

1 SOFTWARE

“O mundo precisa de software.” [Steve Jobs, criador do Apple II]

“Quando um software de computador é bem-sucedido – quando satisfaz às necessidades das pessoas que o usam, tem desempenho sem falhas por um longo período, é fácil de modificar e ainda mais fácil de usar – ele pode e efetivamente modifica as coisas para melhor. Mas, quando o software falha – quando seus usuários ficam insatisfeitos, quando tem tendências a erros, quando é difícil de modificar e ainda mais difícil de usar – podem e efetivamente acontecem coisas desagradáveis. Todos nós desejamos construir softwares que tornem as coisas melhores, evitando os problemas que espertam na sombra dos esforços malsucedidos. Para obter sucesso, precisamos de disciplina quando o software é projetado e construído. Precisamos de uma abordagem de engenharia.” [Roger Pressman, 2006]

As citações acima destacam a importância que o software adquiriu na chamada **Sociedade da Informação**. Mas, como destacam Roger Pressman (2006) nem todos o defendem, também existem autores que lançaram uma série de obras consideradas “anti-computadores”. Para exemplificar essa situação pode ser dada a seguinte citação, de Andy Rooney: “Os computadores tornam mais fácil fazer uma série de coisas, mas a maior parte das coisas que eles facilitam não precisa ser feita.”

1.1 PANORAMA SOBRE SOFTWARE

Origem: Engenharia de Software, Roger Pressman, 2006

- **O que é?** Software de computador é o produto que os profissionais de software constroem, e depois, mantêm ao longo do tempo. Abrange programas que executam em computadores de qualquer tamanho e arquitetura, conteúdo que é apresentado ao programa a ser executado e documentos tanto em forma impressa quanto virtual, que combinam todas as formas de mídia eletrônica;
- **Quem faz?** Engenheiros de software constroem e mantêm, e praticamente, todas as pessoas do mundo industrializado usam direta ou indiretamente;
- **Por que é importante?** Porque afeta praticamente todos os aspectos de nossas vidas e tornou-se difundido no nosso comércio, na nossa cultura e nas nossas atividades do dia-a-dia;
- **Quais são os passos?** Você constrói software de computadores como constrói qualquer produto bem-sucedido, aplicando um processo ágil e adaptável que leva a um resultado de alta qualidade e que satisfaz às necessidades das pessoas que vão usar o produto. Você aplica uma abordagem de engenharia de software;
- **Qual é o produto do trabalho?** Do ponto de vista do engenheiro de software, o produto do trabalho são os programas, o conteúdo (os dados) e documentos que compõem um software de computador. Mas, do ponto de vista do usuário, o produto do trabalho é a informação resultante que, de algum modo, torna melhor o mundo do usuário.

1.2 SISTEMA BASEADO EM COMPUTADOR

Origem: Engenharia de Software, Notas de Aula, prof. Denise Franzotti Togneri, FAESA

Os sistemas podem ser divididos em sistemas naturais, tais como o sistema solar e o sistema digestivo, e sistemas construídos pelo homem ou artificiais, tais como um sistema político e um sistema de injeção eletrônica.

Os sistemas artificiais podem ser definidos como um conjunto de componentes interrelacionados, que podem ser vistos como um todo, onde os componentes trabalham juntos na execução de um conjunto de funções para alcançar um propósito. Assim, um sistema artificial possui:

- **Componentes:** partes básicas ou elementos que compõem o sistema;
- **Estrutura:** maneira como os componentes são organizados;
- **Funções:** transformações que o sistema executa; e
- **Propósito:** objetivo que o sistema deve alcançar.

Dentre os sistemas construídos pelo homem, existem aqueles onde um dos componentes fundamentais é a informação. Esses sistemas, ditos **sistemas de informação**, provêem procedimentos para armazenar e tornar disponíveis informações, que são utilizadas em atividades relacionadas a uma organização. Se os sistemas de informação utilizam computadores, eles são denominados **sistemas de informação computadorizados** ou **automatizados**.

Em uma definição abrangente, um **sistema baseado em computador** é um conjunto de elementos inter-relacionados, que possuem características comuns e que podem ser entendidos como um todo. Esses elementos inter-relacionados são (Pressman, 2002):

Software: composto de (1) instruções que, quando executadas, fornecem a função e desempenhos desejados; (2) as estruturas de dados que permitem aos programas manipular, de forma adequada, as informações e (3) os documentos que descrevem a operação e uso dos programas;

Hardware;

Pessoas: usuários e operadores de hardware e software;

Bancos de dados: uma coleção grande e organizada de informações que são acessadas via software;

Documentação: artefatos gerados ao longo do processo de software e que devem ser mantidos atualizados;

Procedimentos: os passos que definem o uso específico de cada elemento do sistema ou o contexto de procedimentos no qual o sistema reside.

Esses elementos estão organizados para alcançar algum objetivo, através de processamento de informação. Este objetivo é basicamente apoiar alguma função de negócio ou desenvolver um produto que possa ser vendido para gerar renda.

1.3 O PRODUTO DE SOFTWARE (OU SIMPLEMENTE SOFTWARE)

Origem: Engenharia de Software, Notas de Aula, prof. Denise Franzotti Togneri, FAESA

O produto de software é o produto que os engenheiros de software projetam e constroem. Ele engloba os programas que são executados dentro de um computador de qualquer tamanho e arquitetura, os documentos, que englobam formulários virtuais e material impresso produzido por computador (*hard-copy*), e dados, que combinam números e texto, mas também incluem representações de informações em áudio, vídeo e pictóricas.

O **produto de software** (ou simplesmente **software**) é composto de (Pressman, 2002):

- (1) as **instruções** (os **programas de computador**) que quando executados fornecem a função e desempenho desejados;
- (2) as **estruturas de dados** que permitem aos programas manipular as informações de forma adequada;

(3) os **documentos** que descrevem a operação e uso dos programas.

O produto de software é o componente lógico de um sistema informatizado, e não físico. São produzidos pelo processo e pelas suas atividades e servem de matéria-prima para os mesmos. Por exemplo: documento de requisitos, programa executável.

