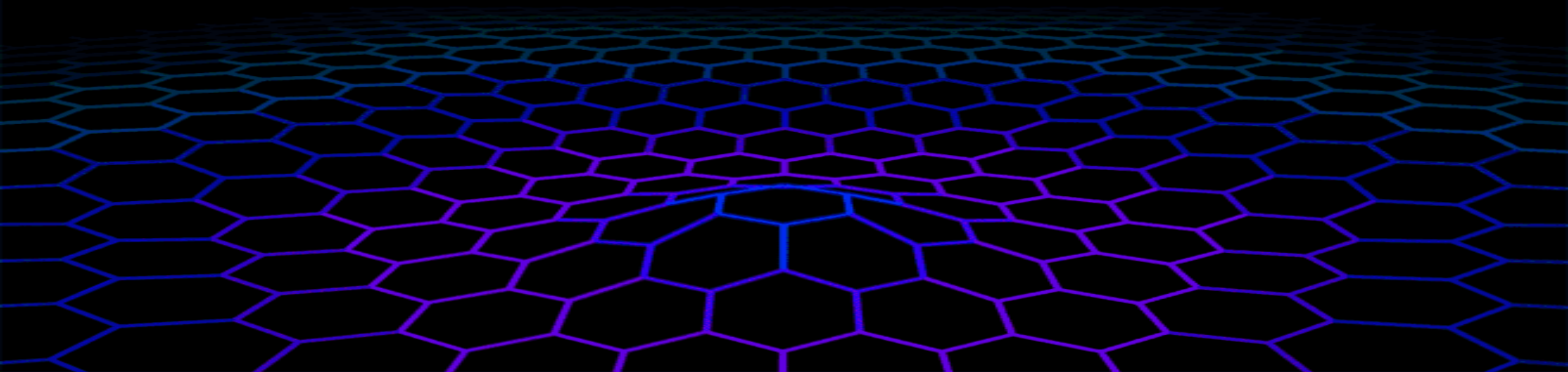
Faculdades senac

Engenharia de Software 1



Metodologias de desenvolvimento de software

Qual objetivo?

As metodologias de desenvolvimento de software servem para não tornar a tarefa, complexa por natureza, um verdadeiro caos;

Dependendo do projeto, os métodos tradicionais podem deixar os desenvolvedores “amarrados” a requisitos desatualizados, que não correspondem às reais necessidades do cliente.

Quantas metodologias existem?

Muitas!

Existem metodologias boas criadas a muito tempo;

Existem metodologias que não são mais usadas;

Existem metodologias novas para desenvolvimento;

Qual a ideia chave?

Muitas metodologias foram criadas ao longo dos anos e seguem sendo usadas até hoje.

A ideia é ver as metodologias mais populares para entender as diferenças entre elas e, principalmente, o que elas têm em comum.

No fim do semestre veremos também o que são metodologias ágeis.

Quais metodologias vamos estudar?

Codifica-Corrige

Modelo em Cascata (ciclo de vida clássico)

Prototipação

Modelo Espiral

Modelo por Incremento

RUP (Rational Unified Process)

Codifica-Corrige

Codifica-Corrige

Ela não é bem uma metodologia, mas é muito usada por desenvolvedores até hoje;

Nessa metodologia o desenvolvimento possui ciclos rápidos de codificação seguidos por correção;

Funcionamento

Inicialmente, o cliente pode fornecer uma especificação do que ele precisa.

Essa especificação pode ser obtida através de algumas anotações, e-mail, ou de qualquer outra fonte não muito consistente.

Desvantagens

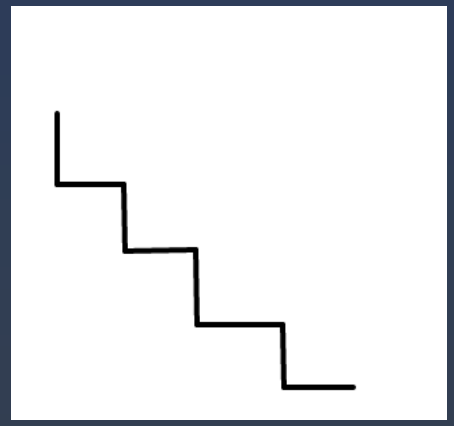
A qualidade do produto é baixa;

O sistema frequentemente se transforma num código bagunçado, com falta de adaptabilidade e reuso;

Os sistemas são difíceis de serem mantidos e aprimorados.

Os sistemas frequentemente tornam-se complicados e com baixa escalabilidade.

