

Introdução a Engenharia de Software

Viviane Torres da Silva
viviane.silva@ic.uff.br

<http://www.ic.uff.br/~viviane.silva/2010.2/es1>

Histórico

- 1968: Crise do Software
 - Nasce a Engenharia de Software

- 1970s:
 - Lower-CASE tools (programação, depuração, colaboração)
 - Ciclo de vida cascata
 - Desenvolvimento estruturado

- 1980s:
 - Ciclo de vida espiral
 - Desenvolvimento orientado a objetos

- 1990s: Upper-CASE tools
 - Processos
 - Modelagem

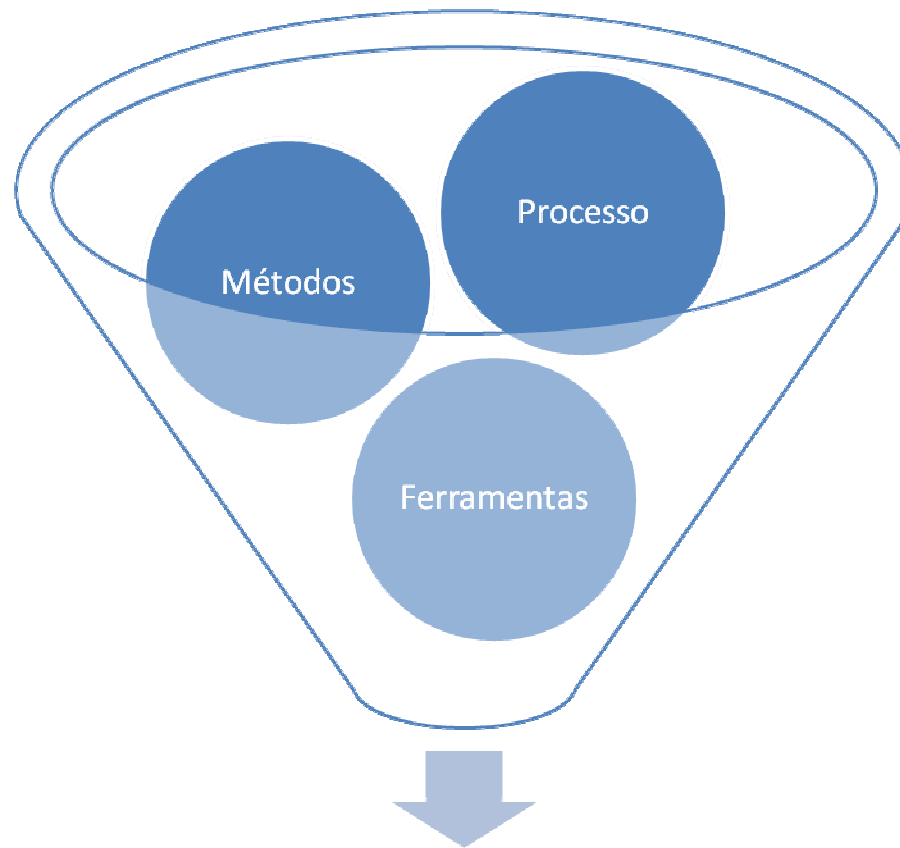
Atualmente

- Métodos ágeis
- Desenvolvimento dirigido por modelos
- Linhas de produto
- Experimentação
- Engenharia de Software para outros paradigmas
 - Agentes de Software

Elementos da ES

Método, Processo e Ferramenta

Elementos da Engenharia de Software



Engenharia de Software

Elementos da Engenharia de Software

➤ Processo

- Define os passos gerais para o desenvolvimento e manutenção do software
- Serve como uma estrutura de encadeamento de métodos e ferramentas

➤ Métodos

- Descrevem como fazer um passo específico do processo
- São os “how to’s”

➤ Ferramentas

- Automatizam / auxiliam o processo e os métodos

Elementos da Engenharia de Software

O que é processo, método ou ferramenta?

1. Coloque em uma panela funda o leite condensado, a margarina e o chocolate em pó.
2. Cozinhe [no fogão] em fogo médio e mexa sem parar com uma colher de pau.
3. Cozinhe até que o brigadeiro comece a desgrudar da panela.
4. Deixe esfriar bem, então unte as mãos com margarina, faça as bolinhas e envolva-as em chocolate granulado.

Elementos da Engenharia de Software

O que é processo, método ou ferramenta?

1. **Coloque** em uma **panela** funda o leite condensado, a margarina e o chocolate em pó.
2. **Cozinhe** [no **fogão**] em fogo médio e **mexa** sem parar com uma **colher de pau**.
3. **Cozinhe** até que o brigadeiro comece a desgrudar da **panela**.
4. Deixe esfriar bem, então **unte** as **mãos** com margarina, **faça as bolinhas** e **envolva**-as em chocolate granulado.

Processo



método



ferramenta

Elementos da Engenharia de Software

- Cuidado com o “desenvolvimento guiado por ferramentas”
 - É importante usar a ferramenta certa para o problema
 - O problema não deve ser adaptado para a ferramenta disponível

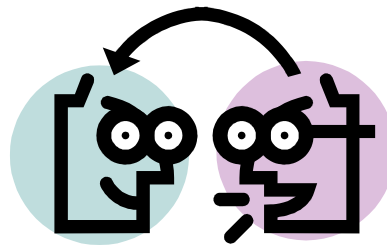
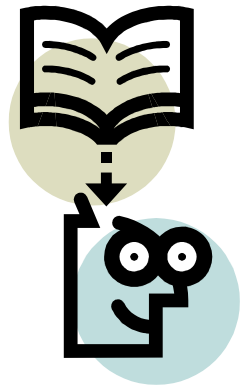


“Para quem tem um martelo, tudo parece prego”

- Cuidado ao escolher o processo, o método e a ferramenta
 - “Muito ajuda quem não atrapalha”
- Não existe o melhor processo, método ou ferramenta
 - Depende da equipe, do projeto, da empresa,