Hoje, o produto de software tem um papel duplo. Ele é um produto, e ao mesmo tempo é um veículo para distribuir um produto. Como produto, ele distribui o potencial computacional personificado através do hardware do computador. Tanto residindo dentro de um telefone celular ou sendo executado em um *mainframe*, o produto de software é um transformador de informação - produzindo, gerenciando, adquirindo, modificando, exibindo ou transformando informação. Como veículo usado para distribuir um produto, o produto de software atua como base para controle de computadores (sistemas operacionais), comunicação de informação (softwares de gerenciamento de redes) e criação e controle de outros programas (ambientes e ferramentas). O produto de software distribui o que muitos acreditam ser o mais importante produto de século XXI - a informação (Pressman, 2002).

1.4 OS CUSTOS DO SOFTWARE

O principal desafio nas três primeiras décadas a partir do surgimento do computador, era produzir um hardware que reduzisse o custo de processamento e armazenagem de dados. Ao longo da década de 80 o desafio era o de melhorar a qualidade (e reduzir os custos) de soluções baseadas em computador. Soluções que são implementadas com software.

Existe um imenso potencial industrial que pode ser melhorado através do uso de novas tecnologias. O software é um dos principais mecanismos que nos possibilita aproveitar e dar a vazão a esse potencial.

Ian Sommerville (2006) destacou o seguinte sobre os custos do software:

- Os custos de software dominam os custos de sistemas computacionais. Em um PC, os custos de software são freqüentemente maiores que o custo do hardware;
- Manter um software custa mais que desenvolvê-lo. Para sistemas com uma longa vida, os custos de manutenção podem ser muito maiores que os custos de desenvolvimento;
- A engenharia de software dedica-se ao desenvolvimento de software com custos adequados.

O mesmo autor ainda prossegue identificando quais são os custos do software, apresentando que:

- Aproximadamente 60% dos custos são custos de desenvolvimento e 40% são custos de testes. Para software sob encomenda, os custos de evolução normalmente excedem os de desenvolvimento;
- Os custos variam dependendo do tipo de sistema que está sendo desenvolvido e dos requisitos de atributos de sistema, tais como desempenho e confiabilidade;
- A distribuição de custos depende do modelo de desenvolvimento que é usado.

Origem: Engenharia de Software, Notas de Aula, Prof. Antônio Maria Pereira de Resende, UFLA

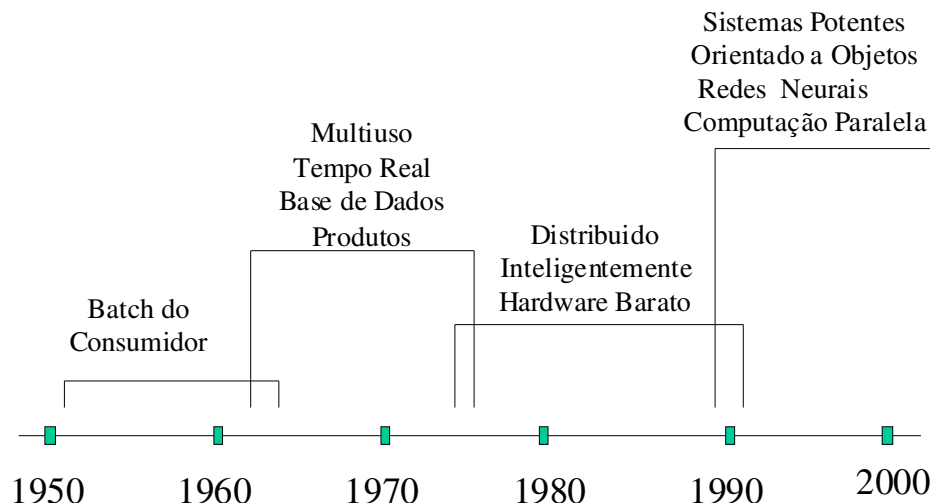
A proporção dos custos com software e hardware mudou bastante ao longo do tempo:

Década	% de Custo	
	Software	Hardware
60	20	80
70	50	50
90	80	20

- Toda empresa deseja a melhor solução pelo menor preço;

- Para se traçar Estratégias de Mercado deve-se primeiramente conhecê-lo;
- Conheça também a legislação da área buscando incentivos fiscais ou custos extras e não previstos em situações diversas;
- Custo do software geralmente domina o custo do desenvolvimento. Os custos do software de um PC são geralmente maiores do que do hardware;
- Software custa mais para manter do que para desenvolver! Para sistemas com uma longa vida, os custos de manutenção são várias vezes maiores do que o de desenvolvimento;
- Existem cada vez mais sistemas controlados por software;
- Os gastos com desenvolvimento de software representam uma fração significativa do PIB de muitos países. Software e Hardware podem trazer um grande desequilíbrio para a balança comercial brasileira, como também para outros países. Desenvolver soluções tecnológicas baseadas em software e hardware é fundamental. A tecnologia é uma bola de neve.

1.5 EVOLUÇÃO DO SOFTWARE



Os primeiros anos (1950 ... Início anos 60)

- Distribuição limitada;
- Software customizado;
- Sem documentação;
- Sistemas Batch;
- Na época muito se sabia sobre a implementação de sistemas baseados em computador, mas sabia-se pouco sobre engenharia de sistemas de computador.

Segunda Era (Meio Anos 60 ... Início Anos 70)

- Multi usuários;
- Sistemas Interativos;
- "Software Houses" (o software era desenvolvido para ampla distribuição no mercado interdisciplinar);
- Manutenção;

- Banco de dados;
- Tempo real;
- Software começa a ser encarado como produto.

Terceira Era (Meio Anos 70 ... Meio Anos 80)

- Comunicações digitais de largura de banda;
- Redes globais e locais;
- Inteligência embutida;
- Sistemas distribuídos;
- Hardware de baixo custo;
- Impacto de consumo;
- Crescente demanda de acesso instantâneo;
- Deslocamento de Hardware para Software;
- Ferramentas CASE;
- “Crise de software”.

Quarta Era (Meio Anos 80 ... 2000)

- Redes neurais artificiais;
- Computação paralela;
- Sistemas de desktop poderosos;
- Tecnologias orientadas a objeto;
- Sistemas especialistas;
- Sistemas Hipermídia;
- Computador na Educação (ITS – Sistemas Tutores Inteligentes).

Quinta Era

- Robotização Pesada;
- Arquiteturas de computação radicalmente diferentes, e seu software correlato exercem um profundo impacto sobre o equilíbrio do poder político e industrial em todo mundo;
- Sociedade da Informação.