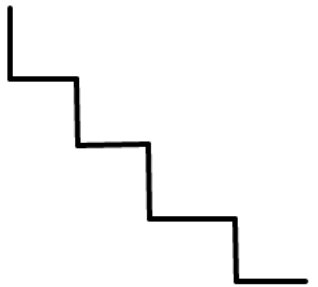
Modelo em Cascata



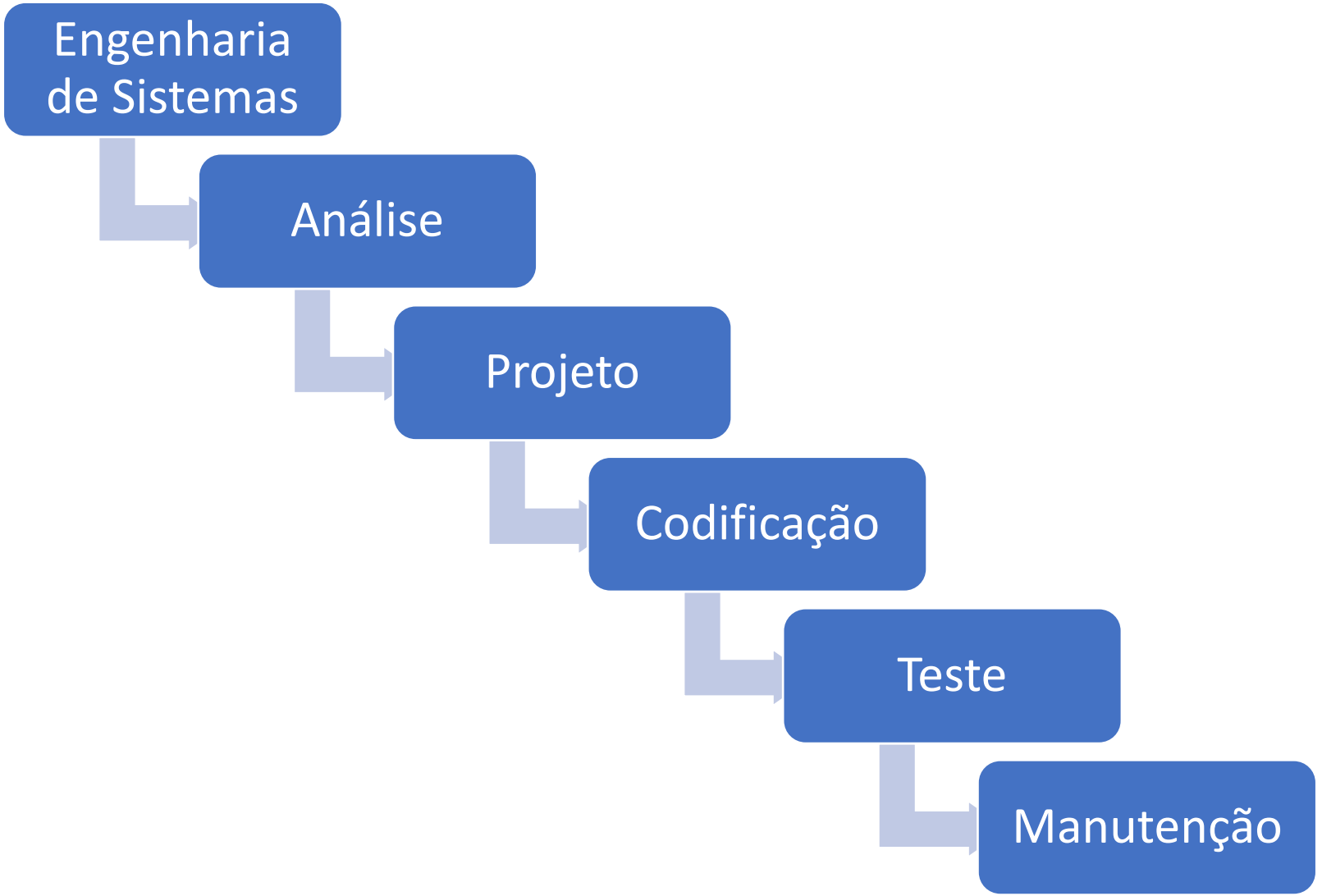
Modelo em Cascata

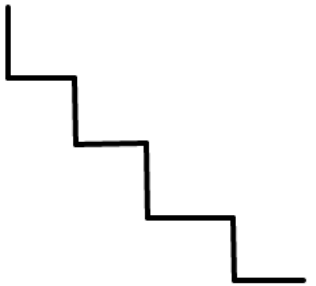
A metodologia de desenvolvimento em cascata foi desenvolvida pela marinha norte-americana nos anos 60 para permitir o desenvolvimento de softwares militares complexos.

A ideia central é que somente após um estado ser finalizado pode-se iniciar outro.



Funcionamento



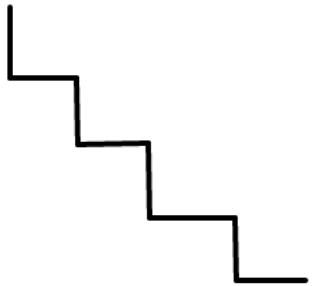


A metodologia

A Metodologia em Cascata é a metodologia mais antiga e mais amplamente usada no desenvolvimento de software;

Ela funciona bem quando os requisitos dos usuários são rígidos e podem ser conhecidos com antecedência;

Mesmo assim, alguns problemas podem surgir.