Processos implícitos x explícitos

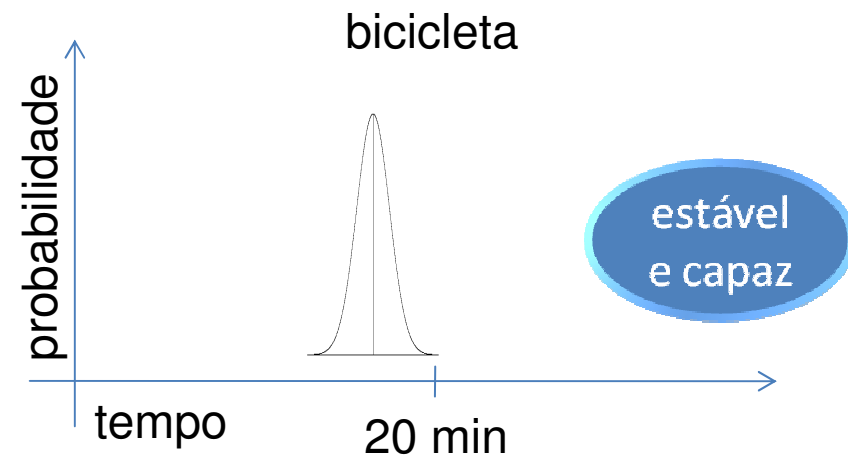
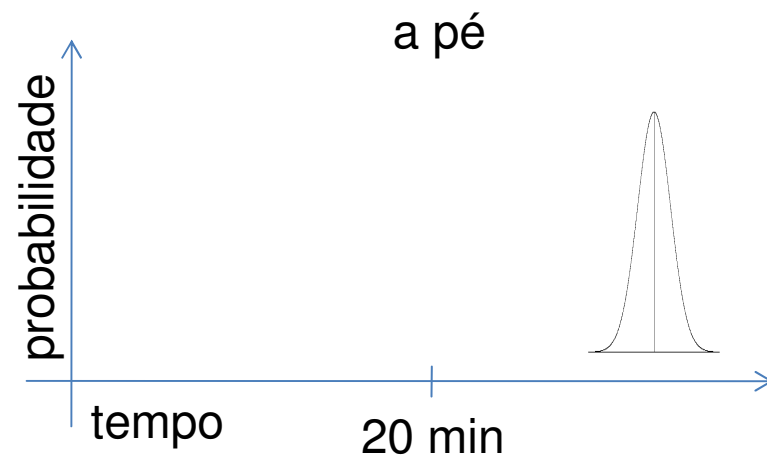
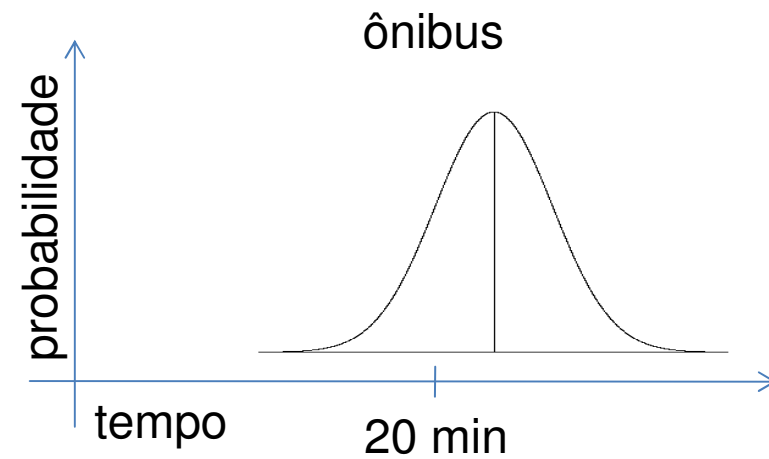
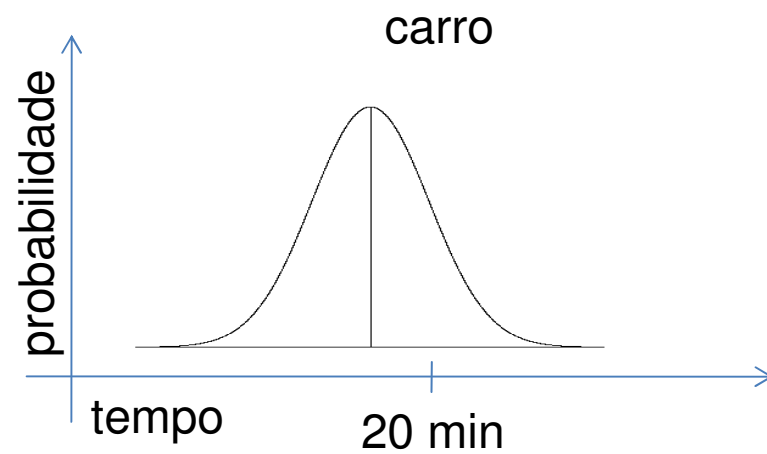
- Lembrem-se: Processos sempre existem, seja de forma implícita ou explícita!
 - Processos implícitos são difíceis de serem seguidos, em especial por novatos
 - Processos explícitos estabelecem as regras de forma clara



Processos estáveis x capazes

- Nem sempre o processo “mais rápido” é um processo estável ou capaz
- Problema:
 - Ir em até 20 minutos de Icaraí para São Francisco
- Processos
 - Ir de carro
 - Ir de ônibus
 - Ir de bicicleta
 - Ir a pé
- Qual é o processo mais rápido num cenário ótimo?
- Quais processos são estáveis?
- Quais processos são capazes?

Processos estáveis x capazes



Modelos de Maturidade

Métrica de Qualidade

Modelos de Maturidade

- A qualidade do produto está intimamente ligada à qualidade do processo
- Um modelo de maturidade é uma coleção estruturada de elementos que descrevem certos aspectos da maturidade de uma organização
- Servem para guiar empresas na busca por qualidade
- Não determinam como algo deve ser feito (não é uma metodologia), mas sim o que deve ser feito (melhores práticas)
- Principais modelos em uso no Brasil
 - CMMI (Capability Maturity Model Integration)
 - MPS.BR

CMMI

- CMMI é uma evolução do CMM para estabelecer um modelo único
- CMM (Capability Maturity Model) tem como objetivo estabelecer – com base em estudos, históricos e conhecimento operacional – um conjunto de "melhores práticas" para diagnóstico e avaliação de maturidade do desenvolvimento de softwares em uma organização
- O CMM fornece às organizações orientação sobre como ganhar controle do processo de desenvolvimento de software e como evoluir para uma cultura de excelência na gestão de software.

CMMI

- 22 áreas de processo:
 - Gerência de Configuração (controle da manutenção),
 - Planejamento de Projeto (planifica tempo e custo do projeto),
 - Validação e verificação de software (Estamos construindo o produto certo?, Estamos construindo certo o produto?)
 - ...

- 5 níveis de maturidade

➤ Nível 1: inicial (*ad-hoc*)

- Nenhum processo é implementado formalmente
- O software é desenvolvido de maneira improvisada (*ad-hoc*)

➤ Nível 2: gerenciado

- Projeto executado de acordo com o planejado quando as etapas são seguidas
- Foco no gerenciamento básico do processo
- Gerência de requisitos
- Planejamento do projeto
- Monitoração e Controle do projeto
- Gerência de Acordos com Fornecedores
- Garantia da Qualidade do Processo e do Produto
- Medição e Análise
- Gerência de configuração

➤ Nível 3: definido (padronização do processo)

- Processo descrito em forma de padrões, procedimentos, métodos
- Foco na padronização do processo organizacional
- Definição do processo organizacional
- Treinamento organizacional
- Gerência integrada do projeto
- Desenvolvimento de Requisitos
- Solução Técnica
- Integração do Produto
- Verificação
- Validação
- Gerência de Riscos
- Análise de Decisão e Resolução

➤ Nível 4: gerenciado quantitativamente

- Medição e controle do processo
- Foco no controle quantitativo do processo
- Gerência quantitativa do processo
- Desempenho do Processo Organizacional

➤ Nível 5: otimizado

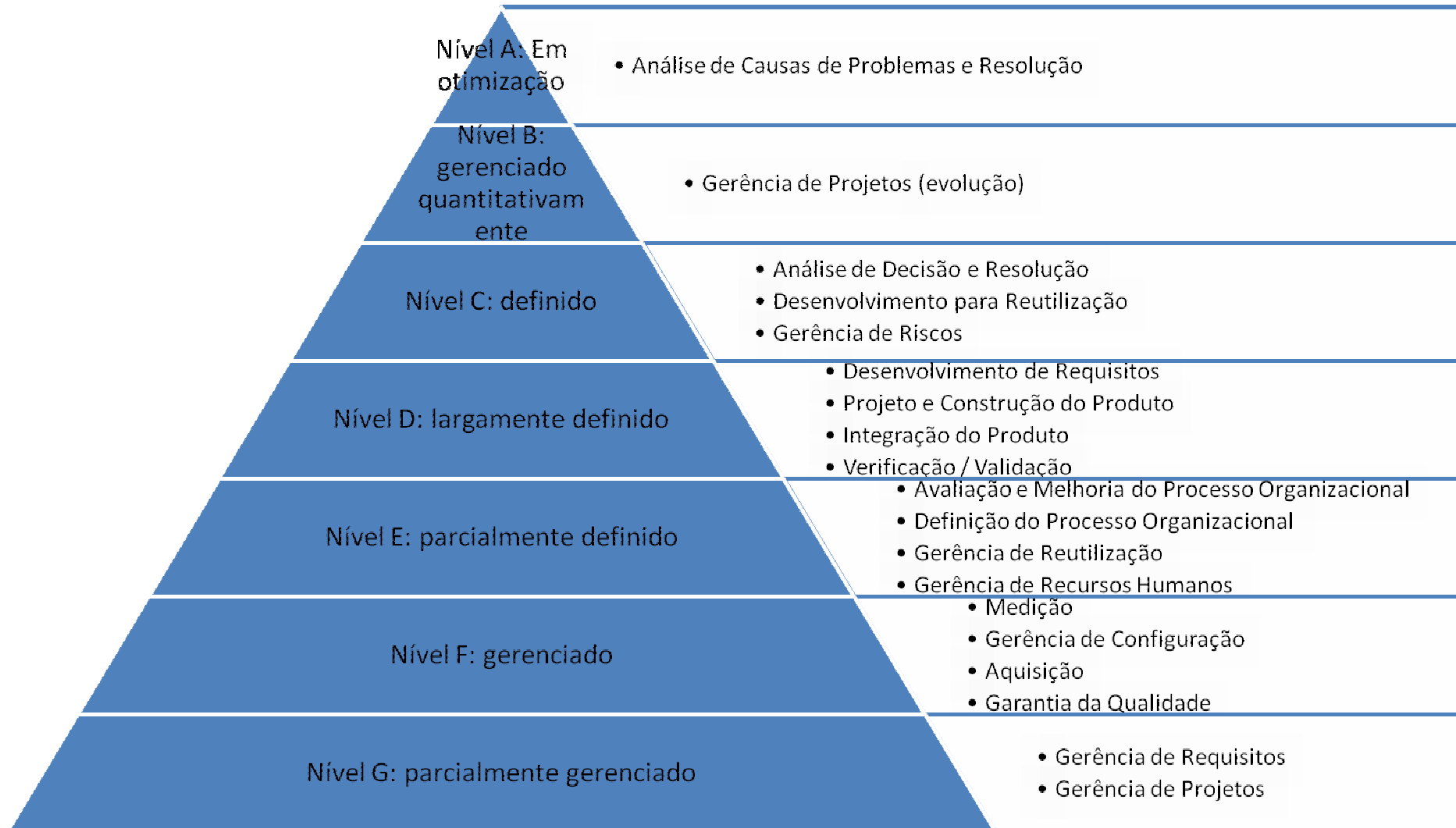
- Foco na melhoria contínua do processo
- Implantação de Inovações na Organização
- Análise de Causas e Resolução