1.6 QUESTIONAMENTOS SOBRE O PROCESSO DE DESENVOLVIMENTP

No começo:

- Os sistemas baseados em computadores eram desenvolvidos para a área de hardware, pois estes eram o maior item de orçamento particular do desenvolvimento do sistema. Para ver o custo do hardware os gerentes instituíram formas e padrões técnicos e exigiam análises do projeto antes que algo fosse construído visando sempre melhorias;
- O desenvolvimento do software era visto como uma forma de arte, havia poucos métodos e poucas pessoas o usavam, sendo que muitos por tentativas e erros. O palavreado era um grande desafio e muito indisciplinado.

A medida que o tempo passou, conforme destacou Pressman (2006): O software tornou-se fator dominante na economia do mundo industrializado. A figura do programador solitário do início da computação foi substituída por equipes de desenvolvimento de sistemas, com várias especialidades complementares. No entanto, algumas das questões que o programador solitário fazia continuam sendo feitas pela equipe moderna. Entre essas questões pode-se citar:

- Por que demora tanto tempo para que os softwares sejam concluídos?
- Por que os custos são tão elevados?
- Por que não são descobertos todos os erros antes da entrega do software aos clientes?
- Por que se gasta tanto tempo e esforço para realizar manutenção nos softwares?
- Por que temos dificuldade em medir o progresso enquanto o software está sendo desenvolvido?

1.7 CONCEITOS SOBRE SOFTWARE

1.7.1 DEFINIÇÃO

A definição mais conhecida e aceita sobre software de computador é aquela que indica que tal produto é composto por três itens:

- Programas de computador que, quando executados, produzem a função e o desempenho desejados;
- Estruturas de dados que possibilitam que os programas manipulem adequadamente a informação;
- Documentos que descrevem a operação e o uso dos programas.

Para responder à pergunta “O que é software?” Ian Sommerville (2006) indica o seguinte:

- Programas de computador e documentação associada, tais como requisitos, modelos de projetos e manuais de usuário.
OBS.: Nesta definição não está sendo destacada a Estrutura de Dados;
- Produtos de software podem ser desenvolvidos para um cliente particular ou para um mercado geral, se destacando basicamente em dois grupos:
 - **Genéricos** – desenvolvidos para serem vendidos para uma grande variedade de clientes, por exemplo, softwares para PC, tais como Excel e Word;
 - **Personalizados** – desenvolvidos para um único cliente de acordo com as suas especificações;
- Um software novo pode ser criado através do desenvolvimento de novos programas, da configuração de sistemas de software genéricos ou da reutilização de um software existente.

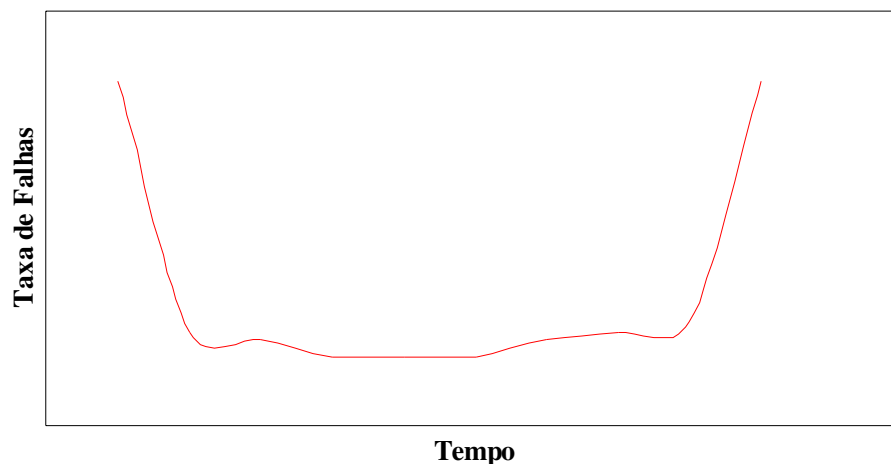
1.7.2 CARACTERÍSTICAS

Apesar de ser um produto, o software possui uma série de características que o diferencia de um produto habitual. Entre essas características podem ser citadas:

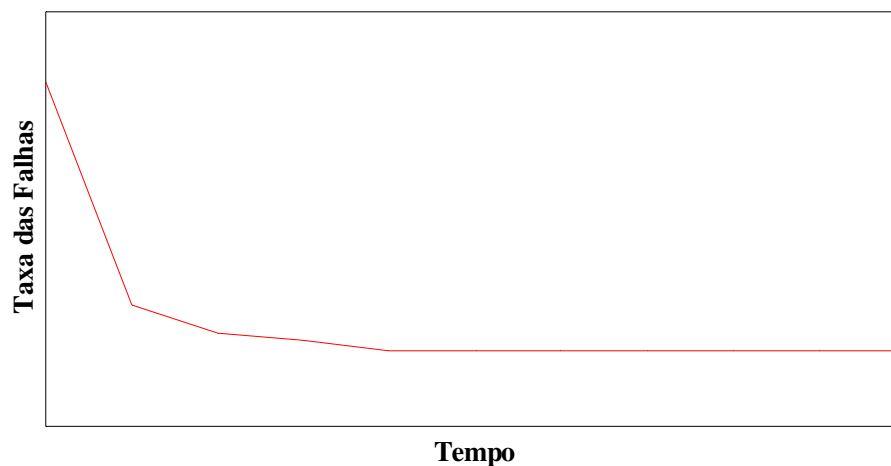
- Elemento de sistema **lógico** e não físico;

- Seus custos estão concentrados no trabalho de engenharia. Isso significa que os projetos de software não podem ser regidos como se fossem projetos de manufatura, isto é, não é fabricado no sentido clássico;
- Não é sensível aos problemas ambientais, como ocorre com o hardware (e por isso diz-se que o hardware se desgasta);
- **Sua alta qualidade é obtida mediante um bom projeto;**
- Não existe peça de reposição para o software, como ocorre com o hardware;
- Toda falha de software indica um erro no projeto ou no processo por meio do qual o projeto foi traduzido em código executável por máquina;
- Com poucas exceções, não existem catálogos de publicação, mas somente uma boa unidade completa, não são como componentes que possam ser montados novamente em novos programas;
- **Software não se desgasta, mas se deteriora**, pois durante a sua vida ele enfrentará mudanças (manutenção). Quando estas são feitas, é provável que novos defeitos sejam introduzidos. Essa situação fica melhor destacada nas figuras a seguir.

