ENGENHARIA DE SOFTWARE

Introdução à Engenharia de Software

Prof. Sérgio Aragão

Novembro/2005

Contexto

- A economia de todos os países do mundo depende cada vez mais de software.
- "O software é um fator que diferencia".
- "A terceira onda de mudança" na história humana. [Toffler]
- "Sociedade da Informação" [Naisbitt]
- "Conhecimento é Poder e o computador é um amplificador desse poder" [Feigenbaum]
- Os gastos com desenvolvimento de software representam uma fração significativa do PIB de muitos países.

Contexto: Primórdios

- O desenvolvimento de software era puramente artesanal;
- As pessoas desenvolvendo sistemas erravam constantemente nas suas estimativas de custo e tempo;
- Os sistemas continham muitos erros;
- Consertar erros → produzia mais erros;
- Tamanho dos sistemas crescendo.
- Complexidade das aplicações aumentando.

Contexto: Histórico

- Estudo feito em 1979 pelo governo dos Estados Unidos em relação ao software produzido:
  - 2% Funcionava;
  - 3% Funcionaria com poucas correções;
  - 20% Usados mas bastante modificados ou abandonados;
  - 45% Entregues mas nunca foram usados com sucesso;
  - 30% Pagos mas nunca foram terminados e/ou entregues.

Contexto: Situação Atual

- Custo de desenvolvimento e manutenção de software é muito alto
  - Na média os projetos de software custam 50% mais do que planejado
- Qualidade de software é insuficiente
  - 75% dos sistemas não funcionam como planejado ou nunca serão usados
  - 57% dos sistemas são entregues sabendo que eles têm defeitos

Contexto: Situação Atual

- Duração de projetos de software é muito longa
  - 68% dos projetos não mantém os prazos
- Projetos de software são difíceis de planejar e controlar

Contexto: Evolução

- A evolução do software:

1950-1964	1964-1973	1973-1988	1987-2000
Sistema em batch, aplicações específicas, distribuição limitada	Multiusuário, tempo real, banco de dados, análise e programação estruturada	Sistemas distribuídos, inteligência imbutida, hardware de baixo custo, impacto de consumo	Sistemas de desktop poderosos, orientação a objeto, sistemas especialistas, redes neurais, computação paralela

Contexto: Custos

- A proporção dos custos com software e hardware mudou bastante ao longo do tempo:

Década	Custos com	
	Software	Hardware
50-60	20%	80%
70	50%	50%
80-90	80%	20%

O que é Software?

- Há 20 anos, menos de 1% do público saberia explicar o que é software.
- Hoje, praticamente todo mundo, acha que sabe ...
- Software não é só os programas!
- A documentação necessária para instalar, usar e manter os programas são uma parte integrante do software.

Algumas características do Software

- O software é projetado / desenvolvido, ao invés de manufaturado no sentido clássico.
- Software não se desgasta com o uso: não existem peças de reposição.
- Software não pode ser visto ou tocado: para analisar o progresso de um projeto de software é preciso recorrer à sua documentação.
- Grandes sistemas de software são normalmente desenvolvidos uma única vez. Assim, a experiência adquirida com outros projetos tem um valor limitado.

Algumas características do Software

- A maioria dos softwares é feita sob medida (por encomenda):
  - Produtos Genéricos: sistemas produzidos por uma organização e vendidos a todos os clientes que quiserem comprá-los.
  - Produtos customizados: sistemas que são encomendados e desenvolvidos para um determinado cliente.