- Modelo brasileiro semelhante ao CMMI
 - Foco nas pequenas e médias empresas brasileiras
 - Menor custo para implementação e avaliação
 - Mais degraus intermediários, ajudando na melhoria progressiva

- Modelo com 19 processos divididos em 7 níveis de maturidade

- Mapeamento para o CMMI
 - Nível 5 = A
 - Nível 4 = B
 - Nível 3 = C
 - Nível 2 = F

MPS.BR



Certificação ou melhoria?

- Avaliações CMMI e MPS.BR tem como foco a melhoria contínua, e não a certificação
- Resultado de uma avaliação
 - Nível atingido
 - Pontos fortes
 - Pontos fracos
 - Oportunidades de melhoria
- Validade de 3 anos

Processo de qualidade

- Última palavra para medir a qualidade de um processo:
Satisfação do Cliente

- Outros indicadores importantes
 - Qualidade dos produtos gerados
 - Custo real do projeto
 - Duração real do projeto

Modelos de Ciclo de Vida de Projeto

Modelos de ciclo de vida de projeto

- Existem alguns processos pré-fabricados
 - Esses processos são conhecidos como modelos de ciclo de vida
 - Esses processos apresentam características predefinidas

- Devem ser adaptados para o contexto real de uso
 - Características do projeto
 - Características da equipe
 - Características do cliente

- Exemplos:
 - Cascata
 - Incremental
 - RAD (Rapid Application Development)
 - Prototipação
 - Espiral

Ciclo de vida Cascata



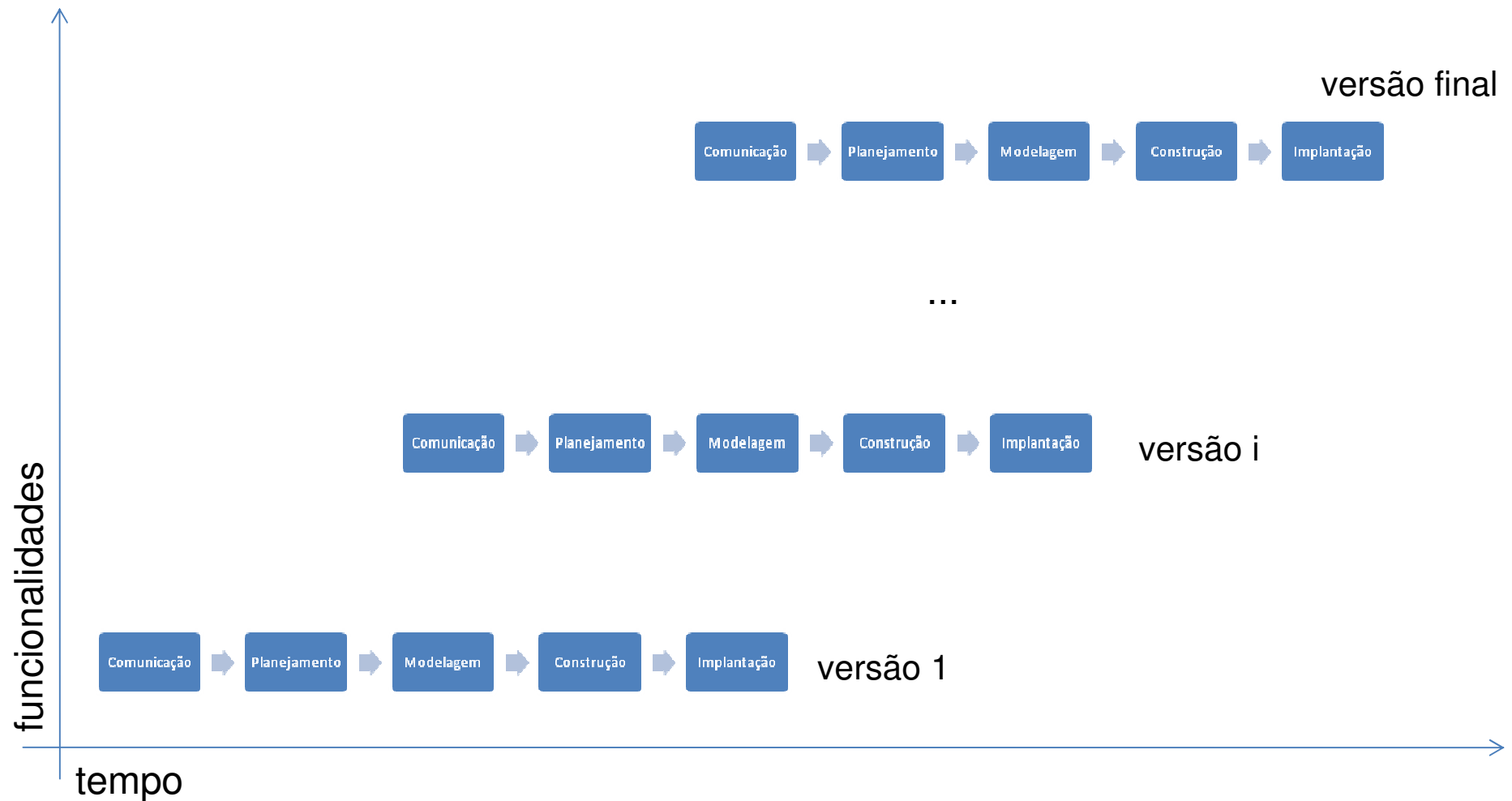
Ciclo de vida Cascata

- Útil quando se tem requisitos estáveis e bem definidos
 - Ex.: Adicionar um novo dispositivo legal em um sistema de contabilidade

- Não lida bem com incertezas

- Fornece pouca visibilidade do estado do projeto
 - Muito tempo para a primeira entrega
 - Dificuldade na obtenção de feedback do cliente