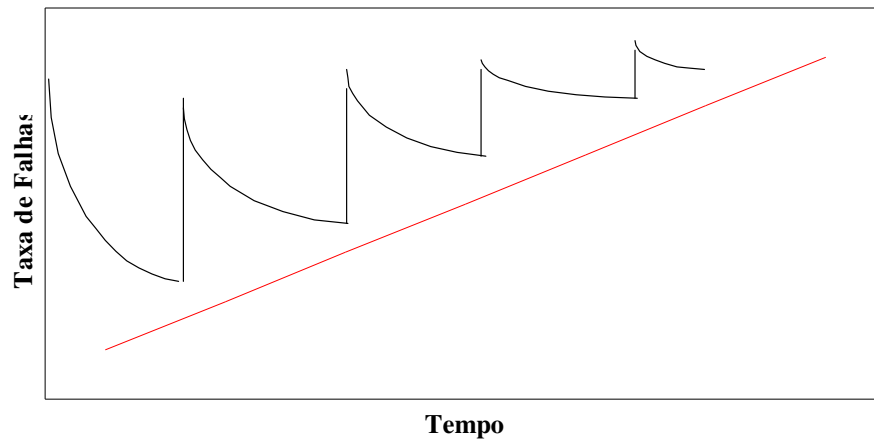
Curva das Falhas do Hardware



Curva Ideal das Falhas do Software



Curva Real das Falhas do Software



1.7.3 COMPONENTES

- Software de computador é uma informação que existe em duas formas básicas:
 - Componentes não executáveis em máquina.
 - Componentes executáveis em máquina
- Os componentes de software são criados por meio de uma série de conversões que mapeiam as exigências do cliente para código executável em máquina. O projeto de software é convertido numa forma de linguagem que especifica a estrutura de dados do software, os atributos procedimentais e os requisitos relacionados.

1.7.4 REUSABILIDADE

- A reusabilidade é uma característica importante de software de alta qualidade;
- Um componente deve ser projetado e implementado de forma que possa ser reusado em muitos programas diferentes;
- Na década de 60, construíam-se bibliotecas de sub-rotinas que reusavam algoritmos bem definidos efetivamente, mas tinham um domínio de aplicação limitado. Atualmente, ampliou-se a visão de recuo a fim de envolver não somente algoritmos, mas também estruturas de dados e interface;
- As interfaces interativas de hoje freqüentemente são construídas utilizando-se componentes reusáveis que possibilitam a criação de janelas gráficas, menus pull-down e uma ampla variedade de mecanismo de interação. As estruturas de dados e detalhes de processamento exigidos para se construir a interface com os usuários estão contidas numa biblioteca de componentes reusáveis para construção de interfaces.

Observação:

Ainda que muita coisa tenha sido escrita sobre “reusabilidade de software”, estamos apenas começando a ver as primeiras implementações bem-sucedidas do conceito.

1.8 CATEGORIAS DO SOFTWARE

Software pode ser incluído basicamente em qualquer situação, desde que previamente especificado por um algoritmo. Pode ser usado para diversos tipos de processos como, por exemplo, controlar

uma máquina automatizada que recebe e fornece várias informações e produz comandos de máquina individuais em rápida sucessão. Nessa situação, o programa só aceita dados com uma ordem predefinida, executa os algoritmos e fornece os resultados em um relatório ou em formato gráfico e essas aplicações são **determinadas**. Não é como outro sistema operacional, que aceita entradas de dados sem uma ordem cronológica e são **indeterminadas**.

Algumas categorias de software que indicam o tamanho das aplicações potenciais para o software:

Software Básico (Software de Sistema)

- Esse software é como vários programas reunidos para dar assistência a outros programas, tanto nos que processam as informações complexas e determinadas, como os que processam informações amplamente indeterminadas;
- A área do software básico é caracterizada por forte interação com o hardware de computador, intenso uso por múltiplos usuários, compartilhamento de recursos e sofisticada administração do processo, estruturas de dados complexas e múltiplas interfaces externas;
- Alguns exemplos: Sistemas Operacionais, compiladores, gerenciadores de banco de dados, editores, gerenciadores de redes.

Software Comercial (de aplicação)

- É a maior área de aplicação de software;
- Consiste de programas isolados que resolvem uma necessidade específica do negócio;
- Nesse sistema, a aplicação do software irá atuar no processo de informações comerciais. Atua nas áreas tanto administrativas quanto de produção de uma empresa, dando acesso a um ou mais bancos de dados contendo informações comerciais;
- Alguns exemplos: folha de pagamento, controle de estoque, administração de uma loja.

Software Científico e de Engenharia

- Tem sido caracterizado por algoritmos de processamento de números. Atua em diversos campos de pesquisa dentro da área científica e de engenharia. As novas aplicações estão se afastando dos algoritmos numéricos convencionais. Auxiliado por um computador, ele simula vários sistemas e outros tipos de aplicações.

Software Embutido

- Reside na memória só de leitura de alguns equipamentos e é usado para controlar produtos e sistemas para os mercados industriais e de consumo. Pode executar funções muito limitadas e particulares ou oferecer recursos funcionais de controle significativos;
- Alguns exemplos: Teclado para forno de microondas, computador de bordo de um automóvel.

Software para linhas de produto

- Desenvolvido para fornecer uma capacidade específica a ser utilizada pelos mais variados clientes. Exemplos: editores de texto, planilha eletrônica, calculadora.

Aplicações da Web

- As aplicações de comércio eletrônico e B2B estão crescendo em importância, o que destaca a relevância das aplicações Web, que estão evoluindo para ambientes computacionais cada vez mais sofisticados e interativos.

Software de Inteligência Artificial

- Faz uso de algoritmos não-numéricos para resolver problemas complexos que não sejam favoráveis à computação ou à análise direta. A área mais ativa é a dos **sistemas especialistas**. Outras áreas de aplicação para software de AI são o reconhecimento de padrão, jogos e demonstração de teoremas. Um simulador de estrutura dos processos cerebrais, chamado **redes neurais artificiais** também começa a ganhar destaque.

O que está acima identificado é referente a apenas uma classificação. Outras também podem ser consideradas, tendo-se assim novas categorias. Alguns exemplos:

Software de Tempo Real

- Analisa eventos do mundo real. Entre os elementos do software de tempo real incluem-se um componente de coleta de dados que obtém e formata as informações provenientes de um ambiente externo, um componente de controle que responde ao ambiente externo e um componente de monitoração que coordena todos os demais componentes de forma que a resposta em tempo real possa ser mantida;
- Monitora, analisa, e/ou controla eventos do mundo real enquanto esses ocorrem. O tempo de resposta é uma necessidade premente.