Algumas características do Software

- Perspectivas :
  - Fábricas de software – utilização intensa de ferramentas automatizadas
  - Desenvolvimento de Componentes
    - Reusabilidade de componentes com alta qualidade
    - Componentes distribuídos

Áreas de Aplicações de Software

- Software Básico: apoiam outros programas – forte interação com o hardware - compiladores, utilitários, componentes do SO, drivers, etc
- Software de Tempo Real: monitora / analise / controla eventos do mundo real
- Software Comercial: processamento de informações comerciais

Áreas de Aplicações de Software

- Software Científico e de Engenharia: algoritmos de processamento numérico, CAD/CAM
- Software Embutido: residentes em memória *read-only* – controle de funções específicas consumo de combustível, microondas
- Software de Computador Pessoal: planilhas, editores, gráficos, jogos

Áreas de Aplicações de Software

- Software de Inteligência Artificial: algoritmos não-numéricos para resolver problemas complexos – sistemas especialistas ou baseados em conhecimento, redes neurais (reconhecer padrões baseado em experiência passada)
- Software Educativo: auxiliar o aprendizado de um ou mais temas
- Softwares de Apoio à Decisões: EIS - DSS - SIG

O que é Software de qualidade?

- É software que "funciona" (é confiável):
  - Não deve falhar mais do que o especificado na documentação.
- É software que funciona de acordo com a sua especificação:
  - Mesmo software que aparentemente funciona pode não estar satisfazendo a sua especificação.
- É software que é fácil de manter.
  - Código bem escrito e documentação apropriada.

O que é Software de qualidade?

- É software que funciona de maneira eficiente.
  - Software mais eficiente não é necessariamente software que roda mais rápido ou que gasta menos memória/disco.
  - A complexidade do código e o custo também são fatores importantes.
- É software que possui uma boa interface com o usuário:
  - Muitos softwares não funcionam direito porque são difíceis de usar.

A Crise de Software

- O que é esta crise?
- Métodos de desenvolvimento de software existentes não são bons o bastante para o desenvolvimento de software de grande porte.

A Crise de Software: Crise???

- CRISE?
  - Momento decisivo ou crucial.
  - "Aflição crônica"
  - Mas a chamada crise de software já dura quase 30 anos ...
- DOENÇA?
  - No momento não se conhece uma *cura*, apenas paliativos para reduzir a dor" ...

A Crise: Principais Problemas

- As estimativas de prazo e de custo freqüentemente são imprecisas;
- A produtividade das pessoas da área de software não tem acompanhado a demanda por seus serviços;
- A qualidade do software, às vezes, é menos que adequada.

A Crise: Outros Problemas

- Os computadores estão cada vez mais rápidos, sofisticados, e baratos;
- Os softwares estão cada vez maiores e mais sofisticados, e a produtividade não acompanha a demanda;
- Os custos com manutenção são muito altos: para sistemas com uma longa vida, eles são várias vezes maiores do que os custos de desenvolvimento.

A Crise: Outras Dificuldades

- Dedica-se pouco tempo à coleta de dados (requisitos dos clientes):
  - Normalmente apenas um subconjunto das necessidades do cliente são levadas em conta ...
  - Os profissionais estão sempre com muita pressa para começar a programar ...

A Crise: Outras Dificuldades

- A qualidade geralmente é suspeita ...
  - Testes sistemáticos e tecnicamente completos raramente são feitos;
  - A flexibilidade da maioria dos softwares também é bastante limitada;
  - A concorrência com software barato mas sem qualidade, feito por pessoas sem qualificação adequada é grande.

A Crise: Outras Dificuldades

- Não há muito interesse em se *gastar* tempo para se entender mais a respeito de estimativas, produtividade, precisão e eficácia de novos métodos e novas ferramentas, etc.
  - A resistência a mudanças é grande ...

A Crise: Existe Solução?

- Uma abordagem de engenharia de desenvolvimento de software aliada a uma contínua melhoria de técnicas, métodos e ferramentas.
- Mais treinamento e educação:
  - atualmente se investe muito pouco!

Mitos do Software

- Propagam desinformação e confusão.
- No passado eram tomados como verdades absolutas.
- Ainda há resquícios: é difícil mudar hábitos antigos.
- Existem 3 tipos de mitos:
  - Do cliente
  - Administrativos, e
  - Do profissional.

Mitos do Cliente

- Uma declaração geral dos objetivos é suficiente para começar a escrever os programas: podemos preencher os detalhes mais tarde.
  - Uma definição inicial ruim é a principal causa da maioria dos fracassos no desenvolvimento de software.

