  - notações e ferramentas
- Metodologias → seguem métodos → que utilizam notações e ferramentas.
- Visam resolver problemas inerentes:
  - ao processo e
  - ao produto de sistema de software.

## Verificação & Validação (V&V)

- It is the process of checking that a software system meets its specification and fulfils its intended purpose.
- Validation checks that the product design satisfies the intended usage (high-level checking).
- Validation ensures that “you built the right thing”. Verification ensures that “you built it right”.
- Verification is the process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase[IEEE-STD-610].

## Conceito de Dependability

- Dependability is the property of a computing system which allows reliance to be justifiably placed on the service it delivers.
- The user is **depending on the provision of service**. For critical systems, it is usually the case that the most important system property is its dependability.
- The dependability of a system reflects the users degree of trust in that system and the extent of the users confidence that the system will operate as expected and that it will not fail in normal use.

## Dimensões de Dependability

- Availability (disponibilidade): The ability of the system to deliver services when requested.
- Reliability (confiabilidade): The ability of the system to deliver services as specified.
- Safety (segurança no funcionamento): The ability of the system to operate without catastrophic failure.
- Security (segurança no uso): The ability of the system to protect itself against accidental or deliberate intrusion (attack).

## Dimensões de Security

- Data Confidentiality is the ability of the system to keep information secret except for authorised users.
- Data Integrity is the ability of the system to guarantee that information has not been modified without permission, which includes the ability to detect unauthorised manipulation.
- Sender Authentication corresponds to the assurance, by communicating parties, of the origin of information transmitted through an insecure communication channel.
- Sender Nonrepudiation is the ability of the system to

prevent an entity from denying its actions or commitments in the future.

## Terminologia de Dependability

- Human Mistake  $\rightarrow$  System Fault  $\rightarrow$  System Error  $\rightarrow$  System Failure.
- System Failure (defeito): "An event that occurs at some point in time when the system does not deliver a service as expected by its user". Ex: Existe um defeito num caixa eletrônico quando um cliente do banco não consegue efetuar um saque nas condições de seu contrato.
- System Error (erro): "Erroneous system behaviour where the behaviour of the system does not conform to its specification".

- Como o comportamento de cada componente de um sistema é função do seu estado interno e das entradas que recebe, a ocorrência de um defeito (failure) pode estar associada a uma parte do seu estado que, sob determinadas entradas, provoca o defeito.
- A parte errônea do estado interno de um componente é chamada de erro. Ex: O saque solicitado pelo cliente pode ser negado em função de um valor incorreto do saldo da sua conta.
- Um erro tem natureza estática e é interno ao componente ou sistema, podendo ou não vir a provocar defeito.



- System Fault (falha): "An incorrect system state i.e. a system state that is unexpected by the designers of the system". A causa de um erro é chamada de falha. Ex: Um valor incorreto para o saldo da conta poderia ter sido causado por uma falha na transmissão dos dados entre o computador do banco e o caixa eletrônico.
- Uma falha é um evento que propicia o surgimento de um erro num componente ou no sistema como um todo.
- Human Mistake: Human behaviour that results in the introduction of faults into a system.

- Fault Tolerance: é a capacidade de um sistema preservar suas funcionalidades, sem apresentar defeitos, mesmo na presença de falhas.

## Princípios da engenharia de software

- Formalidade: produzir produtos mais “dependable”, controlar seu custo e aumentar sua confiança.
- 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: ser modificado com alguma 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 → produtos de alta qualidade, produzidos mais rapidamente e a um custo cada vez menor.
- atributos: complexidade, aceitabilidade, confiabilidade, manutenibilidade, segurança etc.
- possibilitam:
  1. ao gerente: controlar o processo de desenvolvimento de sistemas.
  2. ao desenvolvedor: obter a base para produzir, de maneira eficiente, sistemas que satisfaçam os requisitos pré-estabelecidos.