Ciclo de vida Incremental



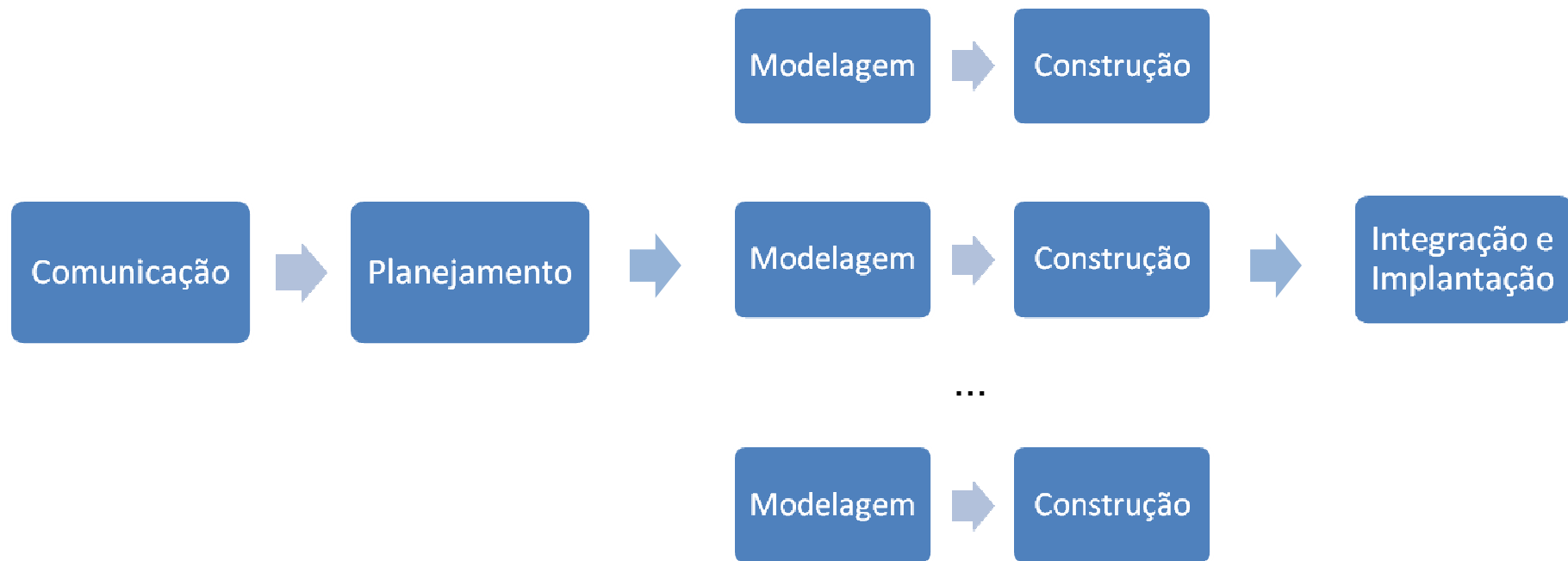
Ciclo de vida Incremental

- Faz entregas incrementais do software
 - Cada incremento é construído via um mini-cascata
 - Cada incremento é um software operacional

- Versões anteriores ajudam a refinar o plano
 - Feedback constante do cliente

- Diminuição da ansiedade do cliente
 - O cliente rapidamente recebe uma versão funcional do software

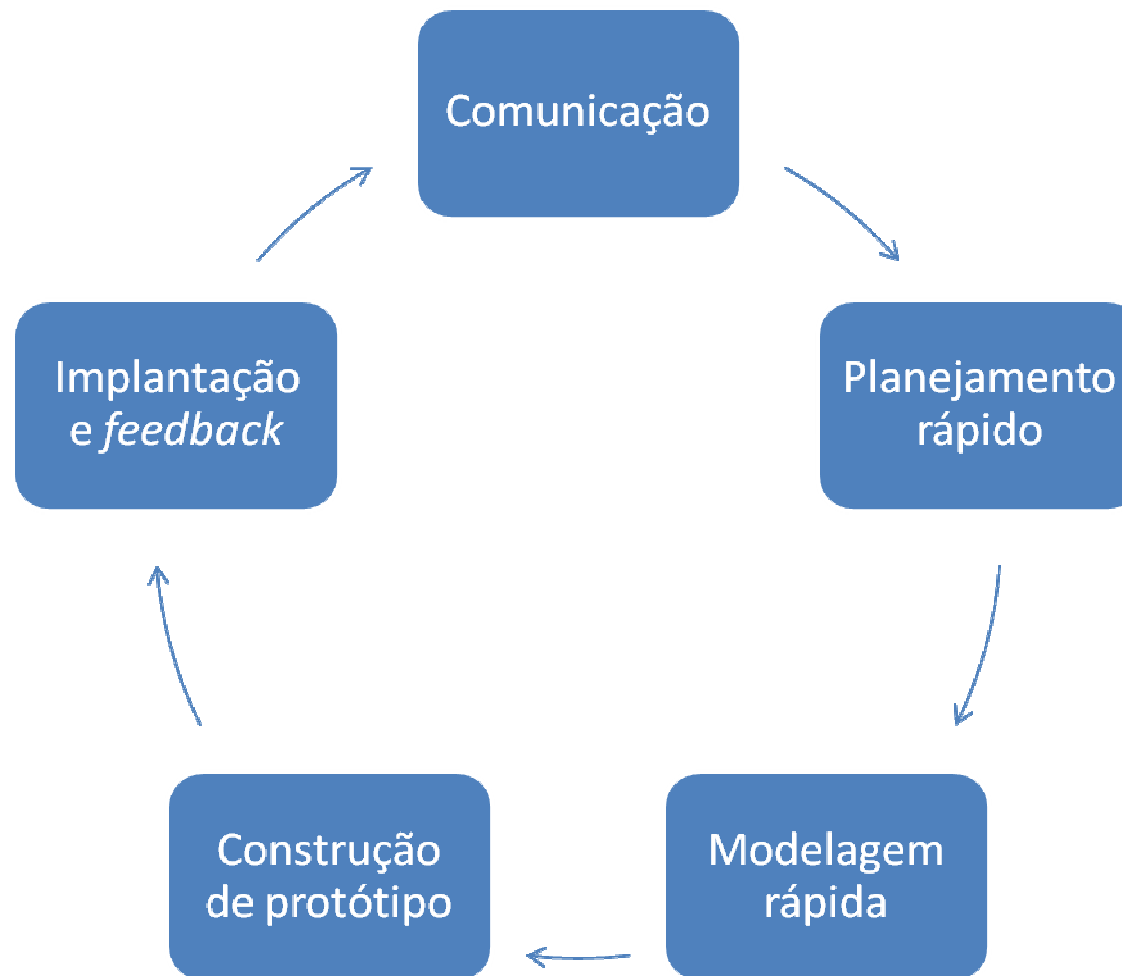
Ciclo de vida RAD (Rapid Application Development)



Ciclo de vida RAD

- Funcionamento equivalente ao cascata
- Principais diferenças
 - Visa entregar o sistema completo em 60 a 90 dias
 - Múltiplas equipes trabalham em paralelo na modelagem e construção
 - Assume a existência de componentes reutilizáveis e geração de código
- Difícil de ser utilizado em domínios novos ou instáveis

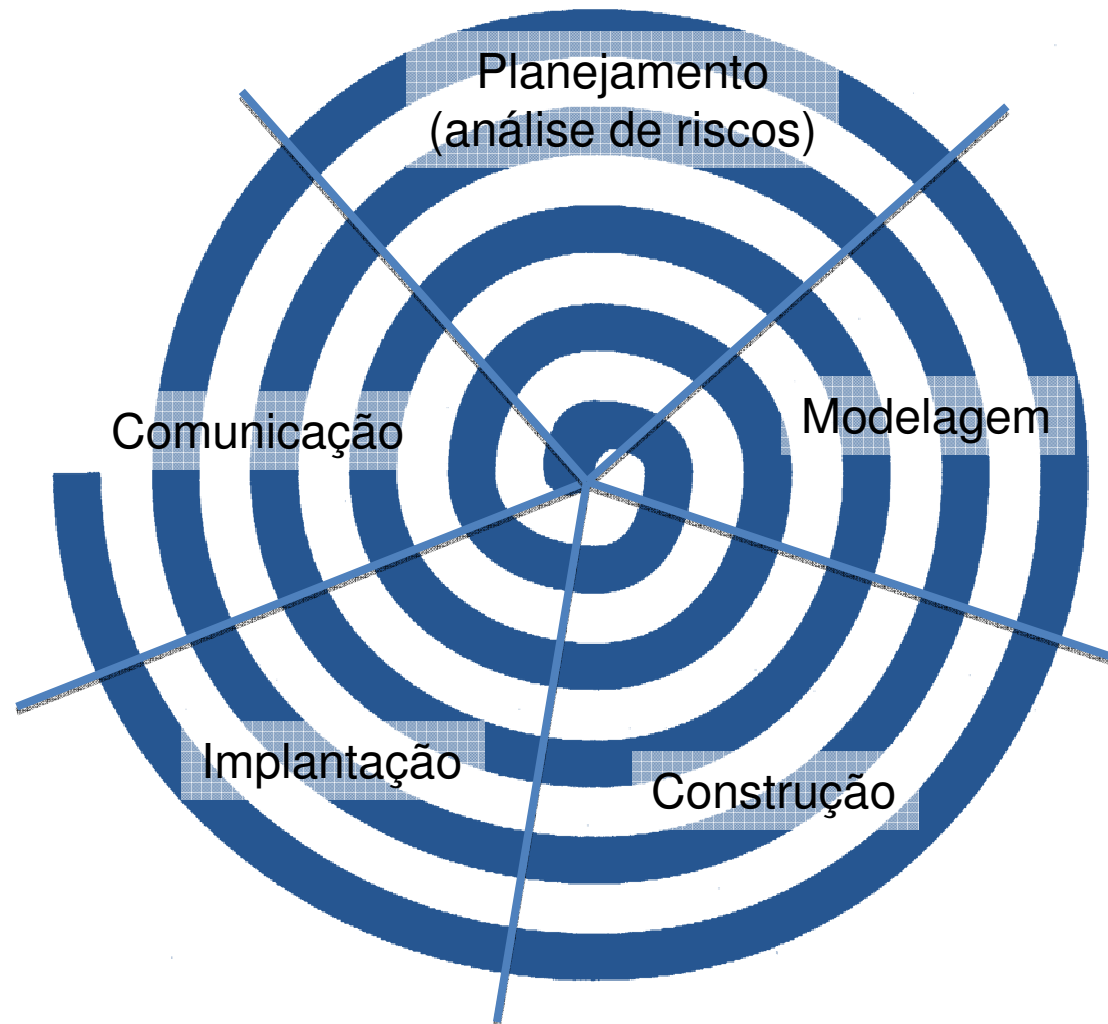
Prototipação



Prototipação

- Usualmente utilizado como auxílio a outro modelo de ciclo de vida
- Útil para
 - Validar um requisito obscuro com o cliente
 - Verificar o desempenho de um algoritmo específico
 - Gerenciar o risco do desenvolvimento
- Protótipo deveria ser jogado fora no final
 - Protótipos não são produtos
 - Usualmente os clientes desejam colocar protótipos em produção