Mitos do Cliente

- As necessidades do projeto mudam continuamente mas isto não é problema pois o software é flexível.
  - Os requisitos do software podem mudar, mas o custo da mudança varia bastante dependendo de que fase ela ocorre:
    - **Definição 1x**
    - **Desenvolvimento . . . . . 1.5x a 6x**
    - **Manutenção . . . . . 60x a 100x**

Mitos Administrativos

- Temos um manual de padrões e procedimentos para a construção de software: isto basta!
  - O manual é usado?
  - Os profissionais de software sabem que ele existe?
  - Ele reflete as técnicas mais modernas?
  - Ele é completo?

Mitos Administrativos

- Temos ferramentas de desenvolvimento de última geração pois compramos os computadores mais novos!
  - Em geral, ter ferramentas de auxílio ao desenvolvimento de software (ex. CASE) é mais importante do que ter a última geração em termos de hardware.

Mitos Administrativos

- Estamos atrasados no prazo: podemos tirar o atraso colocando mais programadores no projeto.
  - Normalmente isto *não funciona*!
  - As novas pessoas precisam se integrar ao projeto ...

Mitos do Profissional

- Assim que escrevermos o programa e ele *funcionar* o nosso trabalho estará terminado.
  - Em geral, de 50 a 70% de todo o esforço gasto num programa ou sistema ocorre *depois* que ele foi *entregue* ao cliente (manutenção).
  - Na maioria das vezes, quanto mais cedo se começa a escrever o código mais tempo se gastará para terminá-lo.

Mitos do Profissional

- Enquanto o programa não estiver funcionando não há como avaliar a sua qualidade.
  - Revisões técnicas podem ser feitas desde o começo de um projeto e são uma das formas mais efetivas de garantia de qualidade de software.

Mitos do Profissional

- A única coisa a ser entregue em um projeto bem sucedido é o programa *funcionando*.
  - O programa funcionando é só uma parte.
  - Uma boa documentação incluindo os requisitos, projeto da estrutura de dados, especificação de testes, etc. é o alicerce para um projeto bem sucedido e serve como guia de manutenção.

Definições de Engenharia de Software

- É o estabelecimento e uso de sólidos princípios de engenharia visando obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais. [Fritz Bauer]
  - Engenharia de Software trata do desenvolvimento de software de forma eficaz.

Definições de Engenharia de Software

- É uma disciplina que se preocupa com os problemas práticos inerentes ao desenvolvimento de sistemas de grande porte.
  - Não é simplesmente programação;
  - Também não é só ciência da computação;
  - Uso de métodos, ferramentas e procedimentos na resolução de problemas – elementos fundamentais segundo Pressman.

Elementos Fundamentais da ES

- Métodos - proporcionam os detalhes de "como fazer" para construir o software:
  - Planejamento e estimativas de projeto
  - Análise de requisitos
  - Projeto da estrutura de dados
  - Arquitetura de programa e algoritmo
  - Codificação
  - Testes
  - Manutenção

Elementos Fundamentais da ES

- Ferramentas: proporcionam apoio automatizado aos métodos:
  - Ferramentas CASE – Computer-Aided Software Engineering – ambiente integrado com banco de dados onde são armazenados os metadados

Elementos Fundamentais da ES

- Procedimentos: definem a seqüência em que os métodos serão aplicados, os produtos gerados(documentos), os controles que ajudam a assegurar a qualidade e os marcos de referência que possibilitam aos gerentes avaliar o progresso.

Características da Engenharia de Software

- Engenharia de Software se refere a software (aplicativos) desenvolvidos por grupos ao invés de indivíduos;
- Engenharia de Software usa princípios de engenharia ao invés de arte, e
- Engenharia de Software inclui tanto aspectos técnicos quanto não técnicos.

Características da Engenharia de Software

- O principal objetivo da Engenharia de Software é produzir, a um custo baixo, software de qualidade.
  - Custo é fácil de ser medido.
  - Qualidade não é.
- O processo de planejamento é crucial na engenharia de software. A implementação é só uma parte do processo.
- A Engenharia de Software engloba todo o ciclo de vida do software (concepção, implementação, uso e manutenção).

