Processos de Software

Prof. Márcio Lopes Cornélio
Alexandre Vasconcelos

Slides originais elaborados por Ian Sommerville
O autor permite o uso e a modificação dos *slides* para fins didáticos

O processo de software

- Um conjunto estruturado de atividades, procedimentos, artefatos e ferramentas necessários para o desenvolvimento de um sistema de software
  - Exemplo de Atividades: Especificação, Projeto, Validação, Evolução
- Exemplos de processos: Processo Unificado (RUP), OpenUP, Programação Extrema (XP), SCRUM

O processo de software: pontos de consenso

- Iteratividade
- Participação de usuários
- Flexibilidade de configuração para projetos específicos
- Comunicação entre membros da equipe

O processo de software: pontos de divergência

- Nível de detalhamento de atividades a serem seguidas
- Critério de conclusão da execução das atividades
  - Arquitetura robusta (RUP)
  - Arquitetura para o contexto da iteração atual (agile modeling)
- Rigor na atribuição de tarefas a responsáveis
  - workers (RUP)
  - alocação sob demanda e interesse (XP)
- Artefatos (documentação) gerados
- Nível de Automação
- Nível de (im)pessoalidade

O processo de software: polêmica

- Se a tendência é processos leves, afinal o desenvolvimento de software é
  - Arte+Sociologia+Psicologia+...

  ou
  - Lógica+Modelos+Engenharia+...???
- E todo o esforço de consolidação da Engenharia de Software como uma ciência exata???
- Compromisso
  - Balancear agilidade e disciplina

Processo versus Metodologia

- Alguns autores consideram que processos incluem
  - uma metodologia
  - pessoas
  - tecnologia (suporte de ferramentas)
- Outros consideram que uma metodologia é a especialização de um processo com um conjunto de métodos

Método

- Descrição sistemática de como deve-se realizar uma determinada atividade ou tarefa
- A descrição é normalmente feita através de padrões e guias
- Exemplo: Método para descoberta de atores e casos de uso no RUP.

Modelo de Processo

- Um modelo de processo de software apresenta a descrição de um processo de uma perspectiva particular, normalmente focando apenas em algumas atividades.
- Um modelo é usado para entendimento e comunicação do processo, e como base para análise, execução, gerência e melhoria do processo
- Idealmente a descrição deve ser formal e completa para permitir, por exemplo, automação
- A descrição deve ser apresentada em diferentes níveis de abstração

Modelo de Processo

- O formalismo utilizado para representar o processo é o ingrediente mais importante da modelagem
- Não parece haver consenso sobre um formalismo ideal
  - Terminologias distintas – ex: Fase, workflow, disciplina, atividade
  - Mas há esforço de padronização (SPEM e BPMN – OMG)

Modelos genéricos de processo de software

- Representação abstrata e simplificada do processo de desenvolvimento de software
  - Incluindo algumas atividades e sua organização de alto nível
- Exemplos:
  - Modelo cascata
    - Fases separadas e distintas de especificação e desenvolvimento.
  - Desenvolvimento iterativo
    - Sistema desenvolvido através de várias etapas (iterações)
  - Engenharia de software baseada em componentes
    - O sistema é montado a partir de componentes existentes.

Modelos genéricos de processo de software

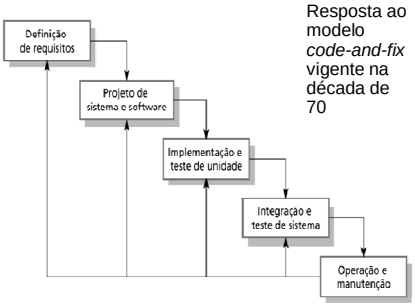
- Dois modelos não são necessariamente mutuamente excludentes!
  - Ex: desenvolvimento formal, onde um processo semelhante ao cascata é usado, mas a especificação formal é refinada durante os vários estágios para um projeto implementável.