Ciclo de vida Espiral



Ciclo de vida Espiral

- Foco principal no gerenciamento de riscos
- A cada ciclo
 - O conhecimento aumenta
 - O planejamento é refinado
 - O produto gerado no ciclo anterior é evoluído (não é jogado fora)
- Cada ciclo evolui o sistema, mas não necessariamente entrega um software operacional
 - Modelo em papel
 - Protótipo
 - Versões do produto
 - Etc.
- O tempo no desenvolvimento pode aumentar mas ocorre a diminuição dos riscos

Outros ciclos de vida

➤ Métodos formais

- Uso de formalismos matemáticos
- Alto nível de complexidade
- Usualmente aplicado somente a software crítico

➤ Processo Unificado

- Tentativa de obter o que há de melhor em cada modelo de ciclo de vida

Exercício

- Sua empresa foi contratada para fazer um software para um cliente. Qual é o melhor modelo de ciclo de vida para o caso abaixo?
- a) Este cliente possui os requisitos do produto que deseja muito bem especificados.
- b) Ele deseja receber o produto em etapas, i.e., deseja ver diferentes versões do produto, e não somente o produto já finalizado no prazo final de entrega.
- c) Sua empresa não está acostumada a desenvolver produtos para este domínio, por tanto obter feedback do cliente é importante, e cuidado com os riscos no desenvolvimento do produto.
- d) O prazo de entrega do produto é de 80 dias

Software x Hardware

Software x Hardware

➤ Software é desenvolvido

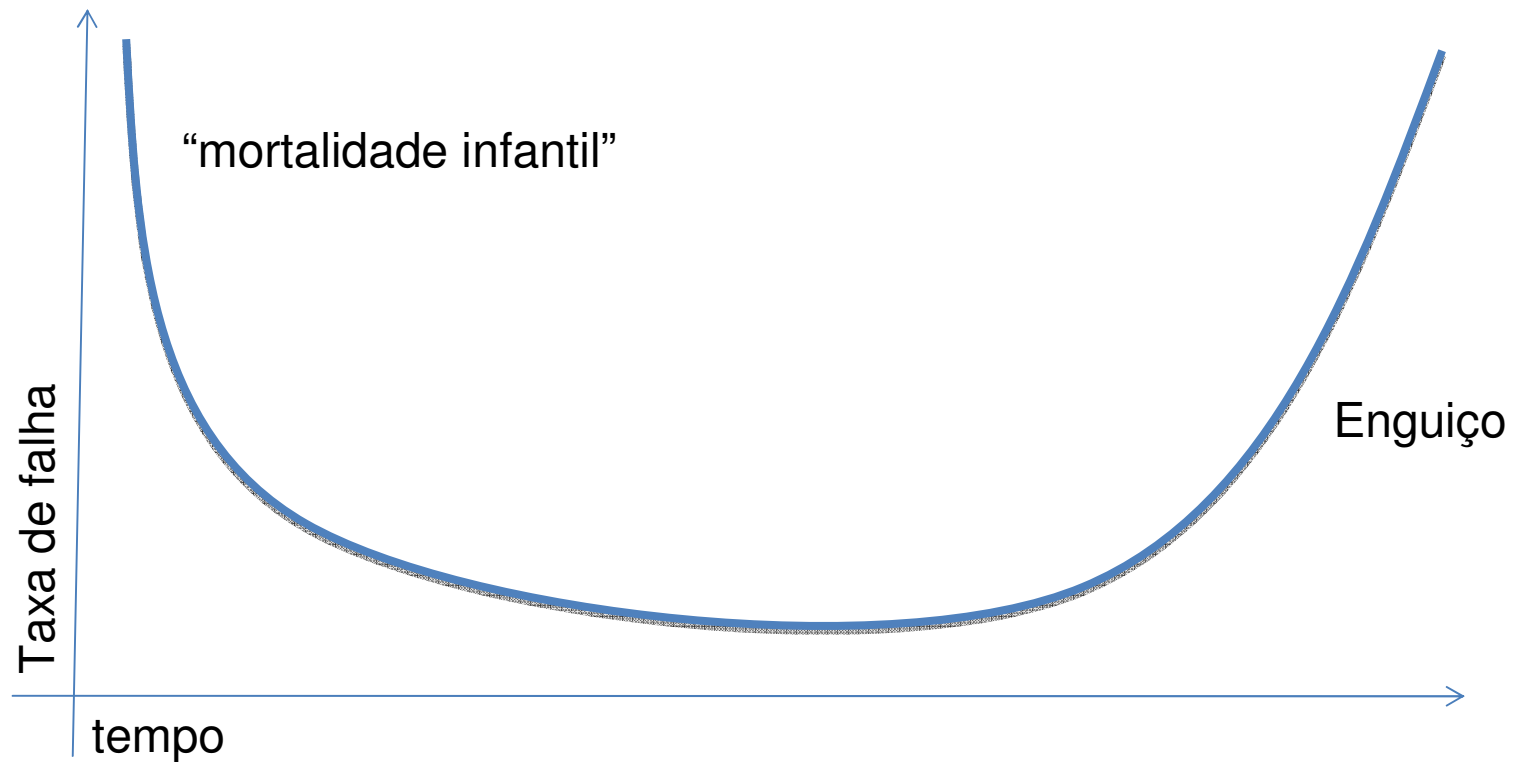
- Alto custo de criação
- Baixo custo de reprodução
- Não enguiça, mas deteriora
- Defeitos no produto usualmente são conseqüências de problemas no processo de desenvolvimento

➤ Hardware é manufaturado

- Alto custo de reprodução
- Pode enguiçar
- Defeitos podem vir tanto da concepção quanto da produção
- Pode ser substituído na totalidade ou em partes

