Engenharia de Sistemas de Informação

Marcos L. Chaim

Ciclo de Vida de Software

EACH-USP

Questões

- O que é o ciclo de vida de um produto de software?
- Por que precisamos de modelos de processos de software?
- Quais são os alvos de um processo de software e o que o faz diferente de um processo industrial?

Ciclo de vida

- Da concepção de uma idéia para um produto por meio de:
  - coleta e análise de requisitos;
  - projeto e especificação de arquitetura;
  - codificação e testes;
  - entrega e implantação;
  - manutenção e evolução;
  - recolhimento.

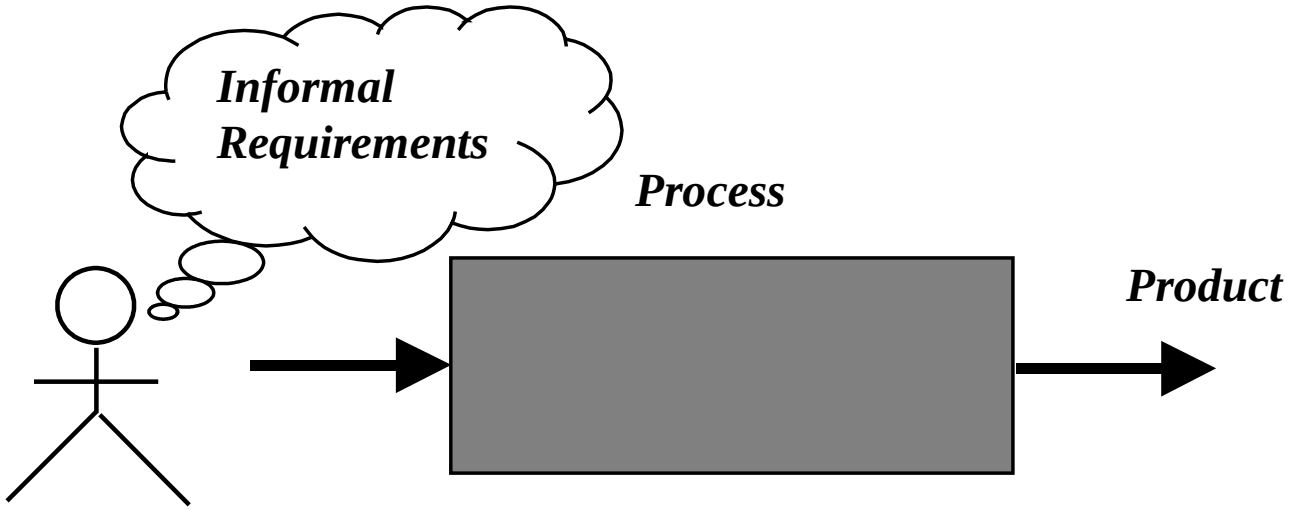
Modelo de Processo de Software

- Tentativa de organizar o ciclo de vida de software pela:
  - Definição de atividades envolvidas na produção de software;
  - Ordem das atividades e seus relacionamentos.
- Alvos de um processo de software
  - Padronização, predictibilidade, produtividade, alta qualidade de produto, habilidade de planejamento de requisitos de tempo e orçamento.

Modelos são necessários

- Sintomas de inadequação:
  - Tempo fixado e custo excedido;
  - Expectativas do usuário não satisfeitas;
  - Pobre qualidade.
- O tamanho e valor econômico de aplicações de software requeriam “modelos de processo” apropriados.

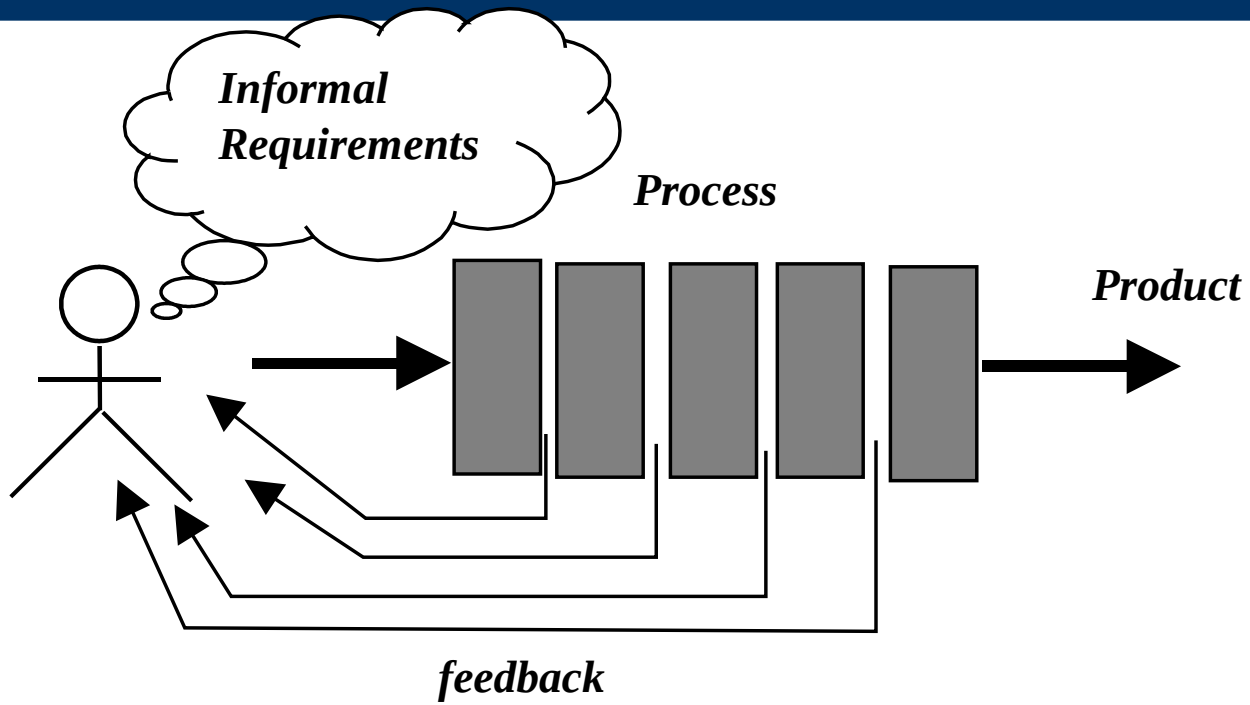
Processo como uma “caixa-preta”



Problemas

- A suposição de que os requisitos podem ser totalmente compreendidos antes do desenvolvimento.
- Interação com o cliente ocorre somente no início (requisitos) e no fim (após a entrega).
- Infelizmente, a suposição quase nunca ocorre.

Processo como uma “caixa-branca”



Vantagens

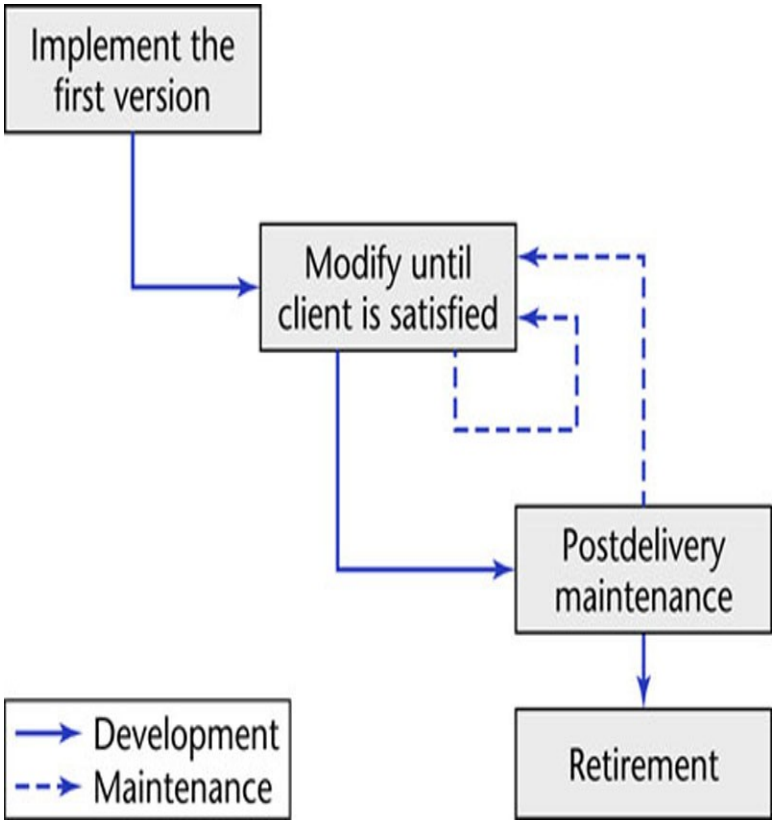
- Reduz os riscos pelo aumento de visibilidade.
- Permite mudança de projeto assim que o projeto progride:
  - Baseado no retorno do usuário.

Processos de desenvolvimento de Software

- Uma vez definido o escopo do software a ser desenvolvido deve-se definir:
  - O método de desenvolvimento;
  - O processo de desenvolvimento.
- Essa definição permite definir:
  - Atividades e tarefas do processo de desenvolvimento de software.
- Há vários modelos de processo de desenvolvimento.