Modelo Cascata (ou clássico)

- Primeiro modelo a organizar as atividades de desenvolvimento
- Derivado de modelos existentes em outras engenharias
- Uma fase tem de estar completa antes de passar para a próxima.
  - Saídas das fases são acordadas contratualmente!
- Sua estrutura é composta por várias fases que são executadas de forma sistemática e seqüencial
  - Todas as fases envolvem atividades de validação
- Na prática, existe uma interação entre as etapas e cada etapa pode levar a modificações nas etapas anteriores

Modelo Cascata (ou clássico)

Figura 4.1
Ciclo de vida de software.



Resposta ao modelo *code-and-fix* vigente na década de 70

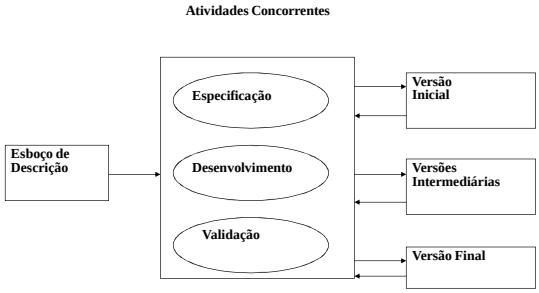
Problemas com o modelo cascata

- Particionamento inflexível do projeto em estágios
  - Dificulta a resposta aos requisitos de mudança do cliente.
- Documentos "completamente elaborados" são necessários para fazer as transições entre estágios
- Apropriado somente quando os requisitos são bem compreendidos e quando as mudanças são raras
  - Poucos sistemas de negócio têm requisitos estáveis.

Modelo de desenvolvimento evolucionário

- Implementação inicial, exposição do resultado aos comentários do usuário e refinamento desse resultado em versões
- Especificação, desenvolvimento e validação são intercalados
- Dois tipos
  - Desenvolvimento exploratório: explora requisitos e entrega um sistema final
  - Prototipação *throwaway*: objetivo é compreender os requisitos do cliente

Modelo de desenvolvimento evolucionário



Desenvolvimento Exploratório

- Idéia geral:
  - Desenvolvimento da primeira versão do sistema o mais rápido possível
  - Modificações sucessivas até que o sistema seja considerado adequado
  - Após o desenvolvimento de cada uma das versões do sistema, ele é mostrado aos usuários para comentários
- Adequado para o desenvolvimento de sistemas onde é difícil ou impossível se fazer uma especificação detalhada do sistema
  - Ex: sistemas especialistas que tentam emular capacidades humanas

Prototipação Descartável

- Como na programação exploratória, a primeira fase prevê o desenvolvimento de um programa para o usuário experimentar
  - No entanto, o objetivo aqui é estabelecer os requisitos do sistema
  - O software deve ser reimplementado na fase seguinte
- A construção de protótipos com os quais os usuários possam brincar é uma idéia bastante atrativa:
  - Para sistemas grandes e complicados
  - Quando não existe um sistema anterior ou um sistema manual que ajude a especificar os requisitos

Prototipação Descartável

- Os objetivos do protótipo devem estar bem claros *antes* do início da codificação. Possíveis objetivos:
  - Entender os requisitos dos usuários
  - Definir a interface com os usuários
  - Demonstrar a viabilidade do sistemas para os gerentes.
- Uma decisão importante a ser tomada é escolher o que será e o que não será parte do protótipo
  - Não é economicamente viável implementar todo o sistema!
  - Os objetivos do protótipo são o ponto de partida

Processos Iterativos

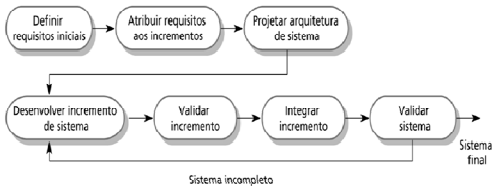
- Requisitos de sistema SEMPRE evoluem no curso de um projeto
- Algum retrabalho é necessário
- A abordagem iterativa pode ser aplicada a qualquer um dos modelos genéricos de processo
- Duas abordagens (relacionadas)
  - Desenvolvimento incremental;
  - Desenvolvimento espiral.
- Essência: especificação desenvolvida em conjunto com o software

Desenvolvimento incremental

- O sistema é entregue ao cliente em incrementos
  - Cada incremento fornece parte da funcionalidade requerida
- Os requisitos são priorizados
  - Requisitos de prioridade mais alta são incluídos nos incrementos iniciais.
- Uma vez que o desenvolvimento de um incremento é iniciado, os requisitos são congelados
  - Os requisitos para os incrementos posteriores podem continuar evoluindo (e incluir requisitos já implementados!)