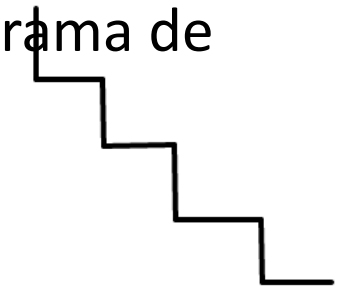
Desvantagens

Os projetos reais raramente seguem o fluxo sequencial que o modelo propõe;

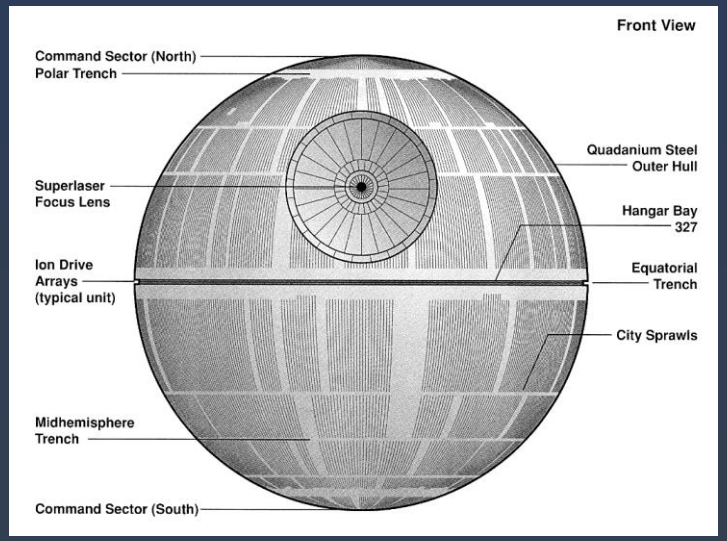
Muitas vezes é difícil para o cliente declarar todas as exigências explicitamente;

~~Os sistemas são difíceis de serem mantidos e aprimorados;~~

O cliente deve ter paciência, pois qualquer erro detectado após a revisão do programa de trabalho pode ser desastroso.



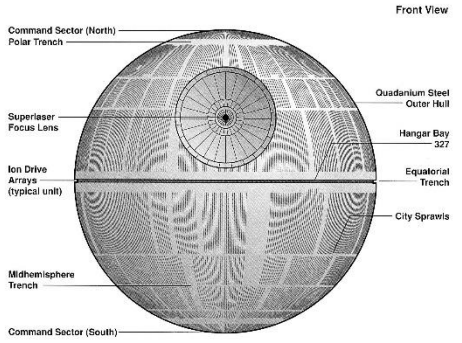
Prototipação



Prototipação

A prototipação capacita o desenvolvedor a criar um modelo de software que será implementado;

O objetivo é antecipar ao usuário final um modelo de sistema para que ele possa avaliar sua finalidade, identificar erros e omissões, quando em utilização, efetuando de imediato correções e ajustes;

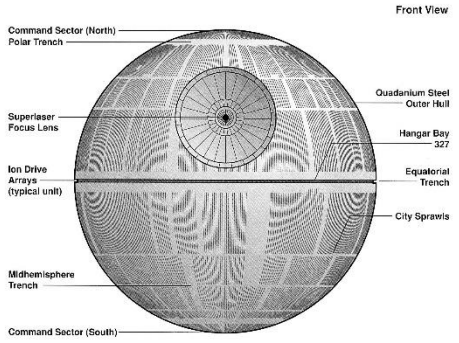


Tipos de protótipos

Um protótipo em papel ou no computador, que retrata a interação homem máquina;

Um protótipo que implemente funções específicas;

Um programa existente que executa parte ou toda a função desejada, mas que tem características que deverão ser melhoradas;



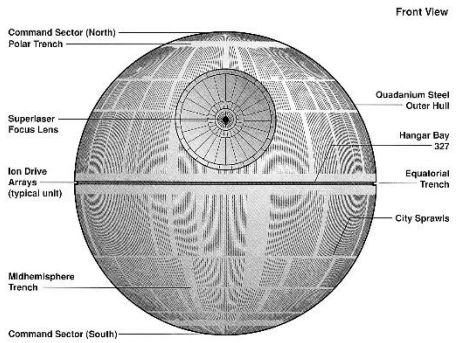
Funcionamento

Engenharia do Produto

Coleta e refinamento de requisitos

Projeto Rápido

Construção do Protótipo



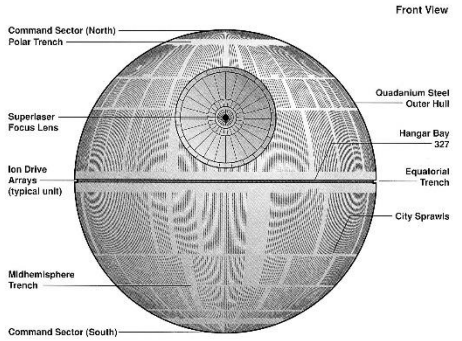
Avaliação do Protótipo pelo Cliente

Refinamento do Produto

Desvantagens

O cliente quer resultados, e, muitas vezes não saberá, ou não entenderá, que um protótipo pode estar longe do software ideal, que ele nem sequer imagina como é;

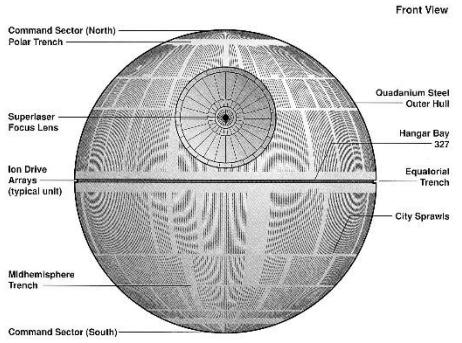
O desenvolvedor, na pressa de colocar um protótipo em funcionamento, é levado a usar uma linguagem de programação imprópria por simplesmente estar à disposição ou estar mais familiarizado. Essa atitude poderá levar a um algoritmo ineficiente;



Prototipoação

O protótipo é baseado no feedback dos usuários, devido a isso mudanças são feitas no protótipo;

Ainda que possam ocorrer problemas, o uso da prototipação é um paradigma eficiente, onde o “segredo” é o entendimento entre o desenvolvedor e o cliente.