Codifique & Conserte

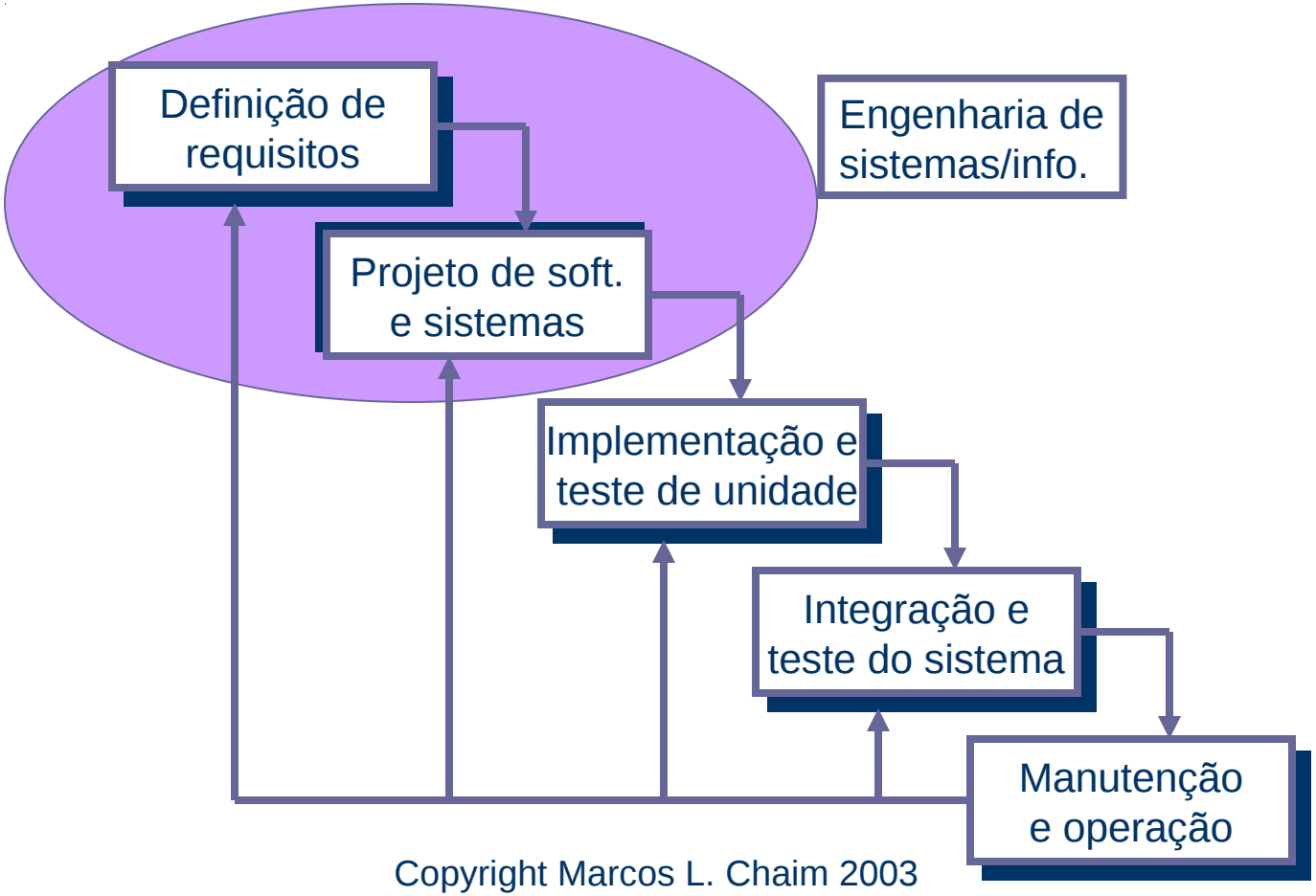
- Sem especificação.
- Sem projeto.
- A maneira mais fácil de desenvolver software.
- A maneira mais cara de manter o software.



Codifique & Conserte

- A abordagem mais antiga
  - Escreva o código.
  - Conserte-o .
    - para eliminar quaisquer defeitos que tenham sidos identificados,
    - para melhorar uma funcionalidade existente ou
    - para adicionar novas características
- Fonte de dificuldades e deficiências:
  - Impossível prever;
  - Impossível gerenciar.

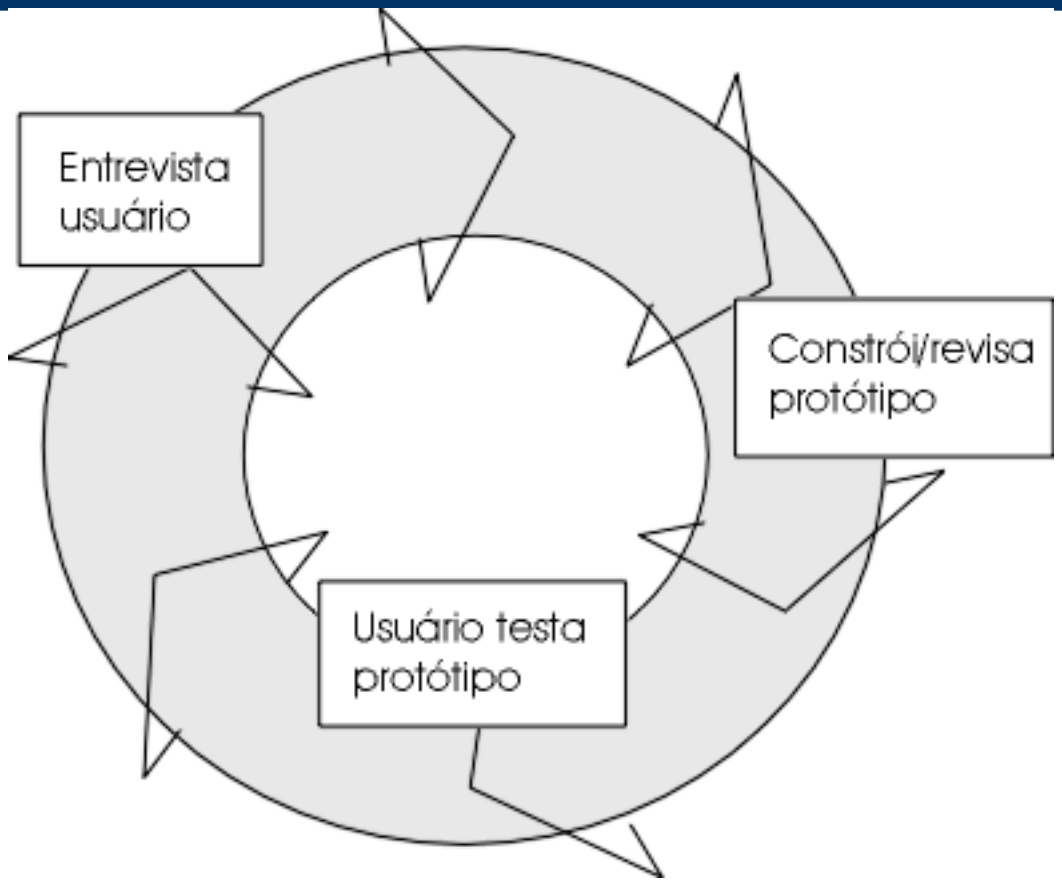
Ciclo de Vida em Cascata



Ciclo de Vida em Cascata

- Problemas:
  - projetos raramente seguem um fluxo seqüencial;
  - difícil de definir todas as restrições *a priori;*
  - primeira versão em um estágio tardio;
- Apesar dos problemas,
  - largamente utilizado ainda hoje;
  - melhor que não ter uma sistemática de desenvolvimento;
- Manutenção é uma atividade pós-entrega do software.