## Exemplos de Sistemas de Software

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

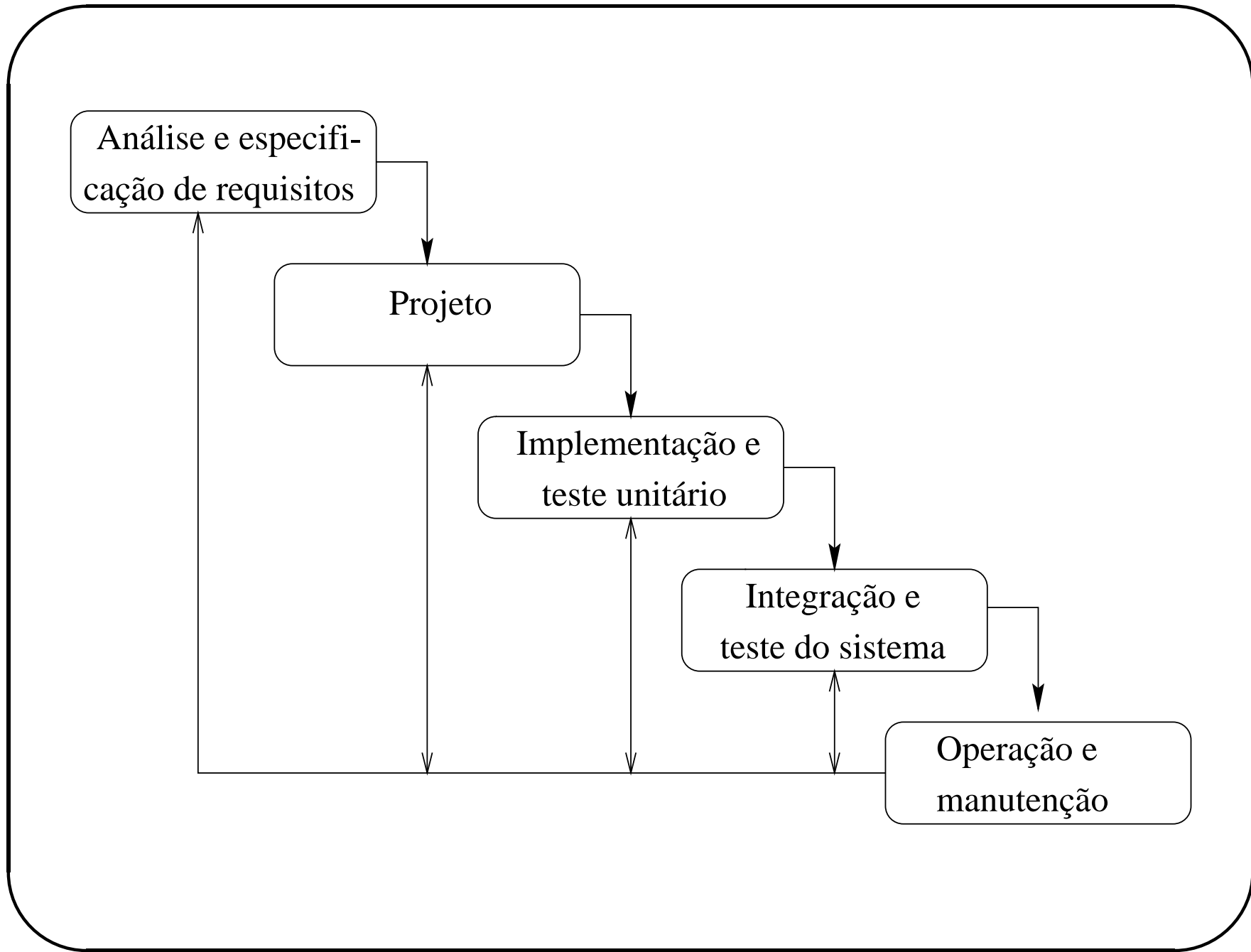
## Ciclo de vida clássico

- 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.





## 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 → 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 → **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, semi-formais, formais ou uma combinação delas
- **não-funcionais ou atributos de qualidade:** dependability, desempenho, problemas de interface, restrições físicas e operacionais, 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 do 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 Arquitetural) e
  - Projeto detalhado.
- Resultado → **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.



## 5. Operação e manutenção

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

- 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 → 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.

## Validação

- 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.