Modelos de Ciclo de Vida de Software

- Se referem à progressão dos projetos de software, do desenvolvimento e manutenção e eventualmente a sua substituição;
- Descrições abstratas do processo de desenvolvimento de software, mostrando as atividades e dados usados no ciclo de vida do software;
- São conseqüência direta da crise de software e da necessidade de se ver o processo de desenvolvimento de software como uma engenharia.

O Processo de Software

- Um conjunto estruturado de atividades necessárias para o desenvolvimento de um sistema de software.
  - Especificação.
  - Projeto.
  - Construção
  - Validação.
  - Evolução.
- Atividades variam com a organização e o tipo de sistema sendo desenvolvido.

Paradigmas da Engenharia de Software

- A estratégia usada no desenvolvimento do software deve definir etapas que envolvem métodos, ferramentas e procedimentos.
- Uma estratégia de desenvolvimento é um modelo de processo ou paradigma de engenharia de software.

Paradigmas da Engenharia de Software

- A escolha da estratégia deve considerar:
  - Natureza do projeto
  - Tipo da aplicação
  - Métodos e ferramentas que serão usados
  - Métodos de controle
  - Prazo de entrega
  - Produtos que serão entregues

Modelos de Ciclo de Vida de Software

- Principais modelos:
  - Modelo clássico (ou *em cascata*);
  - Prototipagem (ou Prototipação);
  - Modelo espiral (ou baseado em riscos);
  - Modelo voltado para o Reuso de Componentes.

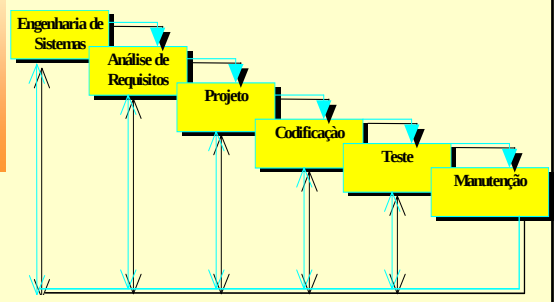
Modelo Clássico (ou *em cascata*)

- Derivado de modelos existentes de outras engenharias (1970);
- Fornece uma estrutura para o processo de desenvolvimento de software;
- Sua estrutura é composta de várias fases que são executadas de forma sistemática e seqüencial;
- Na prática, existe uma interação entre as fases e cada fase pode levar a modificações nas fases anteriores;
- Este é o modelo mais antigo mas ainda o mais usado.

Modelo Clássico: Fases

- Existem inúmeras variações deste modelo e algumas delas incluem novas fases no processo.
- Fases mais comuns:
  - **Análise de viabilidade e engenharia de sistemas**
  - **Análise de requisitos de software.**
  - **Projeto (*design*).**
  - **Codificação (Implementação)**
  - **Testes de unidades, integração e teste do sistema.**
  - **Implantação e**
  - **Manutenção.**

Modelo Clássico: Fases



Modelo Clássico: Vantagens

- Os gerentes de projetos de software aceitaram o modelo entusiasticamente porque:
  - Oferece uma maneira de tornar o processo mais visível.
  - Facilita o planejamento.
  - Fixa pontos específicos para a escrita de relatórios.

Modelo Clássico: Problemas

- Projetos reais raramente seguem o fluxo seqüencial proposto por este modelo: na maioria dos casos existe interação e superposição.
- Raramente os clientes (usuários) declaram todas as exigências de uma vez, no início do projeto.
- Boa parte dos programas não estará disponível até um ponto adiantado no cronograma do projeto: é geralmente difícil convencer o usuário de que é preciso paciência

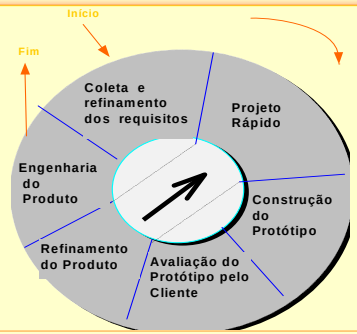
Prototipagem

- 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.