Prototipação



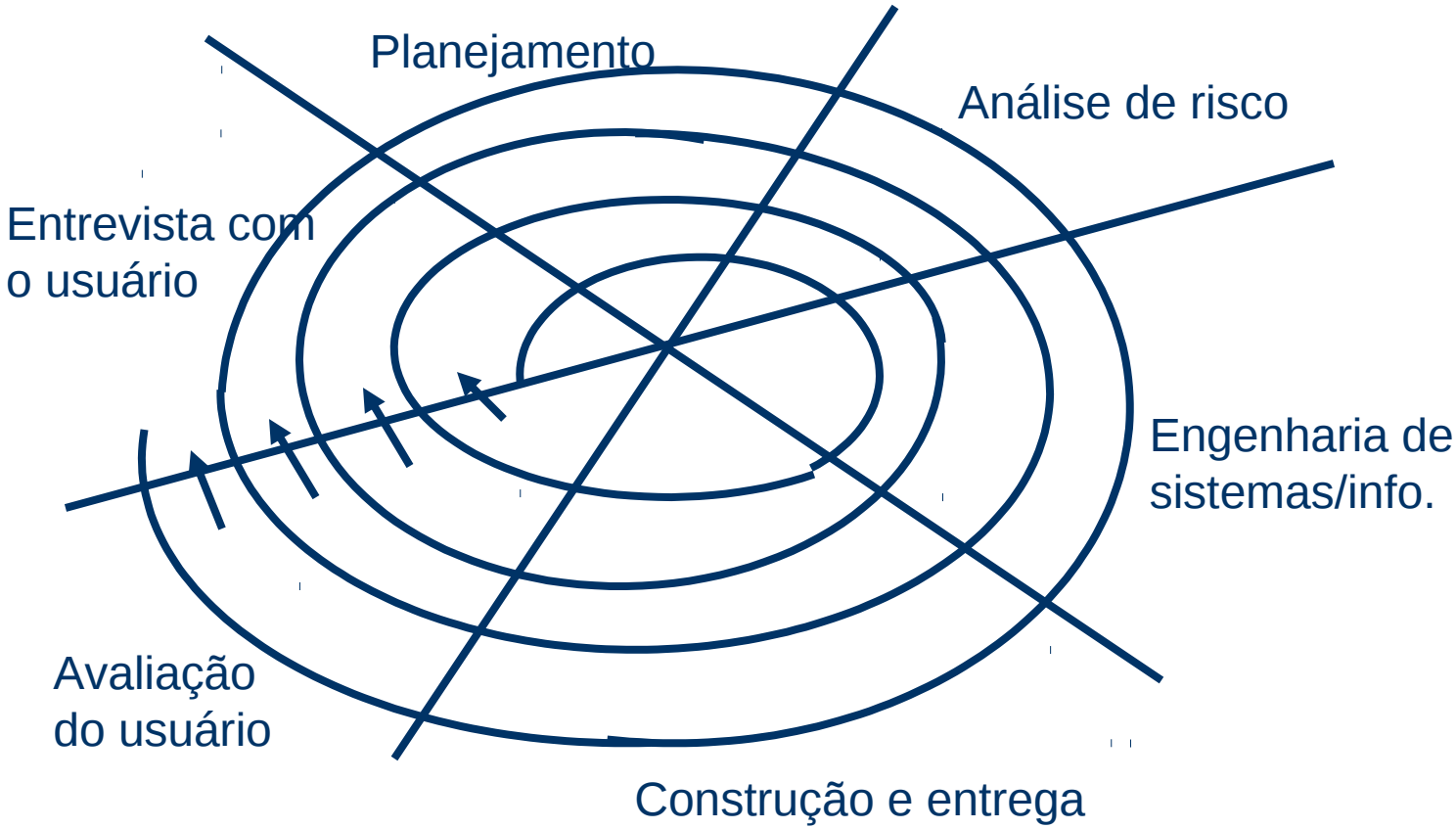
Prototipação

- Benefícios:
  - idéias confusas podem ser identificadas
  - entendimento errado pode ser esclarecido
  - complementar idéias vagas
  - utilidade do sistema antes de pronto

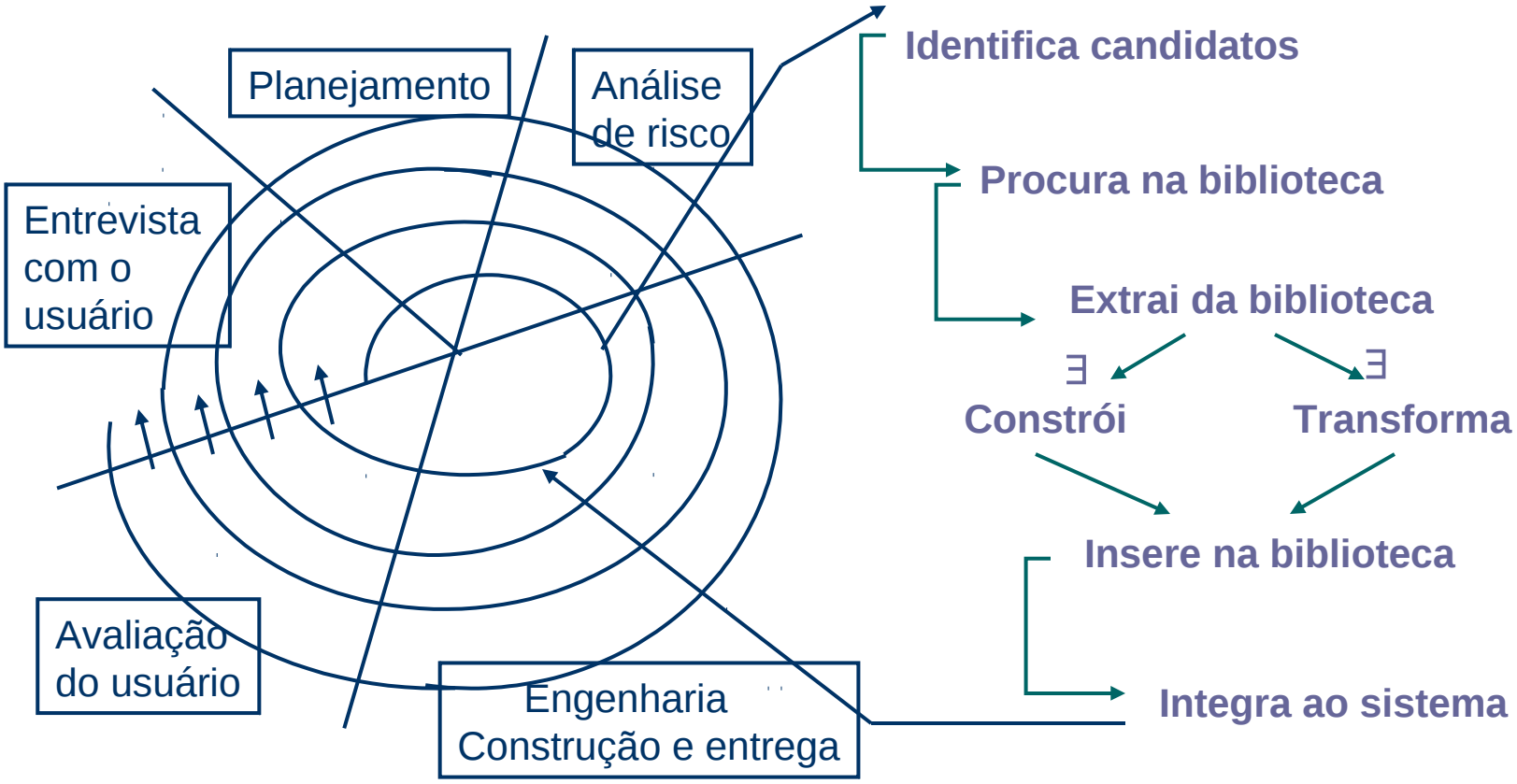
Prototipação

- Problemas:
  - usuários vêem o protótipo como produto final e pensam que o produto está praticamente pronto;
  - ferramentas/linguagem provisórias podem se tornar definitivas - inércia;
  - manutenção de um protótipo tende a ser problemática: falta de estrutura porque não foi preparado para ser a versão final.

Espiral (Boehm)