Modelo Espiral

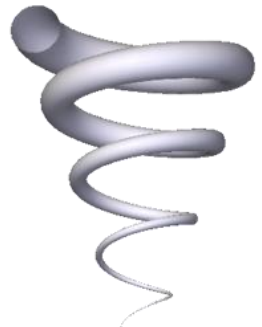
Espiral

Foi desenvolvido para abranger as melhores características do ciclo de vida clássico e da prototipação, ao mesmo tempo que adiciona um novo elemento, que é a análise de risco.

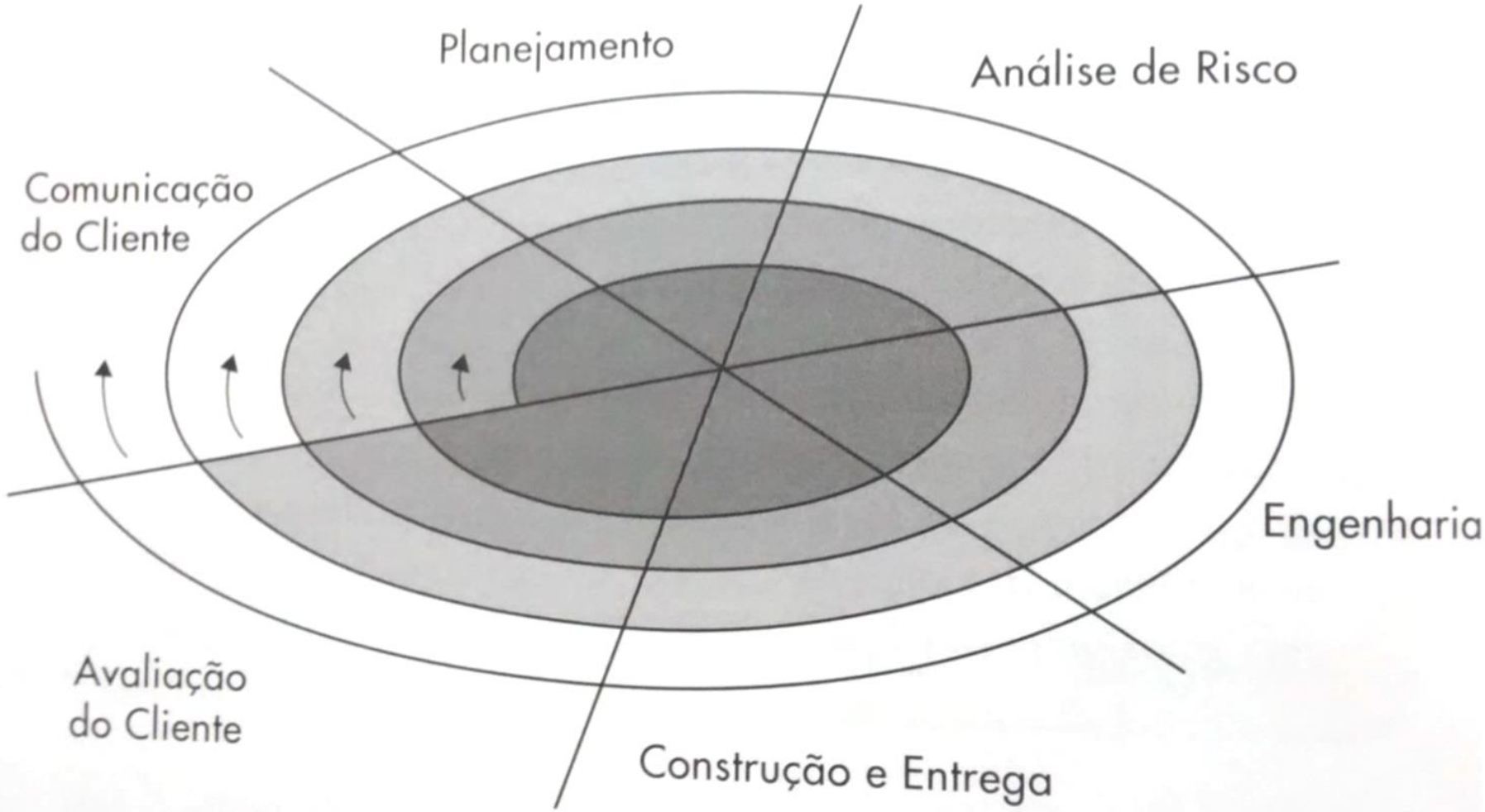
Este modelo de ciclo de vida se utiliza de protótipos por se adequar muito bem com esta filosofia de desenvolvimento;

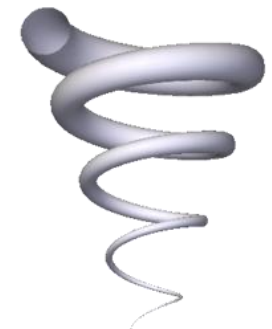
Cada passo através do ciclo inclui: planejamento, análise de risco, engenharia, construção e avaliação.

Os passos vão sendo repetidos até que um produto seja obtido.