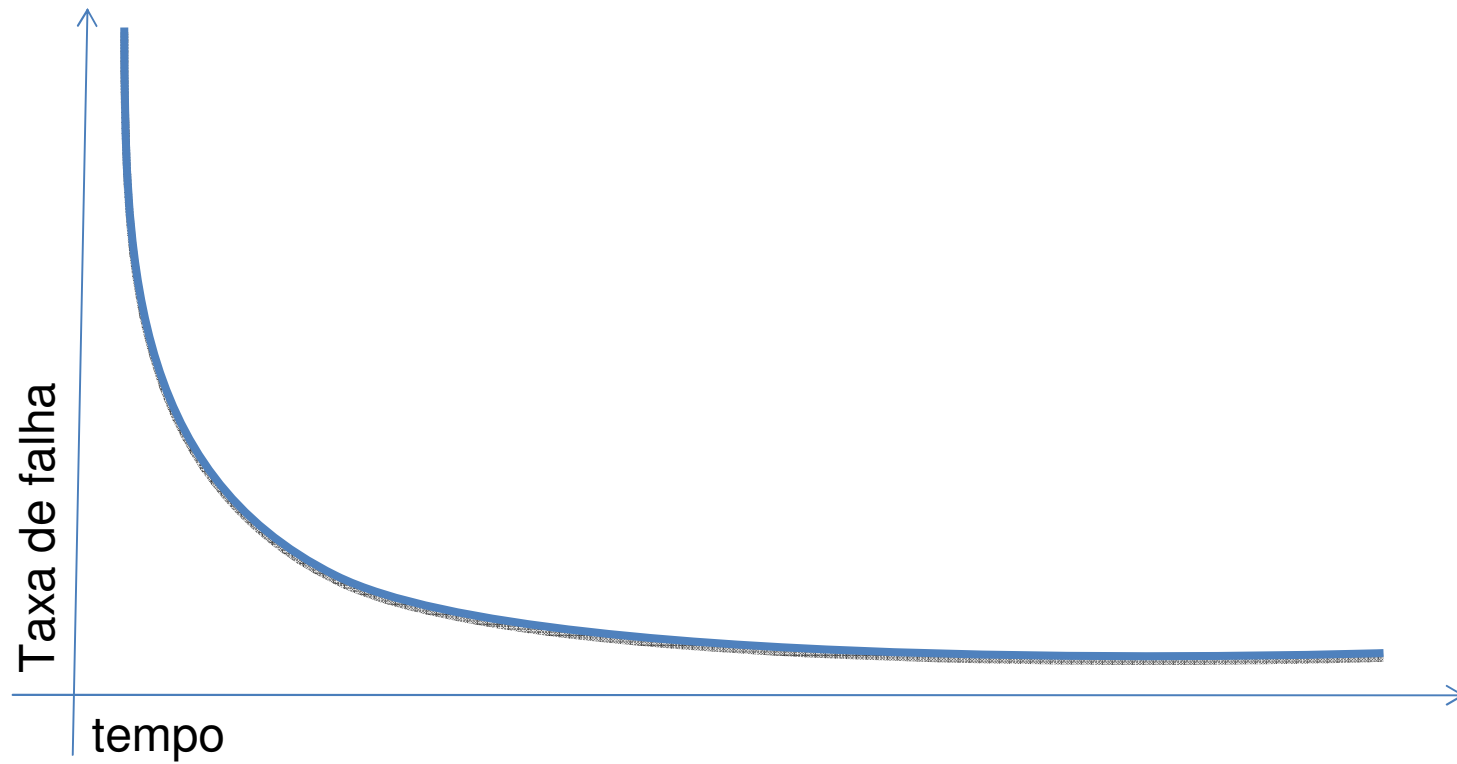
Software x Hardware

➤ Curva de falha de hardware



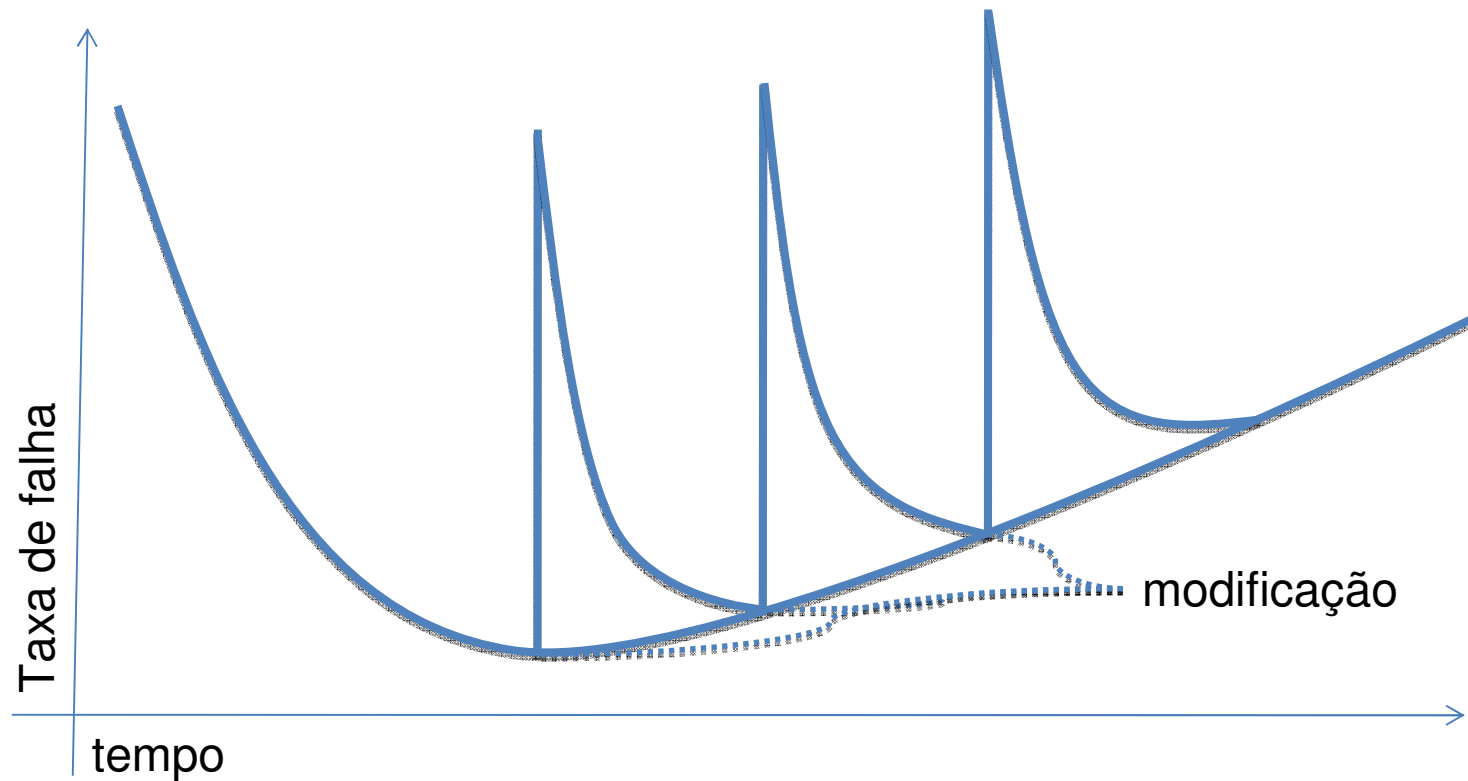
Software x Hardware

- Curva ideal de falha de software



Software x Hardware

➤ Curva real de falha de software



Manutenção



Por que fazer bem feito?

➤ Por que é mais barato!

- Historicamente, 60% a 80% do esforço total ocorre na manutenção

➤ Por que é mais rápido!

- Não ter tempo para fazer bem feito agora significa ter tempo para refazer depois

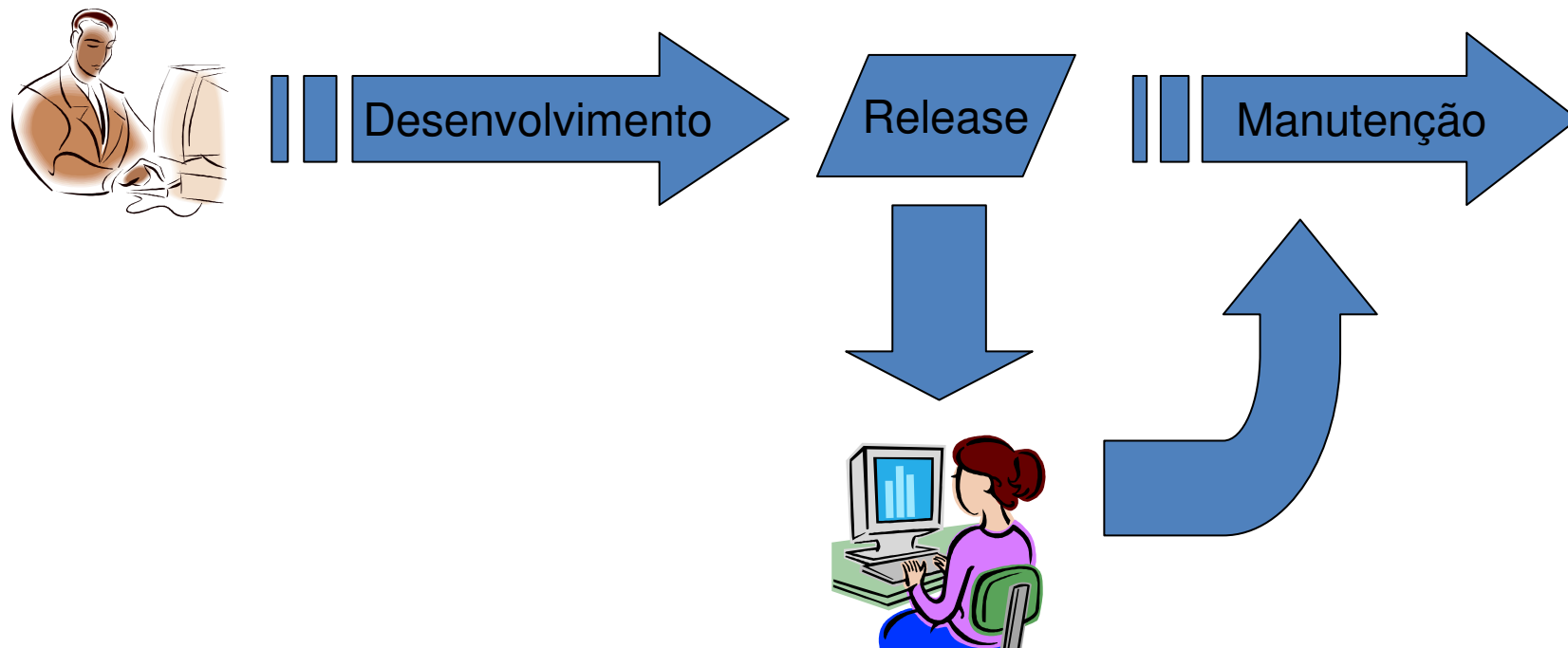
➤ Por que é mais fácil!