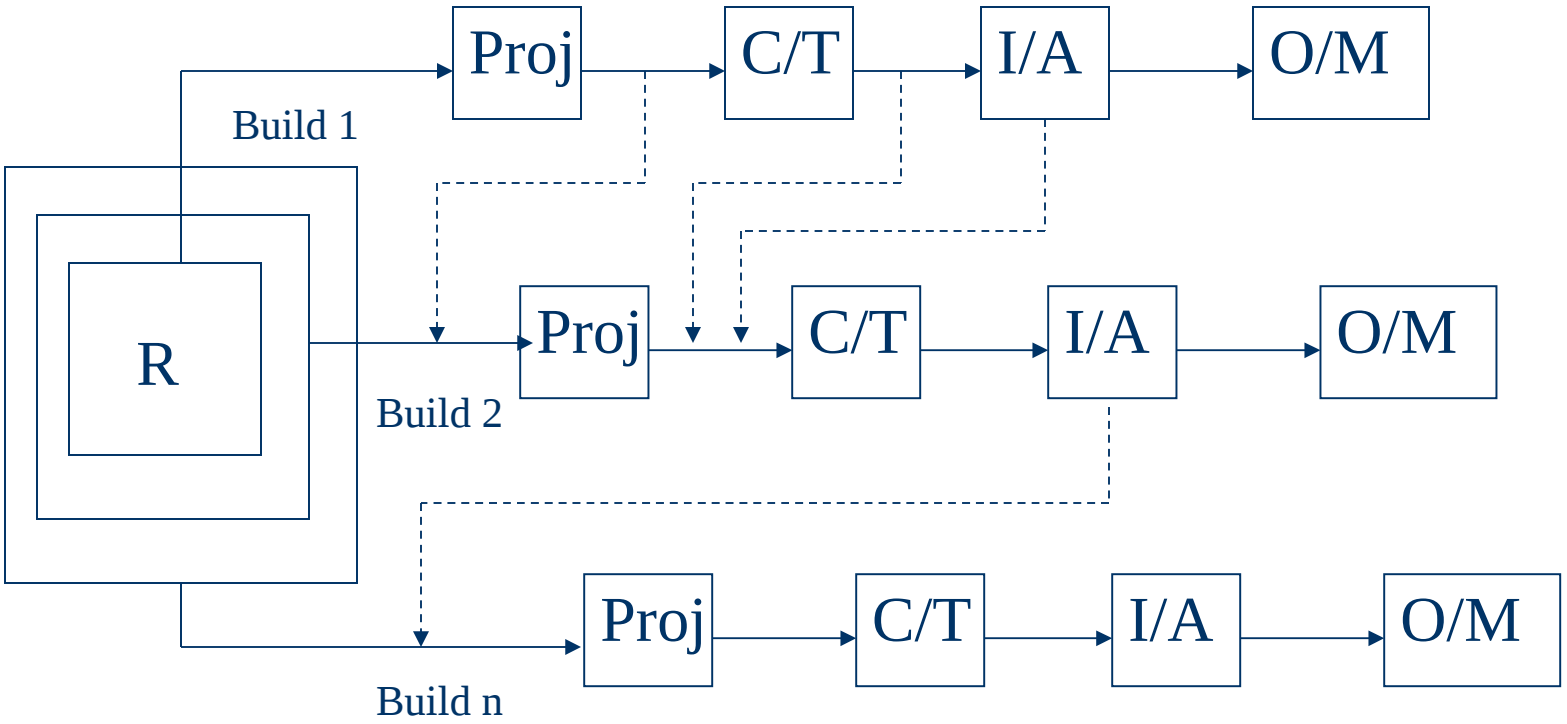
Reutilização de Componentes



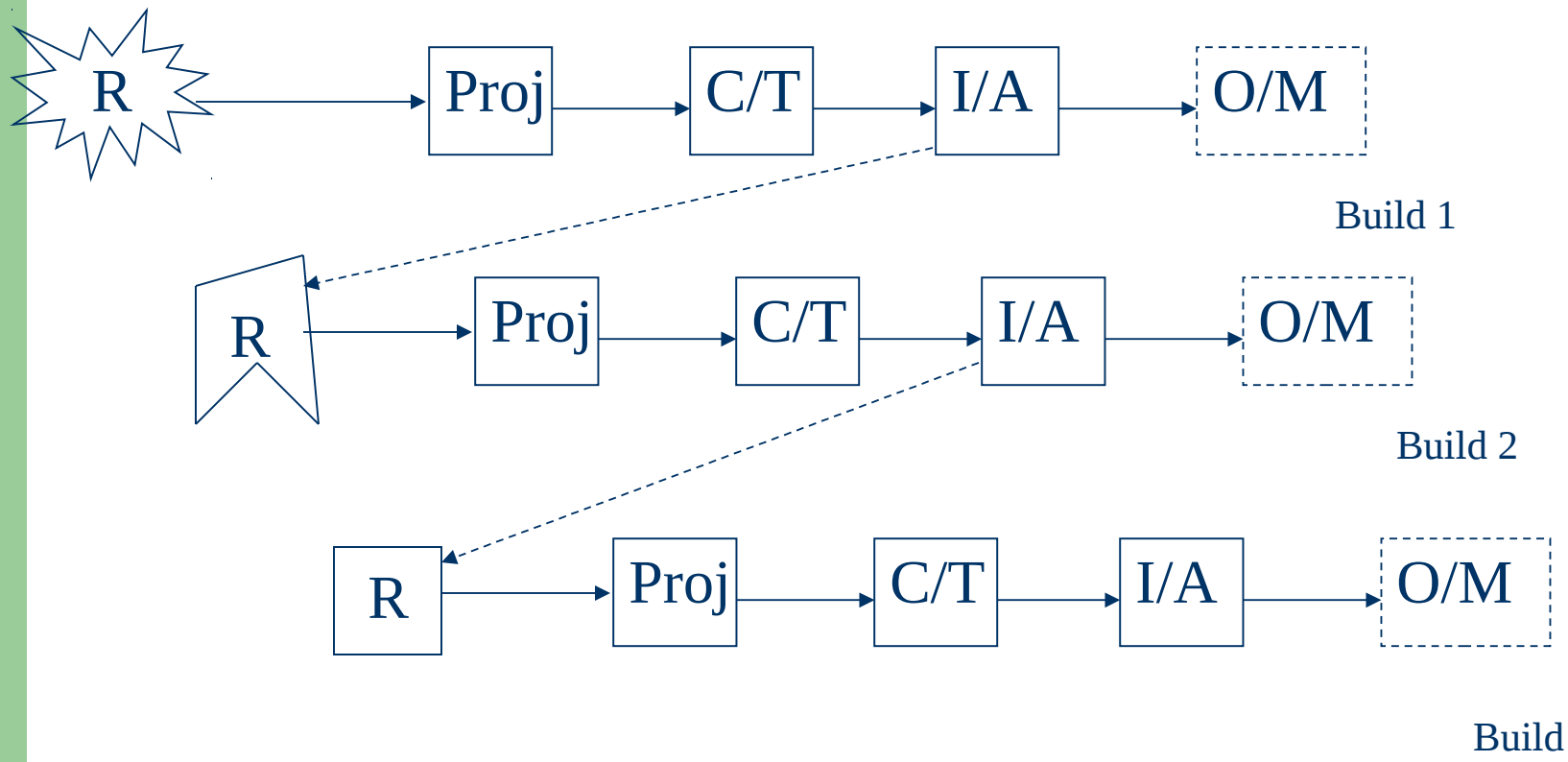
Modelo Espiral

- Cascata + prototipação + análise de risco
- Combinação de paradigmas;
- Versão mais completa a cada ciclo;
- Mais realista para projetos de grande porte;
- E a atividade de manutenção das versões intermediárias? Como é que fica?

Ciclo de Vida Incremental



Ciclo de Vida Evolucionário



Processo Unificado

- É um modelo de processo de software baseado no modelo incremental, visando a construção de software orientado a objetos.

- Usa como notação de apoio a UML (Unified Modeling Language).

Processo Unificado – História

- Raízes no trabalho de Jacobson na Ericsson no final da década de 1960.
- Em 1987 Jacobson iniciou uma companhia chamada de Objectory AB – desenvolvimento de um processo chamado Objectory.
- 1995 – a Rational comprou a Objectory AB, aperfeiçoou o Objectory e foi criado o Processo Objectory da Rational (ROP) (Jacobson, Rumbaugh e Booch).
- Paralelamente desenvolviam a UML.

Processo Unificado – História

- Progresso do ROP e a aquisição e desenvolvimento de ferramentas de desenvolvimento agregaram valor ao ROP.
- 1998 a Rational mudou o nome do ROP para Processo Unificado da Rational (RUP- Rational Unified Process).
- O RUP é uma especialização, com refinamento detalhado, do PU.

O que é o PU?

- É um processo de Software:
  - conjunto de atividades executadas para transformar um conjunto de requisitos do cliente em um sistema de software.
- É um *framework* que pode ser personalizado de acordo com as necessidades específicas e recursos disponíveis para cada projeto.

Elementos do PU

- Um processo descreve:
  - quem (papel) está fazendo o quê (artefato),
  - como (atividades) e
  - quando (disciplina).

Trabalhadores

- Trabalhadores:
  - Um trabalhador é alguém que desempenha um papel e é responsável pela realização de atividades para produzir ou modificar um artefato.

Artefatos

- Porção significativa de informação interna ou a ser fornecida a interessados externos que desempenhe um papel no desenvolvimento do sistema.

Artefatos

- Um artefato é algum documento, relatório, modelo ou código que é produzido, manipulado ou consumido.
  - Exemplos: modelo de caso de uso, modelo do projeto, um caso de uso, um subsistema, um caso de negócio, um documento de arquitetura de software, código fonte, executáveis, etc.

Atividades

- Atividades:
  - tarefa que um trabalhador executa a fim de produzir ou modificar um artefato.

Disciplinas

- Descreve as seqüências das atividades que produzem algum resultado significativo e mostra as interações entre os participantes
- São realizadas a qualquer momento durante o ciclo de desenvolvimento (Fases do PU)
- Requisitos, Análise, Projeto, Implementação e Teste

Princípios básicos do PU

- Desenvolvimento iterativo
- Baseado em casos de uso
- Centrado na arquitetura

Desenvolvimento Iterativo

- O desenvolvimento de um software é dividido em vários ciclos de iteração, cada qual produzindo um sistema testado, integrado e executável.
- Em cada ciclo ocorrem as atividades de análise de requisitos, projeto, implementação e teste, bem como a integração dos artefatos produzidos com os artefatos já existentes.