Software de Computador Pessoal

- Deixou de ser usado só para pequenos passatempos e trabalhos corriqueiros. São utilizados para quem quer fazer aplicações financeiras pessoais e comerciais de sua própria casa, fazer investimentos, etc. Destaca-se entre os mais inovadores projetos de interfaces com os usuários.

1.9 SOFTWARE LEGADO

Um software legado é um software mais velho, porém que permanece vital para o bom andamento das atividades de uma empresa. Dado o seu grau de importância, tal sistema, mesmo sendo antigo, continua sendo amplamente utilizado, recebendo manutenções, porém sem ser trocado / migrado. E as novas aquisições de software da empresa precisam ser integradas a este software legado.

Conforme destacou Roger Pressman (2006) um “software legado é caracterizado por longevidade e criticidade para o negócio”.

O mesmo autor ainda destaca o que talvez seja o maior problema deste tipo de software: a má qualidade. “Sistemas legados, algumas vezes, têm projetos não extensíveis, código complicado, documentação pobre ou inexistente, casos de teste e resultados que nunca foram arquivados, um histórico de modificações mal gerado...”

Porém não pode deixar de ser comentado que, se o software continuar em uso e atendendo as necessidades do cliente então ele não está danificado e não precisa ser consertado.

1.10 PROBLEMAS COM SOFTWARES E SUAS CAUSAS

Muitos observadores da indústria de software definem os problemas associados ao desenvolvimento do software como uma crise. Contudo, há controvérsias. Alguns dizem que o correto é aflição, pois crise deve ser algo com caráter passageiro.

Alguns dos problemas que afligem o desenvolvimento de software são:

- 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 de software, às vezes, é menor do que a adequada;
- Dificuldade de manutenção;
- Não é dedicado tempo para coletar dados sobre o problema a ser resolvido;
- A insatisfação com o sistema concluído ocorre muito freqüentemente. Falha de comunicação;

E as principais causas apontadas são:

- Os problemas ligados à “crise de software” estão relacionados ao próprio software e pelas falhas das pessoas que desenvolviam esses softwares;
- Muitas falhas são encontradas nos softwares e talvez tenham sido colocadas durante o desenvolvimento e passado despercebidas nos testes;
- Como o software é um elemento lógico, isso o torna um desafio para as pessoas que o desenvolvem;
- Pessoas desqualificadas recebem a responsabilidade pelo desenvolvimento do projeto;
- Definição incompleta dos requisitos do sistema.

1.10.1 PROBLEMAS ENFRENTADOS PELOS DESENVOLVEDORES DE SOFTWARE

Conforme destacou Roger Pressman (2006): “... as pessoas apostam seus empregos, sua segurança e suas próprias vidas em softwares de computador. É melhor que esteja correto.”. Ou seja, a responsabilidade que o desenvolvedor de software tem é enorme. No entanto, uma série de problemas são enfrentados para que tal profissional possa realizar suas tarefas, e entre esses problemas podem ser destacados:

- A sofisticação do hardware ultrapassou nossa capacidade de construir um software que extraia todo o potencial do hardware;
- Nossa capacidade de construir programas não pode acompanhar o ritmo da demanda de novos programas;
- Nossa capacidade de manter os programas existentes é ameaçada por projetos ruins e recursos inadequados.

1.11 MITOS DO SOFTWARE

O que é um mito?

- Parecem ser informações verdadeiras, razoáveis, mas não são;
- Informações criadas para propagar confusão em pessoas desinformadas do assunto;
- Atitudes enganosas que têm causado sérios problemas para usuários domésticos, gerentes e técnicos.

Tipos de Mitos:

1. Administrativos
2. Do Cliente

3. Do Profissional

Mitos Administrativos

Os Gerentes que têm responsabilidade pelo software, freqüentemente se encontram sob pressão para manter o orçamento e melhorar a qualidade. Tais gerentes muitas vezes se agarram a uma crença de um mito de software caso esse mito atenuar, mesmo que temporariamente, a pressão que pesa sobre ele.

Alguns exemplos de mitos:

- Possuímos normas, então está tudo controlado;
- Possuímos as melhores máquinas e ferramentas então teremos qualidade;
- Podemos sempre contratar mais pessoas para controlar atraso no projeto;
- Como o projeto está terceirizado, então o terceiro é o responsável por ele.

Mitos do Cliente

O cliente que exige um software acha que este pode ser elaborado por uma pessoa qualquer, e em muitos casos, o cliente acredita nos mitos sobre o software, porque normalmente os profissionais responsáveis pouco fazem para corrigir a desinformação. Os mitos levam a falsas expectativas (por parte do cliente), e levam a insatisfação com o desenvolvedor.

Alguns exemplos podem ser:

- Basta definir em geral os requisitos dos processos para que o desenvolvedor saiba o que é preciso;
- Software deve ser flexível, portanto mudanças podem ser facilmente acomodadas.

Mitos do Profissional

Muitos mitos merecem crédito dos profissionais de software até hoje, sendo que foram sustentados por décadas de cultura de programação. Durante os primórdios do software, a programação era vista como forma de arte, e como velhas maneiras e atitudes dificilmente morrem, ainda hoje existem esses mitos.

Alguns exemplos que se destacam são:

- Quando um programa funciona está pronto;
- Enquanto o programa não estiver rodando não posso avaliar a qualidade do mesmo;
- Só se deve entregar o programa executável utilizado;
- Com o uso de Engenharia de Software vai-se criar uma documentação volumosa e desnecessária, que causa atraso.

1.12 DICAS DE LEITURA

- PRESSMAN, Roger. **Engenharia de software**. 1995. Makron Books. Capítulo 1 – Software e Engenharia de Software (base para a maior parte do tópico descrito aqui);
- PRESSMAN, Roger. **Engenharia de software**. 2006. McGraw-Hill. Capítulo 1 – Software e Engenharia de Software;
- SOMMERVILLE, Ian. **Engenharia de software**. 2006. Pearson Education. Capítulo 1 – Uma introdução à engenharia de software;

CEUNES - UFES	Disciplina: Engenharia de Software
Matéria: Software	Página: 17

- Mitos da Tecnologia:
<http://www.microsoft.com/brasil/pequenasempresas/issues/starting/mitos4.mspix>;
- Software, Definição da Wikipédia. Link: <http://pt.wikipedia.org/wiki/Software>;
- O que é Software? Blog do Prof. Jair C Leite, UFRN:
<http://engenhariadesoftware.blogspot.com/2007/02/o-que-software.html>.