Funcionamento





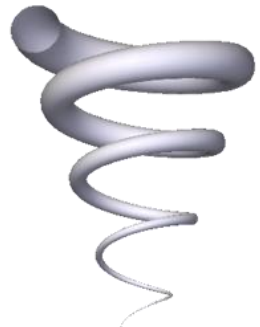
Alguns problemas

O problema a ser resolvido não está totalmente entendido;

A realidade pode mudar enquanto o sistema está sendo desenvolvido;

Pessoas que abandonam a equipe de desenvolvimento;

Ferramentas que não podem ser utilizadas ou falha em equipamentos.



Modelo V

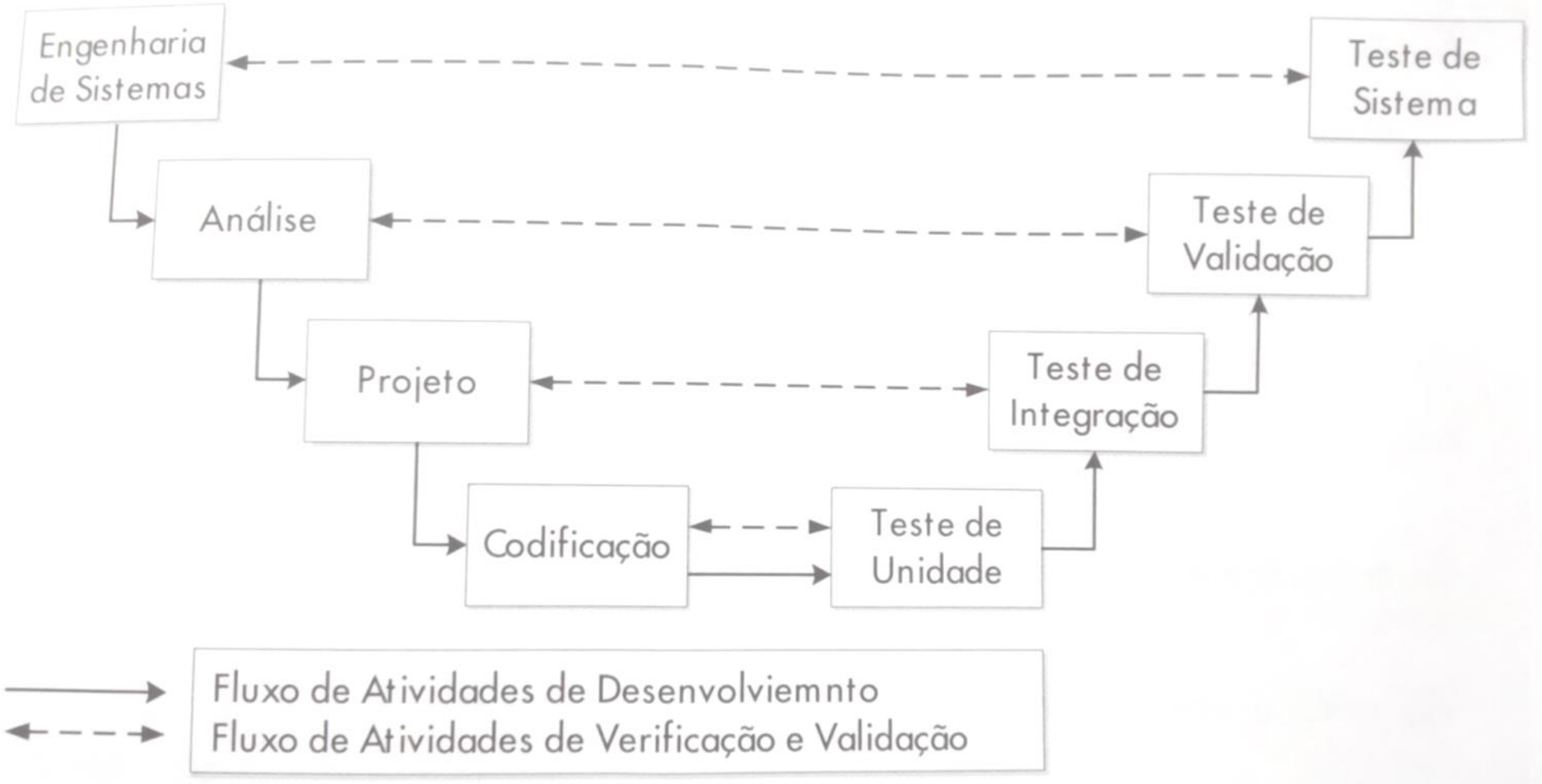
Modelo V

É um processo para desenvolvimento de sistemas, inspirado no modelo Cascata;

É usado como padrão em projetos de software na Alemanha;

O lado esquerdo dá ênfase à decomposição dos requisitos e o lado direito à integração do sistema;

Modelo V


Engenharia de Sistemas
Análise
Projeto
Codificação
Teste de Unidade
Teste de Integração
Teste de Validação
Teste de Sistema
Fluxo de Atividades de Desenvolviemnto
Fluxo de Atividades de Verificação e Validação

Principais características

Divisão entre atividades de desenvolvimento e de verificação e validação;

Clareza nos objetivos de verificação e validação;

Melhor planejamento de testes.