Desenvolvimento Iterativo

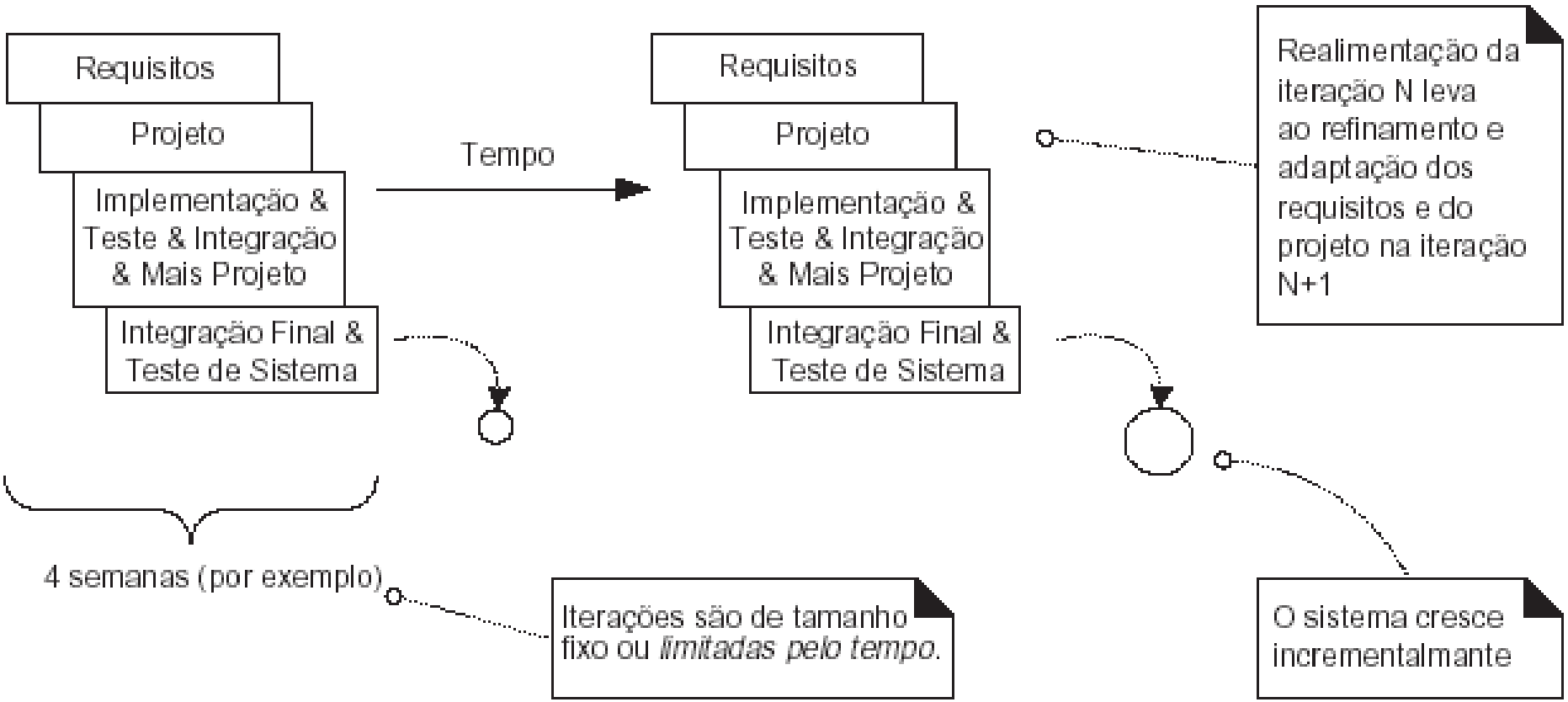


Figura extraída de Larman, 2004

Desenvolvimento Iterativo

- Planejar quantos ciclos de desenvolvimento serão necessários para alcançar os objetivos do sistema
- As partes mais importantes devem ser priorizadas e alocadas nos primeiros ciclos
  - a primeira iteração estabeleça os principais riscos e o escopo inicial do projeto, de acordo com a funcionalidade principal do sistema.
  - partes mais complexas do sistema devem ser atacadas já no primeiro ciclo, pois são elas que apresentam maior risco de inviabilizar o projeto.

Desenvolvimento Iterativo

- O tamanho de cada ciclo pode variar de uma empresa para outra e conforme o tamanho do sistema.
  - Por exemplo, uma empresa pode desejar ciclos de 4 semanas, outra pode preferir 3 meses
- Produtos entregues em um ciclo podem ser colocados imediatamente em operação, mas podem vir a ser substituídos por outros produtos mais completos em ciclos posteriores.

Baseado em Casos de Uso

- Um caso de uso é uma seqüência de ações, executadas por um ou mais atores e pelo próprio sistema, que produz um ou mais resultados de valor para um ou mais atores.
- O PU é dirigido por casos de uso, pois os utiliza para dirigir todo o trabalho de desenvolvimento, desde a captação inicial e negociação dos requisitos até a aceitação do código (testes).

Baseado em Casos de Uso

- Os casos de uso são centrais ao PU e outros métodos iterativos, pois:
  - Os requisitos funcionais são registrados preferencialmente por meio deles;
  - Eles ajudam a planejar as iterações;
  - Eles podem conduzir o projeto;
  - O teste é baseado neles.

Centrado na Arquitetura

- Arquitetura é a organização fundamental do sistema como um todo Inclui elementos estáticos, dinâmicos, o modo como trabalham juntos e o estilo arquitetônico total que guia a organização do sistema.
- A arquitetura também se refere a questões como desempenho, escalabilidade, reuso e restrições econômicas e tecnológicas.

Centrado na Arquitetura

- No PU, a arquitetura do sistema em construção é o alicerce fundamental sobre o qual ele se erguerá
- Deve ser uma das preocupações da equipe de projeto
- A arquitetura, juntamente com os casos de uso, deve orientar a exploração de todos os aspectos do sistema

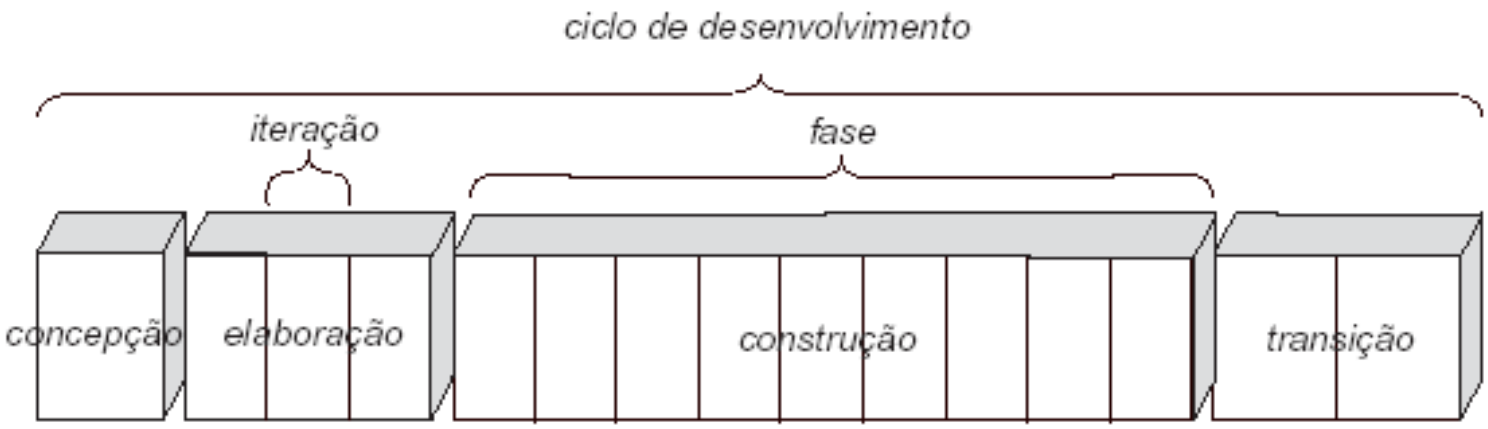
Centrado na Arquitetura

- A arquitetura é importante porque:
  - Ajuda a entender a visão global
  - Ajuda a organizar o esforço de desenvolvimento
  - Facilita as possibilidades de reuso
  - Facilita a evolução do sistema
  - Guia a seleção e exploração dos casos de uso

As Fases do PU

- Cada uma das iterações de desenvolvimento do PU é dividida em quatro fases
  - Concepção
  - Elaboração
  - Construção
  - Transição

As Fases do PU



marco de referência

Um ponto de término de uma iteração no qual ocorre alguma decisão ou avaliação significativa.

versão

Um subconjunto estável executável do produto final. O fim de cada iteração é uma entrega secundária.

incremento

É a diferença (delta) entre as entregas de duas iterações subseqüentes.

entrega final para a produção

Nesse ponto, o sistema é entregue para uso na produção.