Prototipagem

- Construção de modelos concretos destinados a testes e avaliações prévias de sistemas de informações.
- Adequado para o desenvolvimento de sistemas onde é difícil ou impossível de se fazer uma especificação detalhada do sistema;

Prototipagem



Prototipagem

Tipos de Prototipação:

- **Descoberta ou descartável**
  - Descobrir as necessidades do usuário e depois é descartado
- **Evolutiva ou de refinamento**
  - Desenvolvimento de um sistema que é progressivamente aperfeiçoado até que seja plenamente aceito pelo usuário.

Prototipagem Evolutiva

- Em geral este tipo de desenvolvimento exige ferramentas de alto nível e máquinas poderosas e dedicadas;
- A maioria dos sistemas desenvolvidos com sucesso usando a prototipagem de refinamento foi implementada usando pequenos grupos de profissionais altamente qualificados e motivados;
- É raramente usada no desenvolvimento de sistemas de grande porte e de vida longa.

Prototipagem Evolutiva

Principais razões:

- Mudanças contínuas tendem a produzir sistemas cuja estrutura é desorganizada. Como conseqüência, a manutenção tende a ser mais difícil e cara.
- O gerenciamento de projetos de software normalmente se baseia em modelos nos quais deve-se produzir relatórios regulares, usados para avaliar o progresso do projeto.
  - **Como na prototipação evolutiva o sistema é modificado com freqüência, não é razoável produzir muita documentação.**

Prototipagem Descartável

- Como na prototipagem de refinamento, a primeira fase prevê o desenvolvimento de um *programa* para o usuário experimentar.
  - No entanto, o objetivo aqui é estabelecer os requerimentos 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 requerimentos.

Prototipagem Descartável

- Os objetivos do protótipo devem estar bem claros *antes* do início da codificação.
- Possíveis objetivos:
  - Entender os requerimentos dos usuários.
  - Definir a interface com os usuários.
  - Demonstrar a viabilidade do sistema 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.

Prototipagem Descartável: o que incluir no protótipo?

Algumas possibilidades:

- Implementar todas as funções do sistema mas com um número reduzido de detalhes.
- Implementar um subconjunto das funções, possivelmente com um número maior de detalhes.
- Desconsiderar requerimentos associados a velocidade, espaço, confiabilidade, etc.
- A menos que o objetivo do protótipo seja definir a interface com o usuário, desconsiderar a parte de manipulação de erros.

Prototipagem: possíveis vantagens

- Protótipos contribuem para melhorar a qualidade da especificação dos futuros programas, o que leva à diminuição dos gastos com manutenção.
- Em alguns casos, o treinamento dos usuários pode até ser feito *antes* do *produto* ficar pronto.
- Algumas partes do protótipo podem vir a ser usadas no desenvolvimento do sistema final.

Prototipagem: possíveis desvantagens

Em geral o grande argumento contra a construção de protótipos é o custo.

- A construção do protótipo atrasa o início da implementação do sistema final:
  - Atrasos são um dos maiores problemas dos projetos de software.
  - Construir um protótipo pode não ser tão mais rápido assim do que construir o sistema final.
  - Se os ambientes utilizados forem diferentes este será um custo extra.

Prototipagem: possíveis desvantagens

- O cliente vê algo que parece ser uma versão do software desejado e não entende porque o produto precisa ser reconstruído.
  - A tendência é o cliente *exigir* que *pequenos acertos* sejam feitos para que o protótipo se transforme no sistema final.
  - Freqüentemente a gerência cede ...
- Muitas das *concessões* feitas na implementação do protótipo visando a construção rápida podem vir a fazer parte do sistema final.
  - Utilização de linguagens, ferramentas, algoritmos, etc. que sejam inadequados e/ou ineficientes.