2 ENGENHARIA DE SOFTWARE

“Assuma a responsabilidade pelo software que você produz. Identifique o seu cliente e suas necessidades; pense nelas em termos de entradas e saídas. Pesquise o que já foi feito. Mantenha o controle intelectual dos seus esforços e documente aquilo que faz. Quando um problema for muito difícil, divida-o, sempre tentando manter o foco. E quando você tiver feito tudo direitinho, virão as mudanças; portanto, esteja pronto para elas.” [Dick Hamlet & Joe Maybee, em "*The Engineering of Software*"]

“Apesar de gerentes e profissionais reconhecerem a necessidade de uma abordagem mais disciplinada para o software, eles continuam a debater a forma pela qual essa disciplina deve ser aplicada. Muitos indivíduos e empresas ainda desenvolvem software ao acaso, mesmo quando constroem sistemas para servir às tecnologias mais avançadas da atualidade. Muitos profissionais e estudantes desconhecem os métodos modernos. Em decorrência disso, a qualidade do software que produzimos é sofrível e coisas ruins acontecem. Além disso, continua o debate e a controvérsia sobre a verdadeira natureza da abordagem de engenharia de software. O estado atual da engenharia de software é um estudo de contrastes. As atitudes mudaram, houve progresso, mas muito resta a ser feito antes que a disciplina alcance maturidade total.” [Roger Pressman, 2006]

2.1 ALGUMAS DEFINIÇÕES

Não existe um consenso sobre uma única definição para a disciplina de engenharia de software e, portanto, várias são aceitas. Algumas estão abaixo identificadas.

Uma primeira definição de engenharia de software foi proposta por Fritz Bauer na primeira grande conferência [NAU69] dedicada ao assunto: O estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que seja confiável e que funcione eficientemente em máquinas reais.

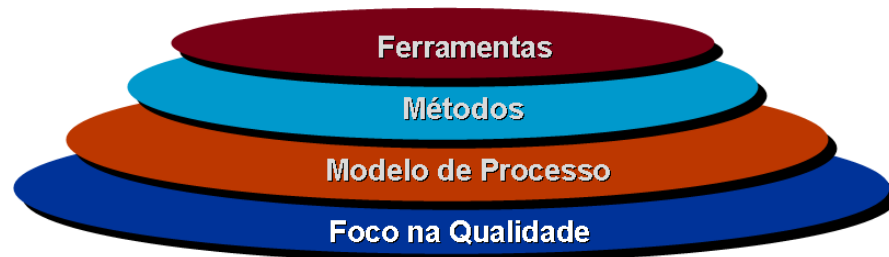
Uma segunda definição a se citar é aquela apresentada pelo IEEE, conforme destacou Pressman (2006). Engenharia de software é: (1) aplicação de uma abordagem sistemática, disciplinada e quantificável, para o desenvolvimento, operação e manutenção do software, isto é, a aplicação da engenharia ao software. (2) O estudo de abordagens como as de (1).

Uma terceira definição é: A área da Ciência da Computação que estuda, analisa e desenvolve técnicas e métodos para responder as seguintes perguntas no processo de desenvolvimento de sistemas:

- Que problemas o sistema de computação deve atacar e solucionar?
- Quais são os custos, tempos e outros recursos necessários para o desenvolvimento e a manutenção do sistema de computação desejado?
- É possível dividir o projeto em partes operacionais?
- Como assegurar que as partes operacionais, uma vez implementadas, serão compatíveis entre si?
- Como será a comunicação entre essas partes?
- Como o sistema de computação desenvolvido será repassado para os usuários interessados e os responsáveis pela sua manutenção?
- Qual a melhor forma de comunicação entre os usuários e o sistema de computação a desenvolver?

Nesta terceira definição o que se questiona é a disponibilização da Engenharia de software como uma parte da Ciência da Computação.

Pressman (2006) destaca que a Engenharia de Software é uma tecnologia em camadas, que tem como base a qualidade, se alicerça em processo, dispõe de métodos que fornecem a técnica de “como fazer” e utiliza apoio automatizado de ferramentas. Isso está representado na figura abaixo.



2.2 OBJETIVOS DA ENGENHARIA DE SOFTWARE

Podem ser destacados os seguintes objetivos para a aplicação da Engenharia de Software:

- Aplicação de teoria, modelos, formalismos, técnicas e ferramentas da ciência da computação e áreas afins para o **desenvolvimento** sistemático de software;
- Aplicação de métodos, técnicas e ferramentas para o **gerenciamento** do processo de desenvolvimento;
- Produção de **documentação** formal destinada a comunicação entre os membros da equipe de desenvolvimento, bem como aos usuários.

2.3 DIFERENÇAS

Entre Engenharia de Software e Ciência da Computação

- Ciência da Computação aborda as teorias e fundamentos;
- Engenharia de Software está interessada nos aspectos práticos de desenvolver e entregar no prazo um software útil.

Entre Engenharia de Software e Engenharia de Sistemas

- A Engenharia de Sistemas está interessada em todos os aspectos de um sistema baseado em computador, incluindo hardware, software, fatores humanos, informação e o processo;
- A Engenharia de Software é parte dela.

2.4 RESPONSABILIDADE PROFISSIONAL E ÉTICA

Conforme destacou Sommerville (2006):

- **Comportamento ético significa mais do que respeitar a lei.**

- A Engenharia de Software envolve responsabilidades mais abrangentes do que simplesmente a aplicação de habilidades técnicas. Os engenheiros devem se comportar de uma forma honesta e eticamente responsável se eles pretendem ser respeitados como profissionais.

Algumas questões de responsabilidade profissional:

- **Confidencialidade** - Os engenheiros devem normalmente respeitar a confidencialidade de seus funcionários ou clientes, independentemente de ter ou não assinado um acordo formal;
- **Competência** - Os engenheiros não devem desvirtuar o seu nível de competência. Eles não devem conscientemente aceitar um trabalho que esteja fora de sua competência;
- **Direitos sobre propriedade intelectual** - Os engenheiros devem estar cientes das leis locais que regem o uso de propriedade intelectual, tais como patentes, direitos autorais, etc. Eles devem tomar cuidado para assegurar que a propriedade intelectual dos funcionários e clientes seja protegida;
- **Mau uso de computadores** - Os engenheiros de software não devem usar as suas habilidades técnicas para fazer mau uso dos computadores de outras pessoas. O mau uso de computadores varia desde relativamente trivial (execução de jogos na máquina do funcionário, por exemplo) até extremamente sério (disseminação de vírus).