Figura extraída de Larman, 2004

Fases do PU: Concepção

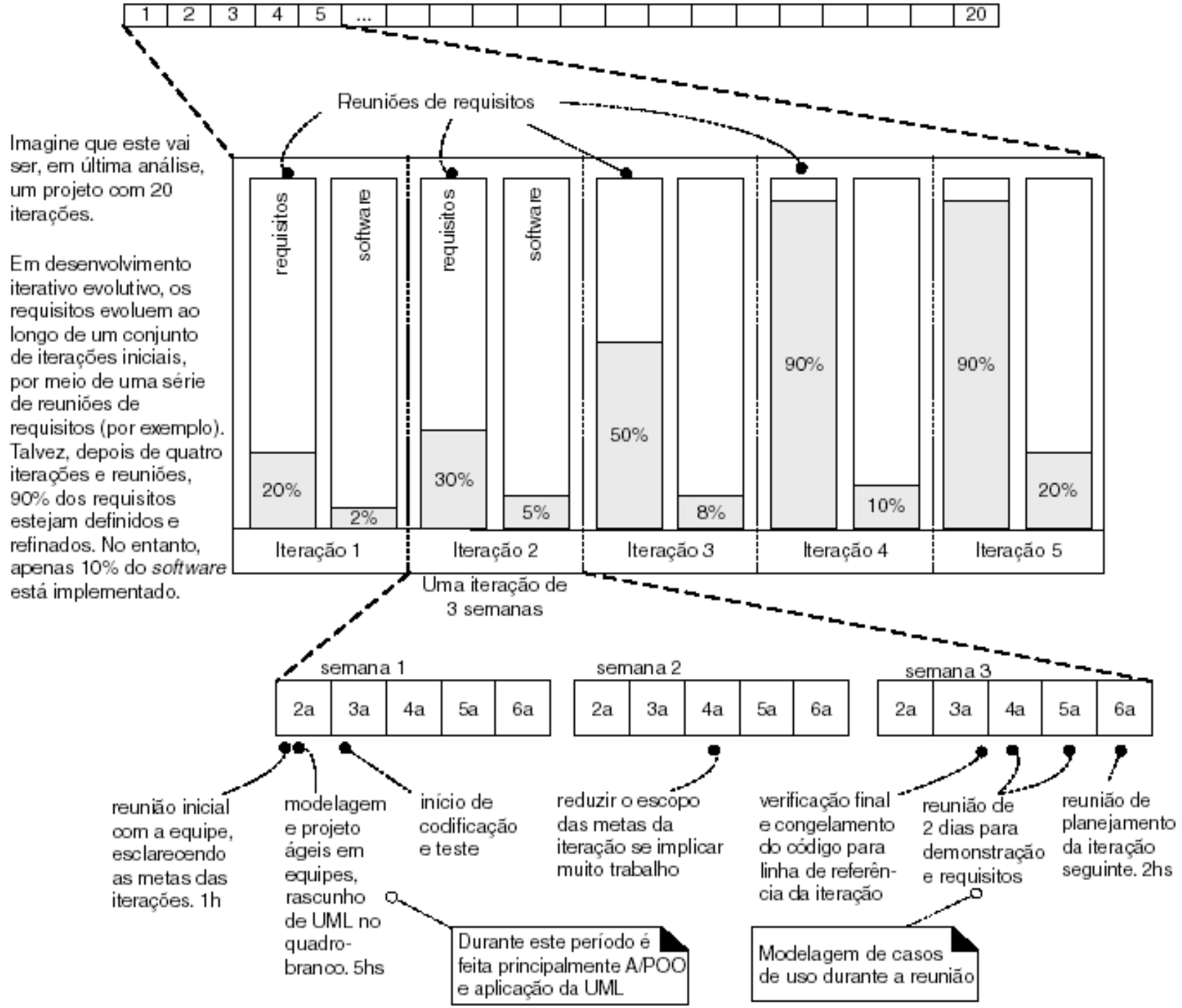
- Estabelece-se a viabilidade de implantação do sistema.
- Definição do escopo do sistema.
- Estimativas de custos e cronograma.
- Identificação dos potenciais riscos que devem ser gerenciados ao longo do projeto.
- Esboço da arquitetura do sistema, que servirá como alicerce para a sua construção.

Fases do PU: Elaboração

- Visão refinada do sistema, com a definição dos requisitos funcionais, detalhamento da arquitetura criada na fase anterior e gerenciamento contínuo dos riscos envolvidos.
- Estimativas realistas feitas nesta fase permitem preparar um plano para orientar a construção do sistema.

Fases do PU: Construção

- O sistema é efetivamente desenvolvido e, em geral, tem condições de ser operado, mesmo que em ambiente de teste, pelos clientes.


1 2 3 4 5 ... 20
Reuniões de requisitos
Imagine que este vai ser, em última análise, um projeto com 20 iterações.
Em desenvolvimento iterativo evolutivo, os requisitos evoluem ao longo de um conjunto de iterações iniciais, por meio de uma série de reuniões de requisitos (por exemplo). Talvez, depois de quatro iterações e reuniões, 90% dos requisitos estejam definidos e refinados. No entanto, apenas 10% do *software* está implementado.
requisitos
software
20%
2%
30%
5%
50%
8%
90%
10%
90%
20%
Iteração 1
Iteração 2
Iteração 3
Iteração 4
Iteração 5
Uma iteração de 3 semanas
semana 1
semana 2
semana 3
2a 3a 4a 5a 6a
reunião inicial com a equipe, esclarecendo as metas das iterações. 1h
modelagem e projeto ágeis em equipes, rascunho de UML no quadro-branco. 5hs
início de codificação e teste
reduzir o escopo das metas da iteração se implicar muito trabalho
verificação final e congelamento do código para linha de referência da iteração
reunião de 2 dias para demonstração e requisitos
reunião de planejamento da iteração seguinte. 2hs
Durante este período é feita principalmente A/POO e aplicação da UML
Modelagem de casos de uso durante a reunião

Fases do PU: Transição

- O sistema é entregue ao cliente para uso em produção.
- Testes são realizados e um ou mais incrementos do sistema são implantados.
- Defeitos são corrigidos, se necessário.

As Disciplinas do PU

- Paralelamente às fases do PU, atividades de trabalho, denominadas disciplinas do PU, são realizadas a qualquer momento durante o ciclo de desenvolvimento
- As disciplinas entrecortam todas as fases do PU, podendo ter maior ênfase durante certas fases e menor ênfase em outras, mas podendo ocorrer em qualquer uma delas

As Disciplinas do PU

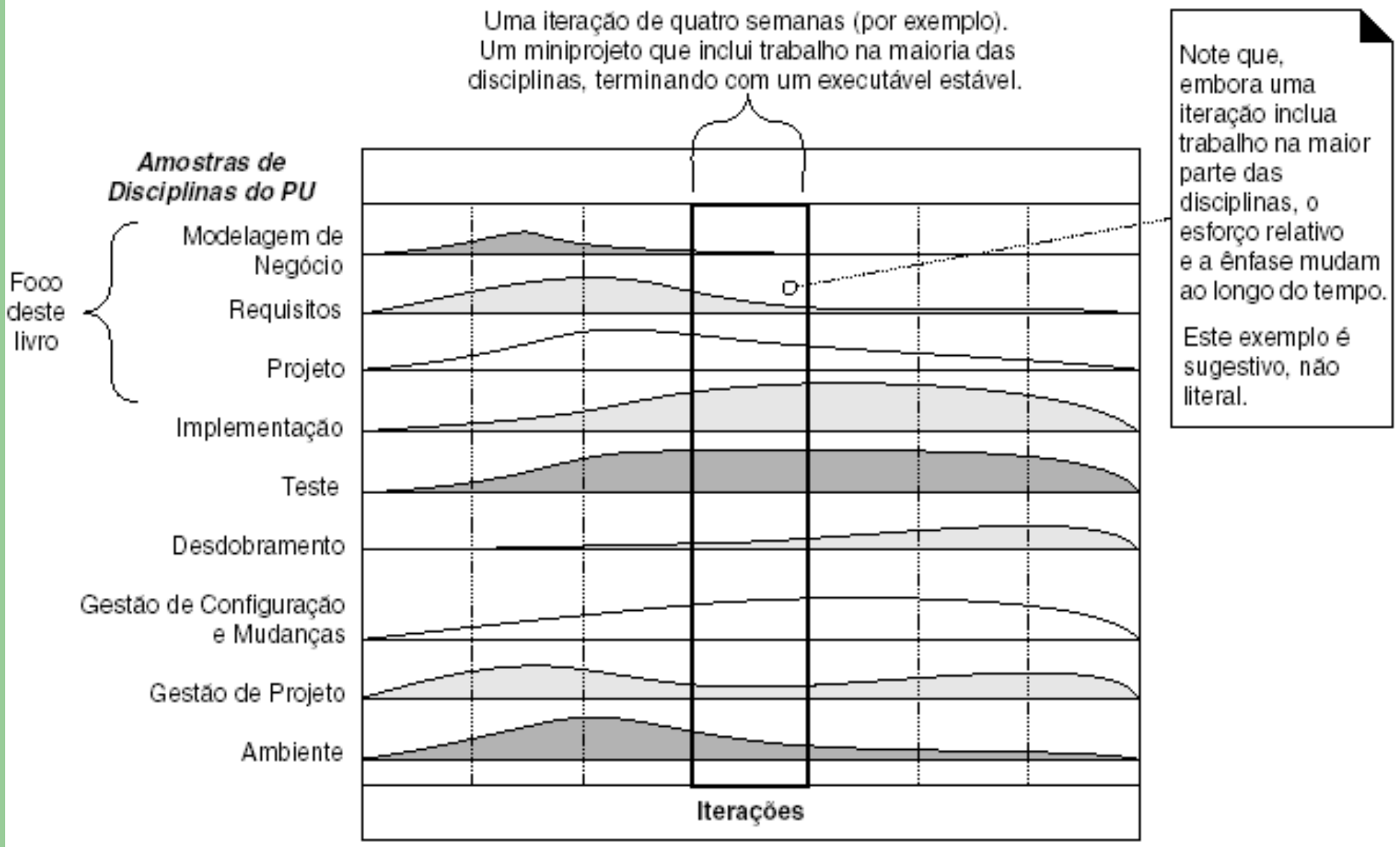
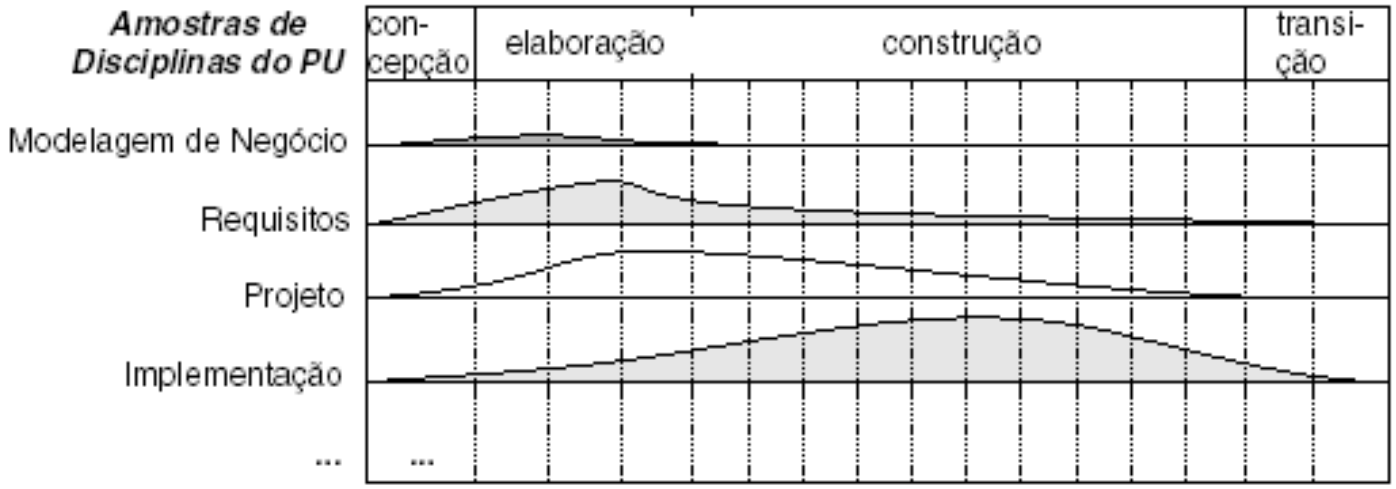