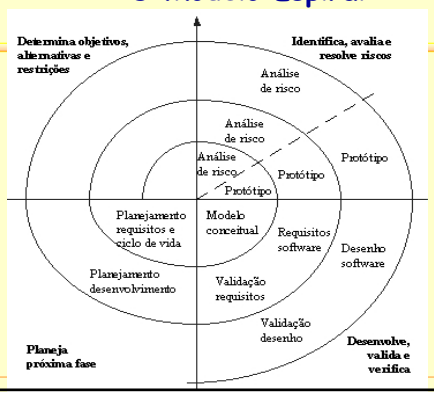
O Modelo Espiral

- O modelo em espiral foi proposto por Boehm em 1988 como forma de integrar os diversos modelos existentes à época, eliminando suas dificuldades e explorando seus pontos fortes.
- Acrescenta aspectos gerenciais ao processo de desenvolvimento de software.
  - análise de riscos em intervalos regulares do processo de desenvolvimento de software;
  - planejamento;
  - controle;
  - tomada de decisão.

O Modelo Espiral

- Prevê quatro fases:
  - Planejamento: determinar objetivos, alternativas, e restrições.
  - Análise de Riscos: análise das alternativas e identificação e resolução dos riscos.
  - Projeto ou Engenharia: desenvolvimento e implementação da próxima versão.
  - Avaliação dos resultados pelo cliente.
- Estas quatro fases são repetidas várias vezes até que o produto final seja satisfatório pelo usuário.

O Modelo Espiral



O Modelo Espiral

- Observações:
  - A cada ciclo da espiral, versões progressivamente mais completas do software são construídas;
  - Antes de cada ciclo, uma análise de riscos é feita;
  - Ao fim de cada ciclo é feita uma avaliação se deve-se prosseguir para o próximo ciclo.

O Modelo Espiral: O Risco

- O que é risco?
  - Difícil de se definir precisamente!
  - Qualquer coisa que possa sair errado.
  - Conseqüência de informação inadequada.
  - O risco de uma atividade é a medida de incerteza do resultado desta atividade.
  - O risco está associado com a quantidade de informação disponível: quanto menos informação, maior o risco.

O Modelo Espiral: O Risco

- Riscos são resolvidos por ações que descubram ou gerem informações que *reduzam o grau de incerteza.*
- Há quem defenda que a tarefa principal dos gerentes de projetos de software é a minimização dos riscos.

O Modelo Espiral: méritos e deficiências

Modelo não foi muito utilizado ainda: é difícil julgar ...

- Foca atenção nas opções de reuso e na eliminação de erros cedo.
- É difícil convencer gerentes de que todo este processo é controlável.
- É preciso experiência na avaliação dos riscos.
- Precisa de refinamento para uso geral.

Modelos voltados para o Reuso de Componentes

- Desenvolvimento de sistemas utilizando componentes de software que já foram testados e aprovados
- Estes componentes podem ser:
  - Classes de objetos
  - Módulos
  - Run-time
  - Subsistemas e até
  - Sistemas completos (ex: sistemas legados)

Modelos voltados para o Reuso de Componentes

- Tenta maximizar o processo de desenvolvimento de sistemas pela reutilização de componentes de software já prontos, melhorando a produtividade e a qualidade do produto gerado

Modelos voltados para o Reuso de Componentes

- Seqüência de ações:
  - Esboçar os requisitos do sistema
  - Procurar componentes reutilizáveis
  - Modificar os requisitos de acordo com os componentes encontrados
  - Projeto de arquitetura
  - Procurar componentes realizados
  - Projetar o sistema

Modelos voltados para o Reuso de Componentes

- Benefícios:
  - Maior confiabilidade
  - Redução dos riscos do processo
  - Uso efetivo de especialistas
  - Conformidade com padrões (interface com o usuário)
  - Desenvolvimento acelerado

Modelos voltados para o Reuso de Componentes

- Problemas:
  - Aumento nos custos de manutenção quando o código fonte do componente não está disponível
  - Falta de ferramentas de apoio (CASE não apropriado)
  - Síndrome do “não foi inventado aqui” (pensam que podem fazer melhor)
  - Manutenção de biblioteca de componentes
  - Encontrar e adaptar componentes reutilizáveis