- Desenvolvimento ocorre uma única vez
- Manutenção é para sempre

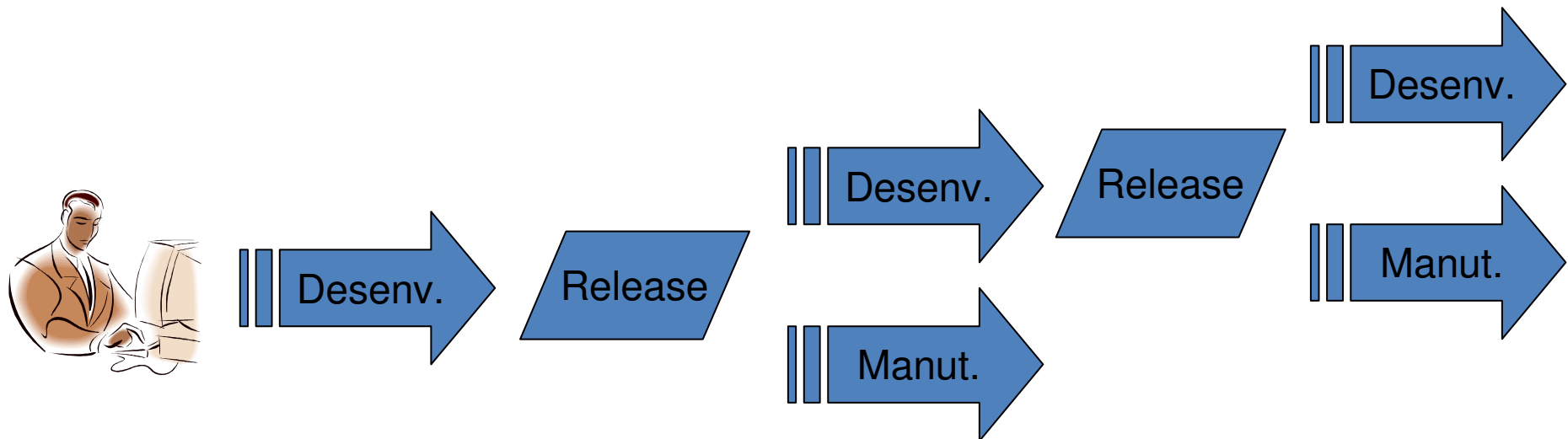
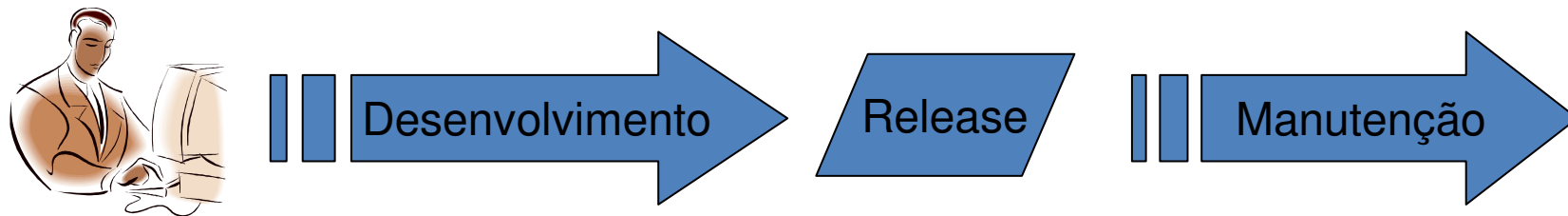
O que é a manutenção?

- O processo de **modificar um sistema de software** ou componente, **depois da entrega**, para **corrigir falhas**, **melhorar desempenho ou outros atributos**, ou **adaptar a mudanças no ambiente**.

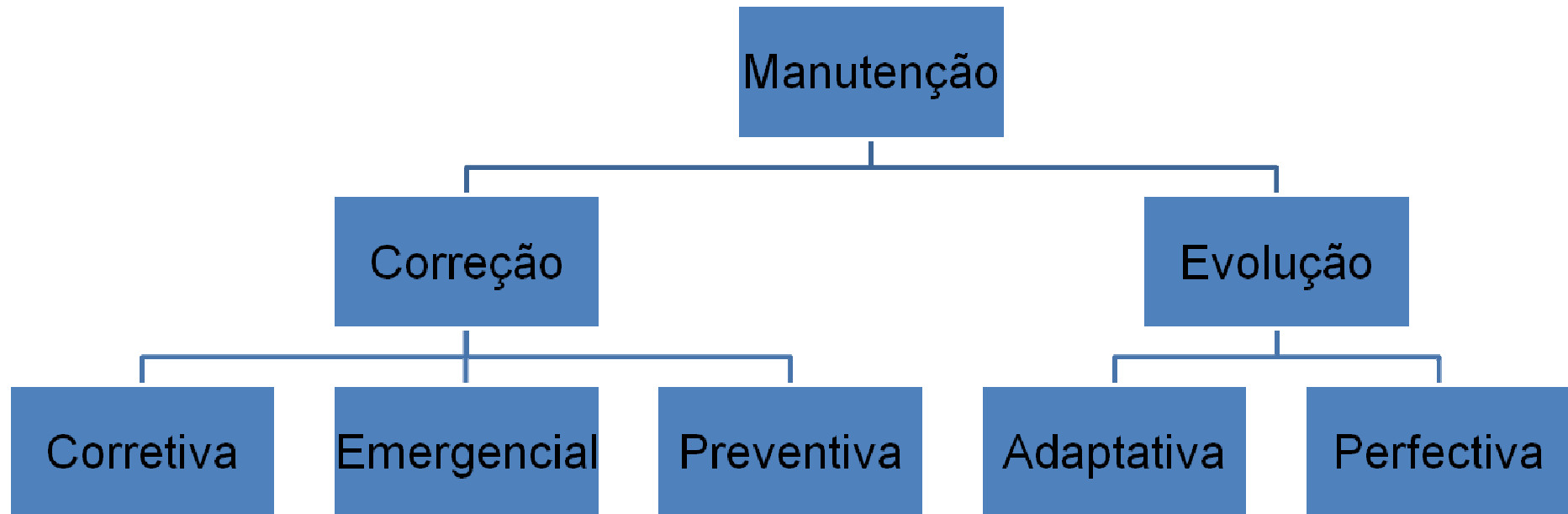
IEEE Std 620.12 1990



Quando inicia a manutenção?



Quais são os tipos de manutenção?