Figura extraída de Larman, 2006

As Disciplinas do PU



O esforço relativo nas disciplinas muda ao longo das fases.

Este exemplo é uma sugestão e não deve ser tomado "ao pé da letra".

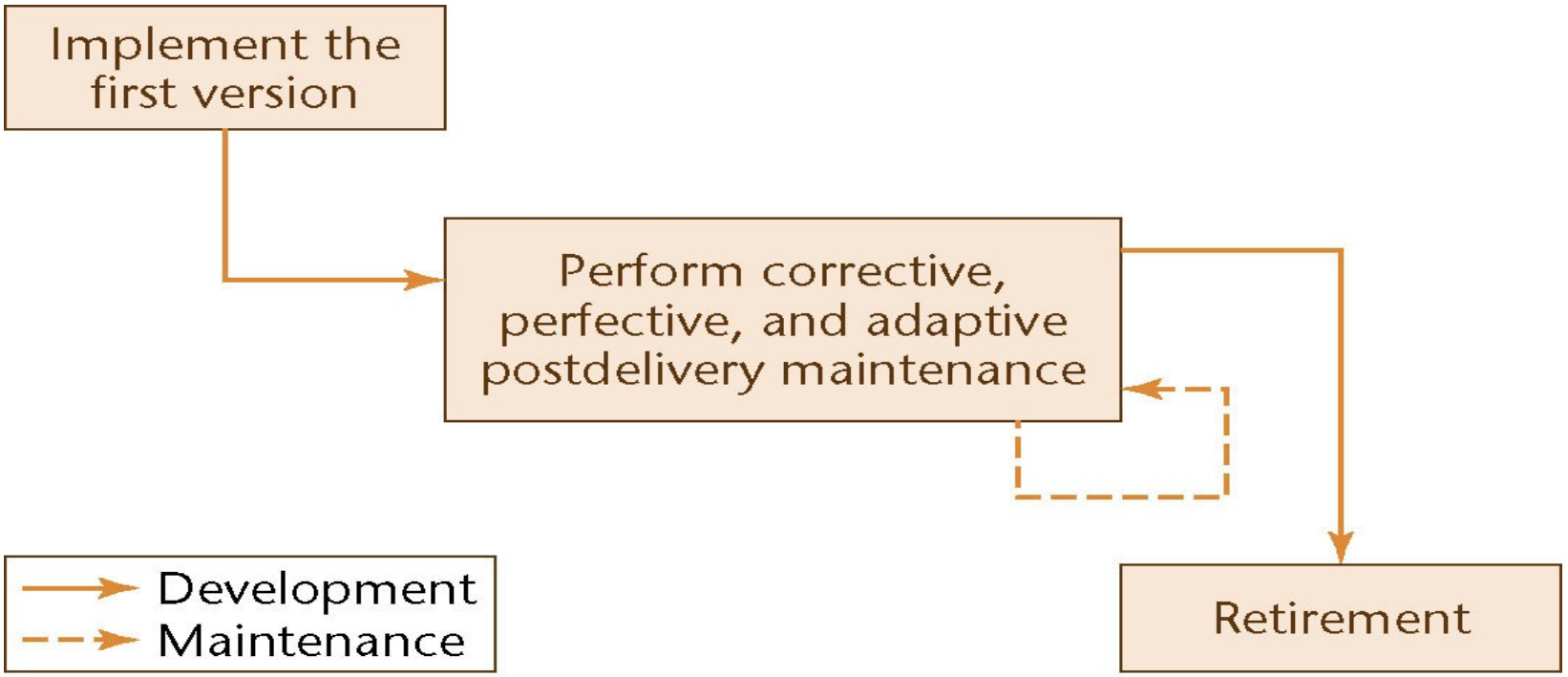
Figura extraída de Larman, 2006

Os Artefatos do PU

- Cada uma das disciplinas do PU pode gerar um ou mais artefatos, que devem ser controlados e administrados corretamente durante o desenvolvimento do sistema
- Artefatos são quaisquer dos documentos produzidos durante o desenvolvimento, tais como modelos, diagramas, documento de especificação de requisitos, código fonte ou executável, planos de teste, etc.
- Muitos dos artefatos são opcionais, produzidos de acordo com as necessidades específicas de cada projeto

Métodos/Processos ágeis

Ciclo de Vida Open-Source



Modelo de Manutenção Pós-entrega

Ciclo de Vida Open-Source

- Primeira fase informal:
  - Um ou mais indivíduos constroem um versão inicial e a deixa disponível na internet (e.g., sourceforge.net).
  - Se houver interesse suficiente no projeto, a versão inicial é baixada por um grande número de interessados; usuários tornam-se desenvolvedores; e o produto é estendido.
- Ponto chave:
  - Indivíduos geralmente trabalham voluntariamente no seu tempo livre.

Ciclo de Vida Open-Source

- A versão operacional inicial é produzida utilizando “Codifique & conserte”, prototipação rápida ou qualquer outro processo.
- A versão inicial é utilizada, usuários relatam bugs, alguns usuários se tornam desenvolvedores e novas versões são geradas.
- Não há especificação ou projeto formal. Por quê ?

Ciclo de Vida Open-Source

- É um modelo de aplicabilidade restrita:
  - Bom para projetos de infraestrutura:
    - SO: Linux, OpenBSD;
    - Browsers: Firefox, Opera;
    - Compiladores: gcc;
    - Web servers: Apache;
    - SGBDs: mySQL, PostGres.
- O modelo de ciclo de vida open-source é aplicável apenas se o produto for utilizado por uma large base de usuários.

Open-Source vs. Closed-Source

- Software de código fechado é mantido e testado por empregados.
  - Usuários podem relatar falhas, mas não defeitos – não possuem acesso ao código.
- Software de código aberto é geralmente por voluntários.
  - Usuários encorajados a relatar falhas ou defeitos no código.
  - Desenvolvedores:
    - *Core group* e *peripheral group*.

Open-Source vs. Closed-Source

- *Core group*:
  - Número pequeno de desenvolvedores com habilidade, disposição e tempo para submeter correções.
  - Tem a responsabilidade de manter o projeto; tem autoridade para instalar correções.
- *Peripheral group*:
  - desenvolvedores que submete relatórios de defeitos de tempos e tempos.

Open-Source vs. Closed-Source

- Novas versões de software de código fechado são liberadas periodicamente, por exemplo, uma vez ao ano.
  - Depois de verificadas cuidadosamente por um grupo de garantia de qualidade.
- No software de código aberto, uma nova versão é liberada tão logo esteja pronta.
  - Não há periodicidade fixa – um dia ou 2 anos.
  - O teste antes da liberação é mínimo.
  - Usuários/desenvolvedores testam usando.
  - “Release often and early”.