Destacando a importância que tal assunto tem ganhado uma série de instituições trabalham no sentido de apresentar Códigos de ética para os profissionais de Computação. Um dos mais conhecidos é o do ACM/IEEE, que consta de oito princípios éticos:

- **PÚBLICO** - Os engenheiros de software devem agir consistentemente com o interesse público;
- **CLIENTE E EMPREGADOR** - Os engenheiros de software devem agir dentro dos melhores interesses do seu cliente e empregador, de forma consistente com o interesse público;
- **PRODUTO** - Os engenheiros de software devem assegurar que seus produtos e as modificações a eles relacionadas atendam aos mais altos padrões profissionais possíveis;
- **JULGAMENTO** - Os engenheiros de software devem manter integridade e independência no seu julgamento profissional;
- **GERENCIAMENTO** - Os gerentes e líderes de engenharia de software devem contribuir e promover uma abordagem ética para o gerenciamento de desenvolvimento e manutenção de software;
- **PROFISSÃO** - Os engenheiros de software devem promover a integridade e a reputação da profissão de forma consistente com o interesse público;
- **COLEGAS** - Os engenheiros de software devem ser honestos e colaborativos com seus colegas;
- **INDIVÍDUO** - Os engenheiros de software devem participar, ao longo da vida, aprendendo, respeitando e promovendo uma abordagem ética na prática da profissão.

Ao longo de sua vida profissional um engenheiro de software pode se ver envolvido em diversos dilemas éticos, e ter um guia, como um código de ética, pode auxiliar na decisão que irá tomar frente a tal dilema.

2.5 Os Atributos de um Bom Software

Alguns atributos que podem ser utilizados para caracterizar um bom software são:

- Funcionalidade e desempenho exigidos pelo usuário;

- Manutenibilidade;
- Confiabilidade;
- Eficiência;
- Usabilidade.

Ian Sommerville (2006) ressalta o que está acima indicado, afirmando o seguinte:

- O software deve fornecer a funcionalidade e o desempenho requeridos para o usuário e deve ser manutenível, confiável e aceitável.
- Facilidade de manutenção - O software deve evoluir para atender às necessidades de mudança;
- Confiança - O software deve ser confiável;
- Eficiência - O software não deve desperdiçar os recursos do sistema;
- Usabilidade - O software deve ser aceito pelos usuários para o qual foi projetado. Isso significa que ele deve ser compreensível, usável e compatível com outros sistemas.

2.6 ASPECTOS HISTÓRICOS

Origem: Engenharia de Software, Notas de Aula, Prof. Antônio Maria Pereira de Resende, UFLA

2.6.1 OS PRIMÓRDIOS

- O desenvolvimento de software era puramente artesanal;
- Os desenvolvedores de sistemas erravam constantemente nas suas estimativas de custo e tempo;
- Os sistemas continham muitos erros;
- Consertar erros geralmente produzia mais erros;
- Sistemas crescendo a base de “remendos” mal feitos;
- Estudo feito em 1979 pelo governo dos Estados Unidos em relação ao software produzido indicou sobre tal produto:
 - 2% funcionava;
 - 3% funcionaria com poucas correções;
 - 45% entregues, mas nunca foram usados com sucesso;
 - 20% usados, mas tremendamente modificados ou abandonados;
 - 30% pagos, mas nunca foram terminados e/ou entregues.

2.6.2 A CRISE DE SOFTWARE

- A inexistência ou não aplicação de métodos, técnicas e ferramentas capazes de auxiliar e normatizar o processo de desenvolvimento causa prejuízos como os descritos acima;
- Após a estatística levantada pelo governo dos EUA e verificar-se tantas perdas, chegou-se ao consenso da necessidade de desenvolver-se métodos, técnicas e ferramentas capazes de auxiliar no desenvolvimento metódico de software e conseqüentemente reduzir as perdas;
- Desde a década de 60 uma crise acompanha a Indústria de Software. A “crise” é questionável quando observa-se o significado da palavra. Crise quer dizer “um ponto decisivo no curso de

algo, momento, etapa ou evento decisivo ou crucial". Até o momento não houve crise na área de software apenas uma contínua mudança lenta e evolucionária;

- Métodos de desenvolvimento de software existentes não são bons o bastante para o desenvolvimento de software de grande porte;
- No momento não se conhece uma cura para esta crise, apenas paliativos para reduzir os impactos muito desastrosos;
- Os principais problemas que envolvem a Crise de Software são:
 - Os grandes softwares não funcionam adequadamente;
 - Os projetos de software estão sempre atrasados;
 - Os custos dos projetos de desenvolvimento de software são sempre maiores do que o previsto;
 - Os computadores estão cada vez mais rápidos, sofisticados e baratos. Isto gera uma demanda cada vez maior de software. Por sua vez, 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;
- Dentre as principais causas dos problemas que envolvem a Crise de Software pode-se citar:
 - Dedicar-se pouco tempo à coleta de dados (requisitos dos clientes ou especificação de requisitos). Normalmente apenas um subconjunto das necessidades reais do cliente é levado em conta. Os profissionais estão sempre com muita pressa para começar a programar e esquecem que planejar é importante;
 - A qualidade geralmente é suspeita porque testes sistemáticos e tecnicamente completos raramente são feitos. Normas, modelos e padrões de qualidade são deixados de lado. A concorrência com software barato, mas sem qualidade, feito por pessoas sem qualificação adequada é muito grande. O cliente leigo infelizmente não sabe quais os fatores devem ser considerados e, portanto, olha apenas o preço;
 - 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. Muitos ainda acham que desenvolver um sistema é uma arte pessoal e não aplicam os devidos métodos, técnicas e ferramentas.

2.6.3 CRISE DO SOFTWARE – EXISTE SOLUÇÃO?

- A escolha e o uso das melhores e mais adequadas técnicas, métodos, e ferramentas;
- Mais treinamento aos funcionários envolvidos no processo. Atualmente se investe muito pouco nesta linha, porém o quadro vem mudando lentamente.

2.7 PRINCÍPIOS DA ENGENHARIA DE SOFTWARE

A fim de nortear o seu desenvolvimento e evolução a Engenharia de Software adota alguns princípios, que estão abaixo identificados:

- **Formalidade** – Trabalhar de maneira sistemática, obtendo produtos mais confiáveis, controle de custo e mais confiança no desempenho do software;
- **Abstração** – Identificar os aspectos importantes, ignorando os detalhes;
- **Decomposição** – Dividir o processo em atividades específicas, atribuídas a diferentes especialistas;

- **Generalização** – Sendo mais geral é bem possível que a solução possa ser reutilizada;
- **Flexibilização** – Modificação com facilidade.