Modelo por Incremento

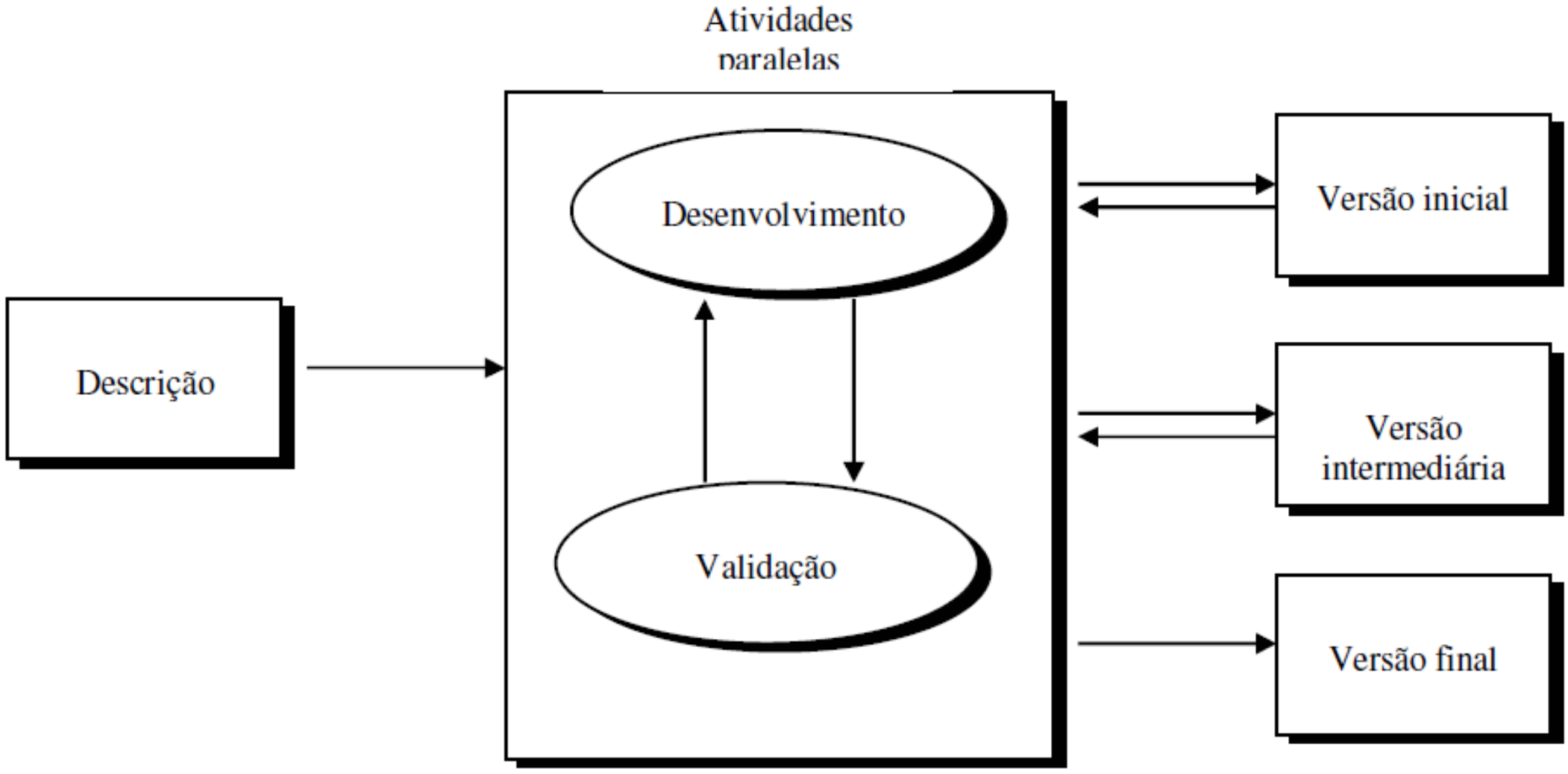
Modelo por incremento

Nos modelos por incremento, também chamado de modelo evolutivo, um único conjunto de componentes é desenvolvido simultaneamente;

Num primeiro tempo o núcleo do software é desenvolvido;

Depois os incrementos vão sendo agregados.

Funcionamento



Vantagens

Cada desenvolvimento é menos complexo;

As integrações são progressivas;

A cada interação, uma nova versão do produto pode ser distribuída;

Desvantagens

Problemas no núcleo do software, inviabilizando novos incrementos, e determinando o fim do ciclo de vida do software;

Incapacidade de integrar novos incrementos.