Sincroniza & estabiliza

- Ciclo de vida da MicroSoft.
- Requisitos:
  - Entrevista vários clientes pontenciais para um pacote e extrai a lista de *features* de maior prioridade para os clientes.
  - Cria as especificações.
- Divide o trabalho em três ou quatro *builds*:
  - *1o. build*: *features* mais críticos.
  - 2o. *build*: próximos *features* mais críticos.
  - Cada *build* é realizado por equipes pequenas em paralelo.

Sincroniza & estabiliza

- Sincroniza **no final do dia**:
  - Junta os componentes parcialmente completados, compila, testa e depura o produto resultante.
- Estabiliza **no final de cada *build***:
  - Corrige os defeitos restantes e não faz mais alterações nas especificações.
- Sincronização garante que os vários componentes sempre trabalham juntos sem defeitos.
- Versões parciais permitem os desenvolvedores conhecerem a operação do produto e mudar a especificação durante o *build*.

Extremme Programming

- A matéria-prima de XP são:
  - Quatro valores: comunicação, simplicidade, realimentação e coragem.
  - Os princípios;
  - As quatro atividades básicas -- codificação, teste, *listening* e criação de arquitetura.
- Porém, XP é feita mesmo por meio de práticas.

Práticas de XP

- Planejamento;
- Pequenas *releases*;
- Metáfora;
- Arquitetura Simples;
- Teste;
- *Refactoring*;
- Programação em pares;

Práticas de XP

- Propriedade coletiva;
- Integração contínua;
- Semana de 40-horas;
- Cliente residente;
- Padrões de codificação.

Planejamento

- Os conhecedores do negócio (clientes e usuários) devem decidir sobre:
  - Escopo;
  - Prioridade;
  - Composição de *releases*:
    - quanto deve ser feito para que o software passe a ser útil para o negócio.
  - Data de entrega.

Planejamento

- Programadores devem decidir sobre:
  - Estimativas;
  - Conseqüências;
  - Processo a ser utilizado:
    - definição do processo; ajuste à cultura, porém, este ajuste não deve ser prejudicial ao objetivo de produzir software;
  - Cronograma detalhado.

Pequenas *releases*

- Toda *release* deve ser a menor possível, contendo os requisitos de negócios mais importantes.
- *Release* deve fazer sentido como um todo.
- É melhor planejar uma *release* para ser entregue em um mês do que em um ano.

Metáfora

- Deve-se escolher uma metáfora que represente a idéia subjacente ao sistema.
- Arquitetura vai ser um reflexo da metáfora.
- Exemplo:
  - Sistema de controle de projetos realizados em unidades de negócios distribuídas.
  - Arquivo central contém projetos contidos em *folders*.

Arquitetura simples

- A arquitetura correta para o software em qualquer momento é aquela que:
  - Permite a execução de todos os casos de teste;
  - Não possui lógica duplicada;
  - Descreve tudo que é importante para os programadores;
  - Possui o menor conjunto de classes e métodos.

Teste

- Um requisito do sistema que não possui um teste automatizado *não* existe.
- Programadores devem escrever teste unitários de forma a ganhar confiança no seu código.
- Clientes devem escrever casos de teste funcionais para obterem confiança no software desenvolvido.
- Confiança no sistema aumenta com teste automatizado.

Refactoring

- Ao implementar uma funcionalidade, o programador deve se perguntar:
  - há uma maneira tornar o programa mais simples para incluir essa funcionalidade e manter os casos de teste funcionando?
- Se houver, então, ao tornar o programa mais simples, está-se realizando *refactoring*.

Programação em Pares

- Todo código é gerado com duas pessoas olhando a mesma máquina, com um teclado e um *mouse.*
- O programador que fica com o teclado e o mouse se preocupa com a melhor forma de implementação.
- O outro tem preocupações mais estratégicas: Essa idéia funciona? Estão faltando casos de teste? Há uma maneira de fazer mais simples?

Programação em Pares

- Atividade de programação é definida no início do dia com uma reunião matinal (*stand-up meeting)*.
- Encontro diário, de pé, em geral em um café da manhã, no qual são discutidos:
  - as atividades de programação,
  - os problemas e soluções encontrados e
  - comunicações gerais para manter o foco do grupo.
- A reunião matinal não deve durar muito, por isso, é realizada de pé.

Propriedade coletiva

- Em XP não existe *propriedade individual* de código.
- Todos programadores têm responsabilidade sobre todo o sistema.
- Nem todo mundo conhece cada parte da mesma maneira, mas todo mundo conhece algo sobre tudo.

Integração Contínua

- O código deve ser integrado e testado depois de algumas horas.
- Uma máquina deve ficar dedicada para integração.
- Um par de programadores, depois de gerar código, deve:
  - carregar a *release* atual,
  - carregar suas modificações e
  - rodar os testes até todos (100%) passarem.

Semana de 40 horas

- As pessoas precisam de descanso para serem produtivas e criativas.
- Horas extras são um sintoma de problemas sérios no projeto.
- XP admite uma semana com horas extras;
- Duas já são uma indicação de problemas de estimativa e gerência do projeto.

Cliente residente

- Um cliente deve estar disponível, no site, para responder dúvidas, disputas e ajustar prioridades menores.
- Este cliente deve ser um usuário real do sistema.
- Apesar de longe do seu site original, o cliente deve continuar a fazer o seu trabalho usual.

Padrões de codificação

- Com todos os programadores podendo *alterar* qualquer parte do código, padrões de codificação é essencial em XP.
- Os padrões devem:
  - exigir pouco esforço, caso contrário, não serão seguidos;
  - evitar duplicação de código;
  - enfatizar comunicação;
  - ser adotados voluntariamente.

Scrum

- Desenvolvimento de um software é um processo que cria um produto *mal definido*.
- Ou seja, a saída do processo não conhecida *a priori*.
- Para resolver isto, a teoria de controle de sistemas exige realimentação constante.
- Scrum procura atingir este objetivo produzindo realimentação constante, mantendo a equipe focada em atingir objetivos curtos.

Elementos do Scrum

- *Product backlog (PB):*
  - Lista de *features* e tarefas prioritárias para o desenvolvimento do sistema.
  - Qualquer pessoa pode inserir um novo *feature* ou tarefa no *product backlog*.
- *Scrum master (SM)*:
  - Responsável por tomar conta do processo.
  - Remove os empecilhos que atravacam o processo.
  - Junto com o *product owner*, define quais os elementos PB serão resolvidos no *Scrum Sprint*.

Elementos do Scrum

- *Product owner (PO):*
  - Definido pela gerência para gerenciar o PB:
    - Gerente de produto, de departamento ou projeto.
  - Deve manter o PB visível.
- *Scrum sprint (SS)*:
  - Esforço de desenvolvimento de um mês no qual a lista de elementos do PB estabelecido como prioritários serão resolvidos no *Scrum Sprint*.
  - No final desse esforço de desenvolvimento, deve-se ter um pedaço de funcionalidade apresentável e funcionando.

Elementos do Scrum

- *Daily scrum (DS):*
  - Reunião diária na qual as dicisões e problemas do projeto são tomadas e discutidas.
  - É gerenciada pelo o SM para eliminar dificuldades e manter a equipe focada no SS.
- *Scrum team (ST)*:
  - A equipe deve se comprometer a atingir o objetivo do SS.
  - A equipe possui autoridade para tomar a decisão que for necessária para atingir o objetivo.

Resumo

- Existem modelos para o processo de desenvolvimento de software.
- Vimos diferentes modelos:
  - Com etapas bem formalizadas e outros nem tanto.
- Não existe um modelo *certo* para o desenvolvimento de um SI qualquer.
- Existe o modelo *certo* para o SI que uma determinada organização quer desenvolver.
- Cabe aos desenvolvedores identificar este modelo para obter os melhores resultados.

Exercícios

- Compare as semelhanças e as diferenças entre os processo ágeis apresentados?
- O PU é um processo ágil? Explique sua resposta.
- Resumo de artigo para entregar:
  - Michael A. Cusumano: Extreme programming compared with Microsoft-style iterative development. Commun. ACM 50(10): 15-18 (2007).

Referências bibliográficas

- Michael A. Cusumano, Richard W. Selby: How Microsoft Builds Software. Commun. ACM 40(6): 53-61 (1997).
- Larman, Craig – Utilizando UML e Padrões, 2a edição, Bookman, 2004.
- Sommerville, I. Engenharia de Software, Addison-Wesley Brasil, 6ª Edição – 2003.
- Pfleeger, S. L. Engenharia de Software: Teoria e Prática, Prentice Hall do Brasil, 2ª Edição, 2004.
- Schach, S. Object-Oriented and Classical Software Engineering, McGraw-Hill Science/Engineering/Math; 6a. edition.

Referências bibliográficas

- Christian Robottom Reis. Caracterização de um Processo de Software para Projetos de Software Livre. Dissertação de Mestrado, ICMC-USP, 2003. (disponível em http://143.107.58.177:8000/projects/list_files/osi)