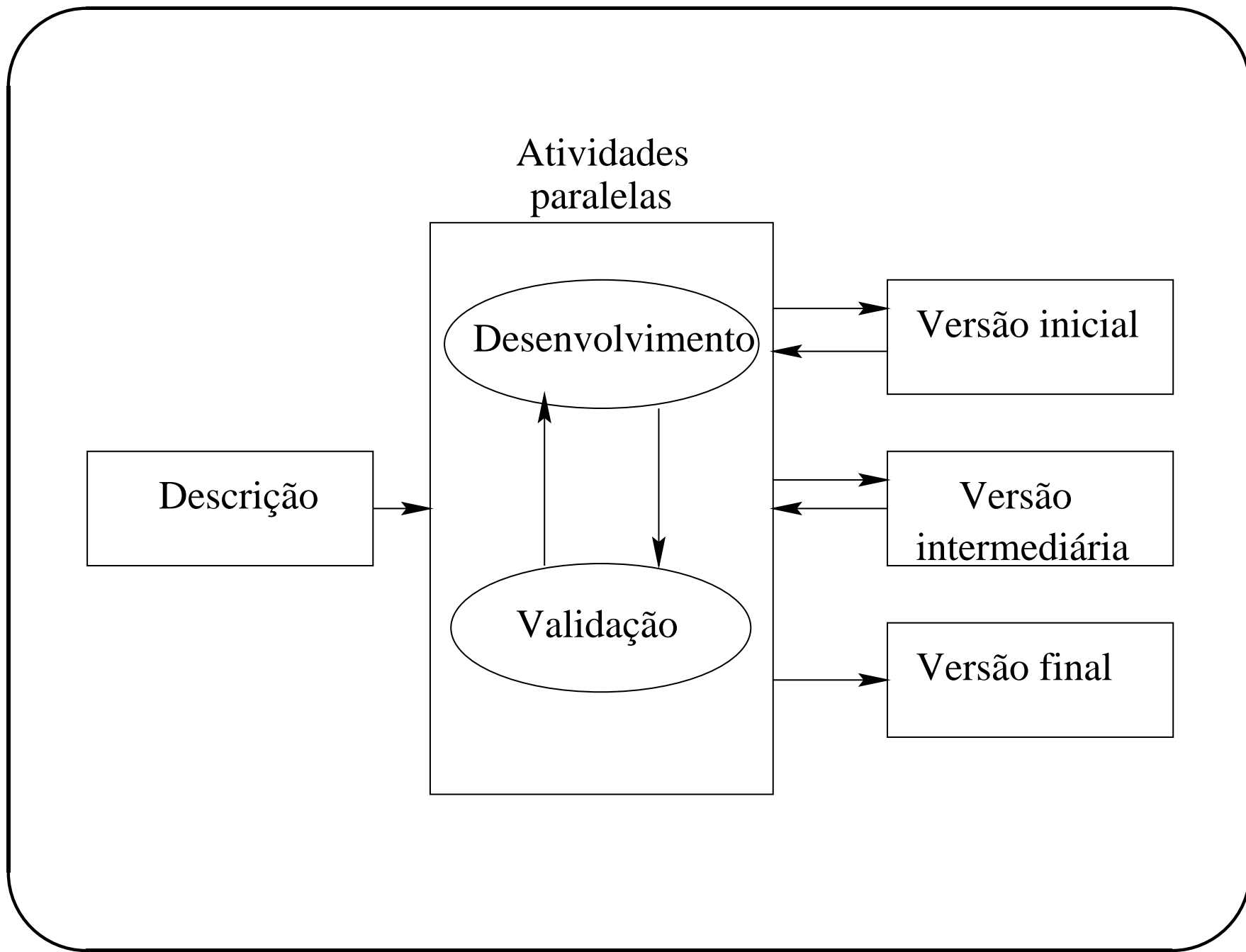


## O Paradigma Evolutivo

- 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, estabelecer uma especificação detalhada dos requisitos do sistema *a priori*.



## 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.

## Rapid software development

- Because of the changing environment, it is often impossible to arrive at a stable, consistent set of system requirements.
- Therefore a waterfall model of development is impractical and an approach to development based on iterative specification and delivery is the only way to deliver software quickly.
- The processes of specification, design and implementation are concurrent. There is no detailed specification and design documentation is minimised.
- The system is developed in a series of increments.

- End users evaluate each increment and make proposals for later increments.
- Advantages of incremental development:
  - Accelerated delivery of customer services. Each increment delivers the highest priority functionality to the customer.
  - User engagement with the system: Users have to be involved in the development which means the system is more likely to meet their requirements and the users are more committed to the system

## Agile methods

- Dissatisfaction with the overheads involved in design methods led to the creation of agile methods.
- Focus on the code rather than the design;
- Are based on an iterative approach to software development;
- Are intended to deliver working software quickly and evolve this quickly to meet changing requirements.
- are probably best suited to small/medium-sized business systems.

## Principles of agile methods

- **Customer involvement:** The customer should be closely involved throughout the development process. His/her role is provide and prioritise new system requirements and to evaluate the iterations of the system.
- **Incremental delivery:** The software is developed in increments with the customer specifying the requirements to be included in each increment.
- **People not process:** The skills of the development team should be recognised and exploited. The team should be left to develop their own ways of working

without prescriptive processes.

- **Embrace change:** Expect the system requirements to change and design the system so that it can accommodate these changes.
- **Maintain simplicity:** Focus on simplicity in both the software being developed and in the development process used. Wherever possible, actively work to eliminate complexity from the system.