Desvantagens

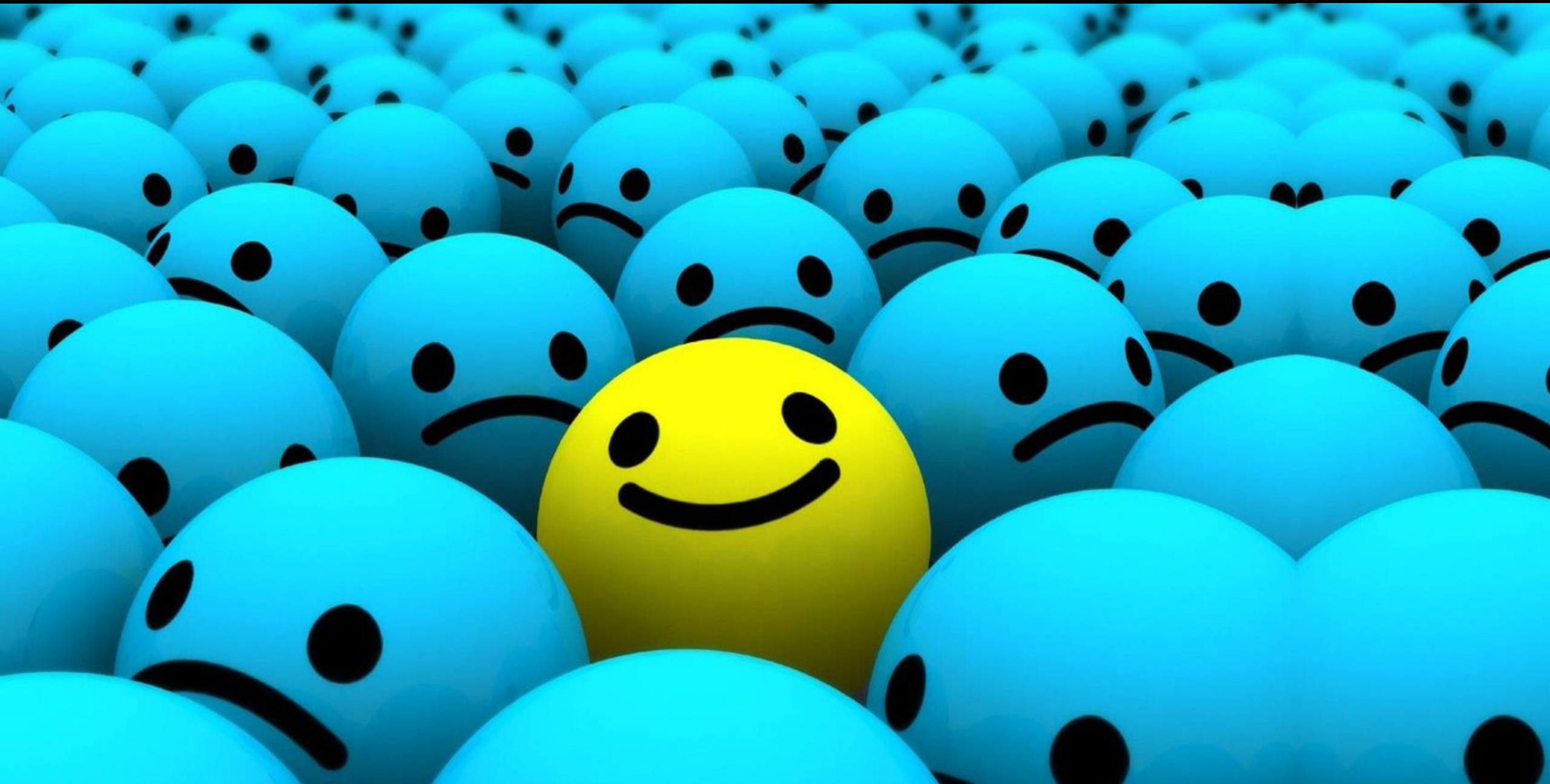
Nesse modelo de desenvolvimento incremental, temos mais uma metodologia:

- RAD (Rapid Application Development);

Fora isso, também temos algumas metodologias “pesadas” e interessantes como:

- UP (Unified Process)
  - RUP (Rational UP) criada pela IBM inspirada na UP;
  - Agile Unified Process também inspirada da UP.

Outras metodologias



Análise de Requisitos

Introdução

A **análise** e **especificação** de requisitos de software envolvem as atividades de determinar os **objetivos** de um software e as **restrições** associadas a ele

Determinar de forma clara quais as principais necessidades do software

Análise de requisitos

É o processo de **observação** e **levantamento** dos elementos do domínio no qual o sistema será introduzido.

Devemos identificar:

- As pessoas envolvidas;
- As atividades realizadas;
- As informações necessárias para a atividade;

Entender como o processo funciona

Especificação de requisitos

Descrição sistemática e abstrata do que o software deve fazer, a partir daquilo que foi analisado

Ela apresenta a **solução** de como os problemas levantados na análise serão resolvidos pelo software a ser desenvolvido

Especificação de requisitos

Descrição sistemática e abstrata do que o software deve fazer, a partir daquilo que foi analisado

Ela apresenta a **solução** de como os problemas levantados na análise serão resolvidos pelo software a ser desenvolvido

O que são requisitos?

Requisitos

Requisitos são **objetivos** ou **restrições** estabelecidas por clientes e usuários do sistema que definem as diversas propriedades do software

Tradicionalmente, os requisitos de software são separados em:

- Requisitos funcionais
- Requisitos não-funcionais

Requisitos Funcionais

Descrição das diversas funções que clientes e usuários querem ou precisam que o software ofereça

Eles definem as funcionalidades que o software irá conter

Exemplos de RF - Requisitos Funcionais

RF 01 – Calcular gastos mensais com pessoal

RF 02 – Emitir relatórios de compras por um intervalo de data pré-determinado;

RF 03 – Cadastrar usuários do sistema

Requisitos NÃO Funcionais

São restrições na qual o sistema deve operar ou propriedades emergentes do sistema

Geralmente são divididos por aspectos, como:

- Manutenibilidade
- Usabilidade
- Desempenho
- Entre outros

São difíceis de validar, exemplo:

- Relação entre segurança/usabilidade

Exemplos de RNF – Requisitos Não Funcionais

RNF 01 – A senha dos usuários deve ser criptografada através de md5

RNF 02 – O carregamento das telas de listagem não pode ultrapassar 10 segundos

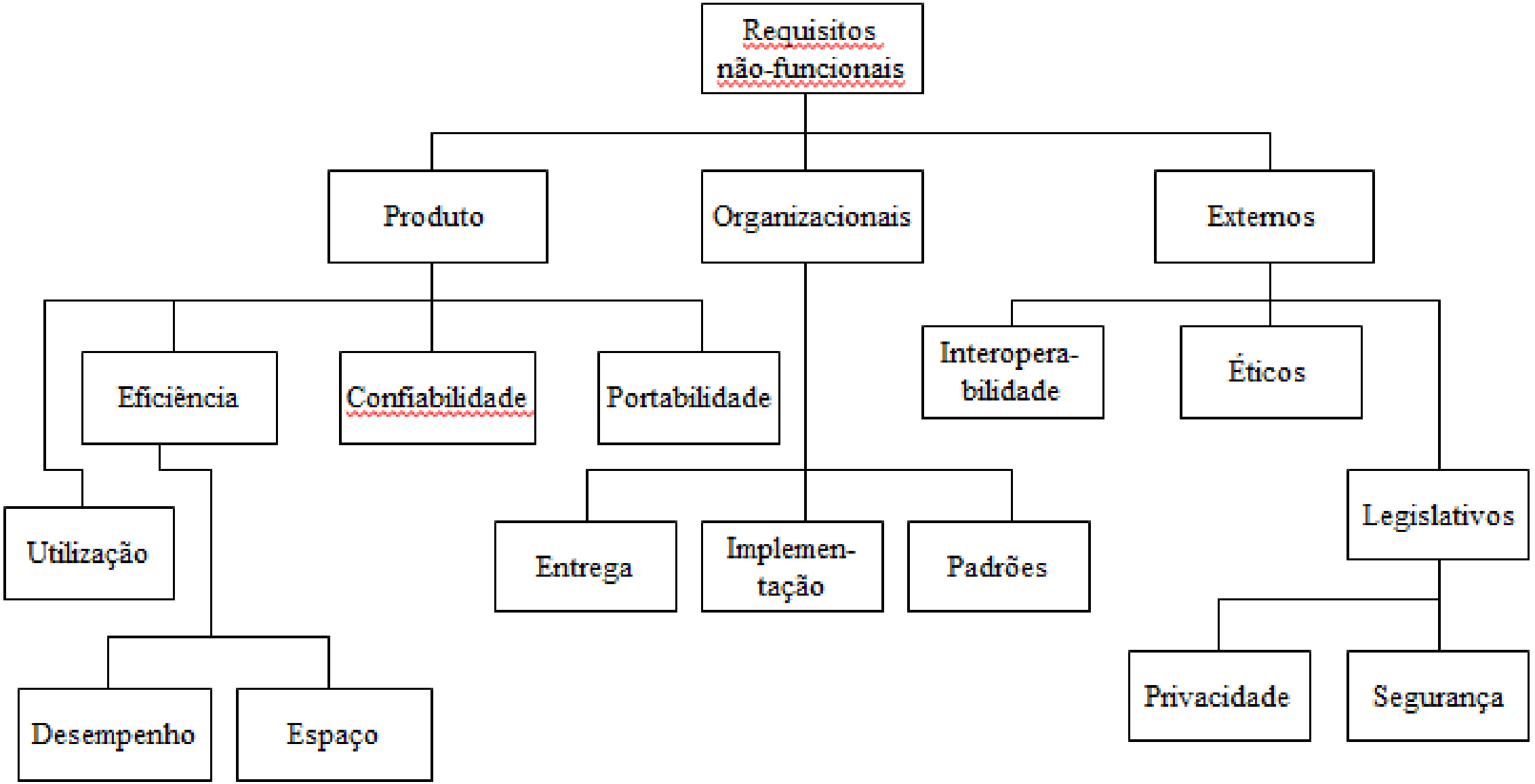
RNF 03 - O software deve rodar em sistemas operacionais Android

Observação

Os requisitos **não-funcionais** podem ser mais **críticos** do que os funcionais porque se não forem satisfeitos o sistema fica sem utilidade

Alguém pode dizer um exemplo disto?

Estrutura de classificação RNF



Outro detalhe importante

Sempre que possível, devemos escrever os requisitos não funcionais quantitativamente, de modo que eles possam ser testados objetivamente:

Propriedade	Medida
Velocidade	Transações processadas/segundo Tempo de resposta de usuário/evento Tempo de atualização da tela
Tamanho	Kbytes
Facilidade de Uso	Tempo de treinamento Número de frames de ajuda
Confiabilidade	Portabilidade de indisponibilidade Taxa de ocorrência de falhas Disponibilidade
Robustez	Tempo para reiniciar após falha Porcentagem de eventos que causam falhas Probabilidade de corrupção de dados por falhas

Pontos importantes

Não jogue esforço fora

Documentar requisitos custa caro, é importante gerarmos documentação que sempre será aproveitada

Não adianta gerar documentos com infinitas páginas de requisitos de um sistema que ninguém irá consultar

É importante ser objetivo e claro na execução dessa tarefa

Como ser produtivo?

O cliente entende o documento?

Os desenvolvedores entendem o documento?

Os demais participantes do projeto entendem o documento?

Como ser produtivo?

Veremos vários artefatos que poderemos usar para gerar nossos requisitos e demais documentações do sistema

O importante é sempre manter o foco no cliente e em suas necessidades

FIM