Quais são os tipos de manutenção? --- Correção

➤ Manutenção corretiva

- Reativa
- Corrige problemas reportados
- Faz o software voltar a atender aos requisitos

➤ Manutenção emergencial

- Não programada
- Mantém temporariamente o sistema funcionando
- Necessita uma manutenção corretiva posterior

➤ Manutenção preventiva

- Pró-ativa
- Corrige problemas latentes

Quais são os tipos de manutenção? --- Evolução

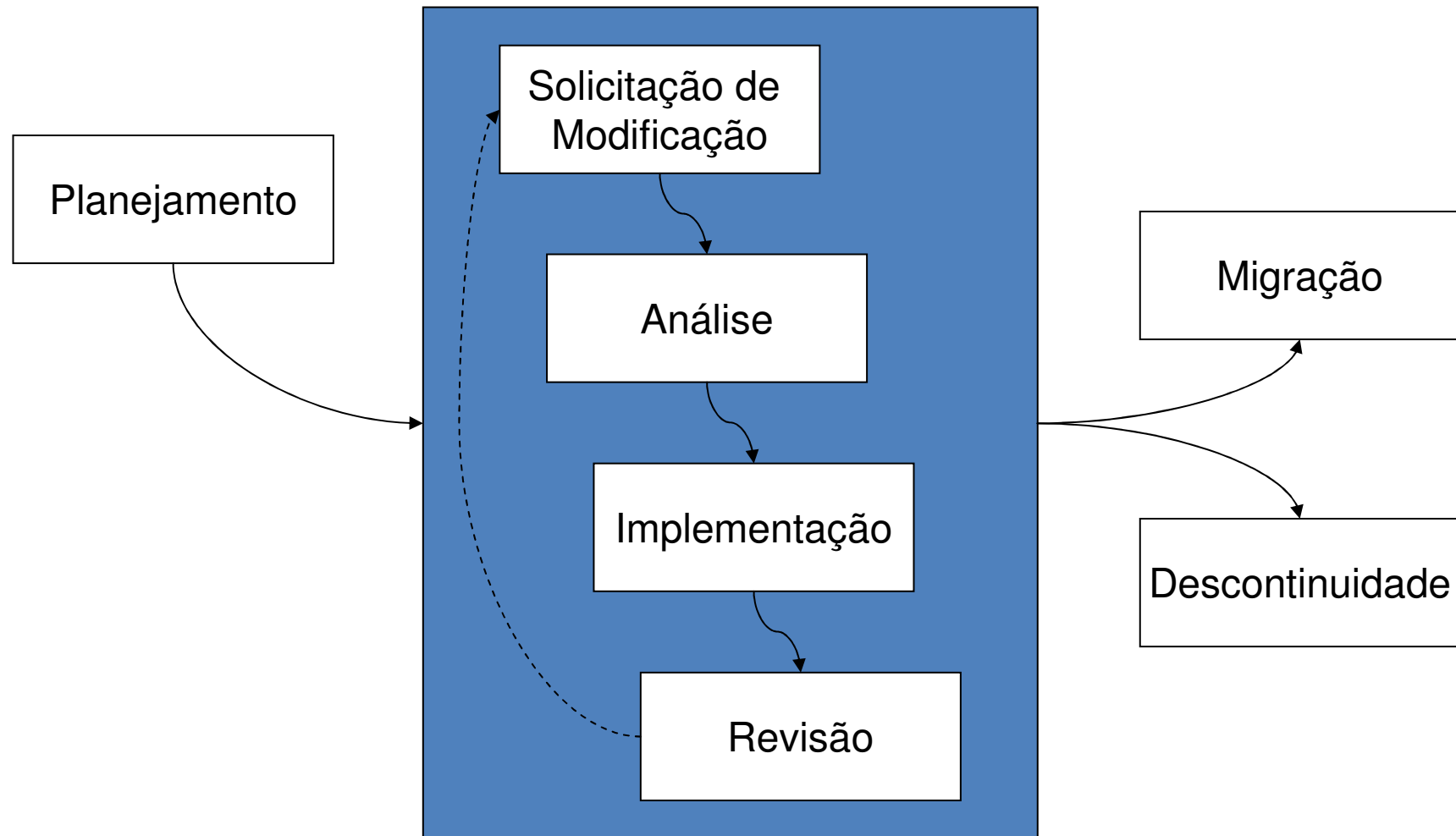
➤ Manutenção adaptativa

- Mantém o software usável após mudanças no ambiente

➤ Manutenção perfectiva

- Provê melhorias para o usuário
- Melhora atributos de qualidade do software

Processo de manutenção



Exercício:

- Após a sua empresa entregar o produto ao cliente:
 - O usuário reportou um problema
 - A empresa detectou possíveis geradores de problemas futuros
 - A empresa deixou o produto mais seguro

- Quais são os tipos de manutenção que precisam ser realizadas neste software?

Alguns Mitos

Mitos gerenciais

- Basta um bom livro de ES para fazer bom software
 - Um bom livro certamente ajuda, mas ele precisa refletir as técnicas mais modernas de ES e ser lido!

- Se estivermos com o cronograma atrasado, basta adicionar mais gente ao projeto
 - Adicionar gente a um projeto atrasado faz o projeto atrasar mais!
 - As pessoas que estão entrando terão que aprender sobre o projeto antes de começar a ajudar no desenvolvimento
 - As pessoas que estão no desenvolvimento, terão que parar para explicar aos que estão entrando

- Se o projeto for terceirizado, todos os meus problemas estão resolvidos
 - É mais difícil gerenciar projetos terceirizados do que projetos internos!

Mitos do cliente

- Basta dar uma idéia geral do que é necessário no início
 - Requisitos ambíguos normalmente são uma receita para desastre!
 - Comunicação contínua com o cliente é fundamental!

- Modificações podem ser facilmente acomodadas, porque software é flexível
 - O impacto de modificações no software varia em função da modificação e do momento em que ela é requisitada!
 - Comunicação contínua com o cliente é fundamental!

Mitos do desenvolvedor I/II

- Assim que o código for escrito o trabalho termina
 - 60% a 80% do esforço será gasto depois que o código foi escrito! (implantação do sistema, testes, manutenção,)
 - Vale a pena se esforçar para chegar a um bom código (boa documentação, bom projeto, etc.)!

- Só é possível verificar a qualidade de um software quando o executável existir
 - Revisões usualmente são mais eficazes que testes, e podem ser utilizadas antes do software estar executável!

Mitos do desenvolvedor II/II

- O único produto a ser entregue em um projeto é o código
 - Além do código, documentações tanto para a manutenção quanto para o uso são fundamentais!

- Engenharia de software gera documentação desnecessária
 - Engenharia de software foca em criar qualidade, e não criar documentos!
 - Algum grau de documentação é necessário para evitar retrabalho!
 - Questione sempre que encontrar um documento desnecessário para o projeto!

7 princípios de Hooker

- Tem que existir uma razão para se fazer software
 - Se não for possível identificar essa razão, é melhor não fazer
 - Fazer software, em última instância, consiste em “agregar valor para o usuário”
 - É importante enxergar os reais requisitos do software!

- Keep it simple, sir! (KISS)
 - “um projeto deve ser o mais simples possível, mas não mais simples que isso”
 - As soluções mais elegantes normalmente são simples
 - Fazer algo simples usualmente demanda mais tempo do que fazer de forma complexa

7 princípios de Hooker

➤ Mantenha o estilo

- O projeto de um software deve seguir um único estilo (estilo de codificação, documentação, teste, um mesmo processo, ...)
- A combinação de diferentes estilos corretos pode levar a um software incorreto
- Padrões e estilos devem ser estabelecidos no início e seguidos por todos

➤ O que é produzido por você é consumido por outros

- Sempre especifique, projete e codifique algo pensando que outros vão ler
- Sempre exija qualidade nos produtos que você consome e forneça qualidade nos produtos que você produz

7 princípios de Hooker

➤ Esteja pronto para o futuro

- Sistemas de boa qualidade têm vida longa
- Projete desde o início pensando na manutenção

➤ Planeje para reutilização

- Pense no problema geral, e não só no problema específico
- Busque por soluções já existentes

➤ Pense!

- “plano é desnecessário, mas planejar é indispensável” – D. Eisenhower
- Avalie alternativas
- Detalhe os riscos

Exercício: Jogo dos “sete” erros

- A nossa empresa fez o levantamento dos requisitos com o cliente tentando esclarecer todas as ambigüidades de uma única vez.
- Após a fase de levantamento dos requisitos, o projeto passou para a fase de codificação.
- Ao final da codificação e geração do executável, o projeto foi testado.
- Só após o teste, a empresa acionou o cliente novamente para a entrega do código gerado.
- Durante a fase de codificação e após verificar um atraso no cronograma, mais profissionais foram incluídos na equipe e parte do projeto foi terceirizada.
- Após a codificação do produto, toda a equipe foi deslocada para o desenvolvimento de outro projeto.

Nós precisamos de ES!



Como o cliente explicou



Como o lider de projeto entendeu



Como o analista planejou



Como o programador codificou



O que os beta testers receberam



Como o consultor de negocios descreveu



Valor que o cliente pagou



Como o projeto foi documentado



O que a assistencia tecnica instalou



Como foi suportado



Quando foi entregue



O que o cliente realmente necessitava

Introdução a Engenharia de Software

Várias transparências foram produzidas
por Leonardo Murta

<http://www.ic.uff.br/~leomurta>