## Problems with Agile Methods

- It can be difficult to keep the interest of customers who are involved in the process.
- Team members may be unsuited to the intense involvement that characterises agile methods.
- Prioritising changes can be difficult where there are multiple stakeholders.
- Maintaining simplicity requires extra work.
- Contracts may be a problem as with other approaches to iterative development.
- Examples: Extreme Programming - XP (Kent Beck, 1999), Scrum (Schwaber and



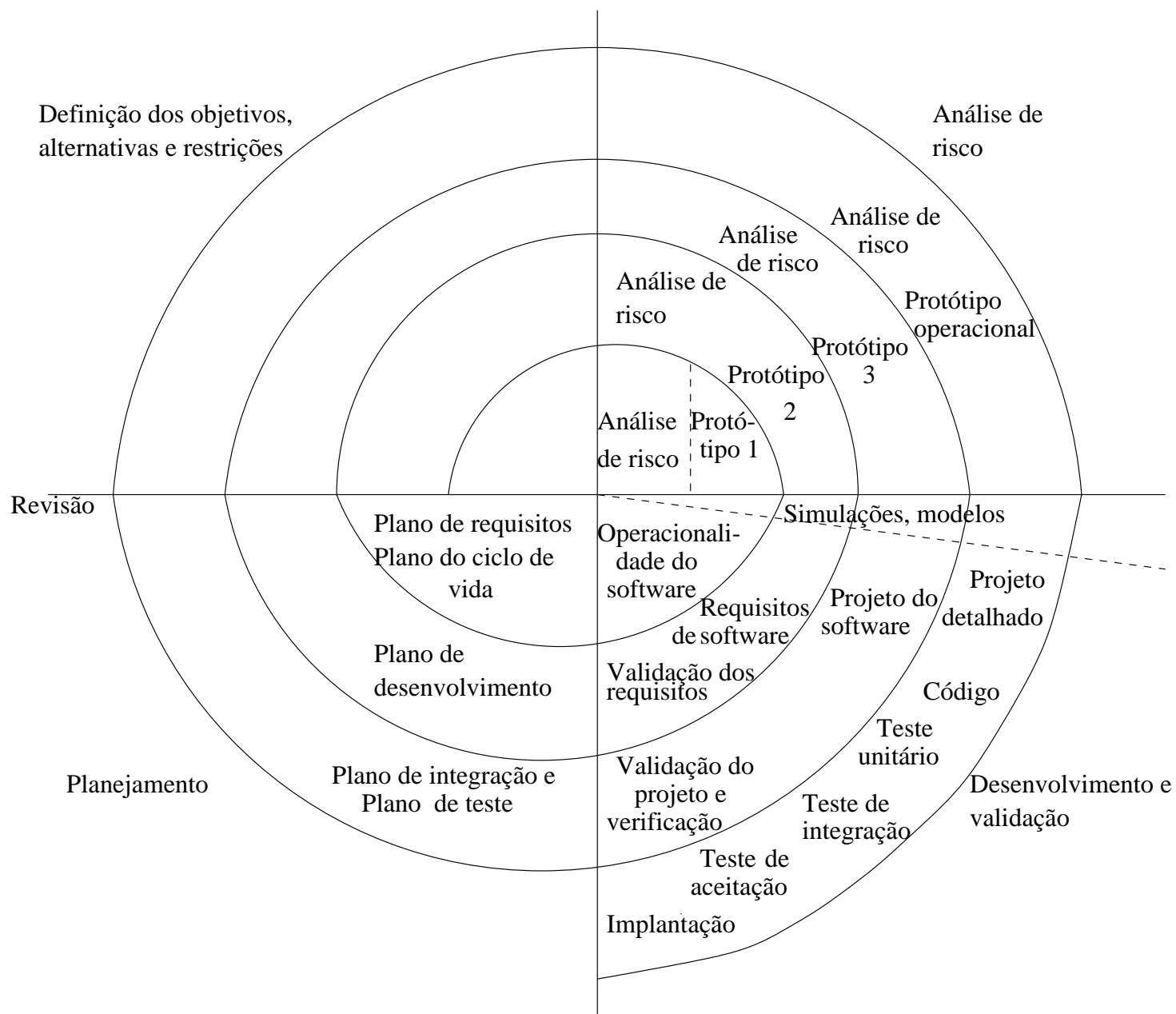
Beedle,2001),Crystal (Cockburn,2001), Adaptive  
Software Development, Feature Driven Development  
etc

## 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.



## 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.