Desenvolvimento incremental

Figura 4.4 Entrega incremental.



Vantangens do desenvolvimento incremental

- Incrementos podem ser entregues regularmente ao cliente e, desse modo, a funcionalidade de sistema é disponibilizada mais cedo.
- Os incrementos iniciais agem como protótipos para elicitar os requisitos para incrementos posteriores do sistema.
- Riscos menores de falha geral do projeto.
- Os serviços de sistema de mais alta prioridade tendem a receber mais testes.

Problemas do desenvolvimento incremental

- Incrementos pequenos exigem mapeamento entre requisitos e incremento de tamanho adequado
- Dificuldade de identificar os recursos comuns exigidos por todos os incrementos

Desenvolvimento espiral

- Acrescenta aspectos gerenciais ao processo de desenvolvimento de software.
- O processo é representado como uma espiral ao invés de uma seqüência de atividades com realimentação.
- Cada *loop* na espiral representa uma fase no processo.
- Sem fases definidas, tais como especificação ou projeto – os *loops* na espiral são escolhidos dependendo do que é requisitado.
- Os riscos são explicitamente avaliados e resolvidos ao longo do processo.

Modelo espiral do processo de software

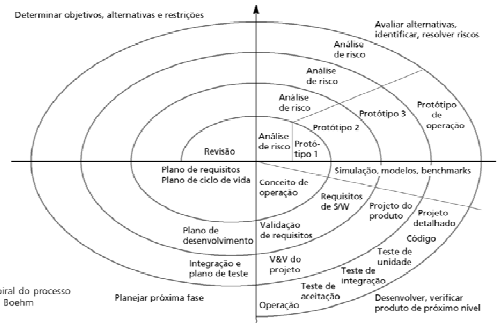


Figura 4.5
Modelo em espiral do processo de software de Boehm (©IEEE,1988).

Quadrantes do modelo espiral

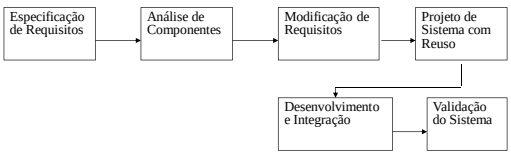
- Processo de Desenvolvimento vs. Processo de Gerenciamento
  - Definição de objetivos
    - Objetivos específicos para a fase são identificados.
  - Avaliação e redução de riscos
    - Riscos são avaliados e atividades são realizadas para reduzir os riscos-chave.
  - Desenvolvimento e validação
    - Um modelo de desenvolvimento para o sistema, que pode ser qualquer um dos modelos genéricos, é escolhido.
  - Planejamento
    - O projeto é revisado e a próxima fase da espiral é planejada.

Engenharia de software baseada em componentes

- Baseado em reuso sistemático onde sistemas são integrados a partir de componentes existentes ou de sistemas COTS (*Commercial-of-the-shelf*)
- Estágios do processo
  - Especificação dos requisitos
  - Análise de componentes;
  - Modificação de requisitos;
  - Projeto de sistema com reuso;
  - Desenvolvimento e integração.
  - Validação

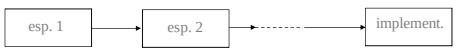
- Esta abordagem está se tornando cada vez mais usada à medida que padrões de componentes têm surgido.
- Reuso acidental vs. Reuso planejado

Engenharia de software baseada em componentes



Transformação Formal

- Idéia geral:
  - Uma especificação formal (definição matemática, não ambígua) do software é desenvolvida e posteriormente "transformada" em um programa através de regras que preservam a corretude da especificação



Transformação Formal

- A grande motivação por trás da idéia de refinamento formal é a possibilidade de gerar programas que são corretos por construção
  - O próprio processo de desenvolvimento deve garantir que o programa faz exatamente o que foi especificado
- Este modelo tem sido aplicado ao desenvolvimento de sistemas críticos, especialmente naqueles onde a segurança é um fator crítico (ex: sistema de controle de ferrovias)