2.8 PROBLEMAS QUE A ENGENHARIA DE SOFTWARE SE PROPÕE A RESOLVER

Alguns dos principais problemas que a Engenharia de software enfrenta desde o início praticamente e que ainda existem são os seguintes:

- Software projetado sob medida para a aplicação;
- Distribuição limitada à organização onde havia sido desenvolvido;
- Software criado e modificado por uma só pessoa ou grupo de pessoas da organização;
- Documentação precária ou inexistente, em razão da confiança na presença do analista;
- Aparecimento da fase de **manutenção de software** em razão de: softwares comprados externamente, criação de software houses, etc;
- A necessidade de mão-de-obra de manutenção assustou a indústria de software;
- A dificuldade de entendimento do software por terceiros, assustou ainda mais a indústria;
- Não há dedicação suficiente de tempo para a coleta de dados sobre o software a ser desenvolvido;
- A insatisfação do cliente com o produto produzido é relativamente freqüente. Isto é decorrente da comunicação falha entre o cliente e o desenvolvedor;
- A qualidade do software nem sempre é testada adequadamente;
- A evolução do hardware como: microprocessadores, computadores pessoais, etc, gerou um amplo conjunto de novos produtos, exigindo uma evolução paralela do software.

Os problemas dos dias atuais estão concentrados principalmente na imensa demanda de novas aplicações e ao desenvolvimento de técnicas que possam suprir essa demanda. A criação dessas técnicas faz surgir outro problema: capacidade de manter esses softwares.

Já Sommerville (2006) destaca os seguintes problemas, como sendo desafios-chaves da área:

- Heterogeneidade - Técnicas de desenvolvimento para construção de software que podem lidar com plataformas heterogêneas e ambientes de execução;
- Entrega - Técnicas de desenvolvimento para conduzir a entrega mais rápida de software;
- Confiança - Técnicas de desenvolvimento que mostram que o software pode ter a confiança dos seus usuários.

2.9 OS ELEMENTOS FUNDAMENTAIS DA ENGENHARIA DE SOFTWARE

MÉTODOS – Como construir software.

Envolvem um amplo conjunto de tarefas que incluem: planejamento e estimativa de projeto, análise de requisitos de software e de sistemas, projeto da estrutura de dados, arquitetura de programa e algoritmo de processamento, codificação, testes e manutenção.

FERRAMENTAS – Suporte (semi-)automatizado aos métodos.

Proporcionam apoio automatizado ou semi-automatizado aos métodos.

Atualmente existem ferramentas para sustentar cada um dos métodos citados anteriormente.

Ferramentas CASE (*Computer – Aided Software Engineering*).

Sistemas de software cujo objetivo é fornecer apoio automatizado as atividades de processo de software.

Exemplos: MS-Project
Rational Rose
Power Designer

PROCEDIMENTO – Seqüência de aplicação dos métodos.

Definem a seqüência em que os métodos serão aplicados, são o elo de ligação que mantém juntos os métodos e as ferramentas e possibilita o desenvolvimento racional e oportuno de software de computador.

2.10 PARADIGMAS DA ENGENHARIA DE SOFTWARE

A Engenharia de Software compreende um conjunto de atividades que envolvem métodos, ferramentas e procedimentos. Essas atividades podem ser executadas em diferentes seqüências e agrupadas em diferentes etapas. Os conjuntos de regras que definem essas etapas e seqüências são chamados de **paradigmas da engenharia de software**.

Para se escolher entre um ou outro paradigma devem ser levados em consideração diversos fatores, entre eles:

- Processo de desenvolvimento a ser adotado;
- Tipo de aplicação a ser desenvolvida (natureza do projeto);
- Métodos e ferramentas a serem utilizados;
- Controles e produtos que precisam ser entregues;
- Expectativas do cliente.

Origem:

<http://64.233.169.104/search?q=cache:iVUyqTfdfEIJ:www.dc.ufscar.br/~junia/aula%25204.pdf+%22modelos+de+ciclo+de+vida%22+%22engenharia+de+software%22&hl=pt-BR&ct=clnk&cd=10&gl=br>

Pode-se considerar 3 tipos de paradigmas que norteiam a atividade de desenvolvimento de software:

- Desenvolvimento de software como um artesanato: o projetista é um artesão.
 - As diversas legislações sobre software de vários países considerando o software protegido pela lei de direito autoral podem ser vistas nesse contexto;
 - Métodos que auxiliam o desenvolvimento de software não fazem muito sentido aqui;
 - Programas são obras pessoais;
 - Quando se considera grandes sistemas desenvolvidos em ambientes industriais, torna-se (no mínimo) inadequado esse paradigma.
- Matemática como modelo de desenvolvimento de software.
 - Um programa é um algoritmo escrito em uma linguagem;
 - Desenvolver algoritmos é resolver problemas, o que é uma atividade básica da matemática;
 - Portanto, desenvolver software é uma atividade intelectual muito próxima da matemática;

- Uso de métodos formais em todo o ciclo de desenvolvimento de software.
- Desenvolvimento de software como engenharia.
 - Leva à abordagem empírica;
 - A pesquisa está na busca de métodos e técnicas que aproximem ao máximo o processo de desenvolvimento de software do desenvolvimento das características de produtos em áreas tradicionais de engenharia;
 - A preocupação em se conseguir visualizar o produto de software já nas fases iniciais de desenvolvimento é um resultado da aplicação desse paradigma.

A tendência em buscar paradigmas em outras áreas de conhecimento é natural, dada a tenra idade da computação.

À medida que se cria um produto, o sistema de software, que será usado e mantido, nos aproximamos da engenharia.

À medida que esse produto não tem a solidez de uma máquina ou obra civil, nos afastamos da engenharia e nos aproximamos de sua analogia com uma obra intelectual, de autoria.

Pode-se dizer que o processo de desenvolvimento de software é um novo híbrido desses paradigmas. Esse processo, chamado de engenharia de software, é uma disciplina que engloba métodos, ferramentas e técnicas para o desenvolvimento de software.

2.11 OS MÉTODOS ADOTADOS PELA ENGENHARIA DE SOFTWARE

Ian Sommerville (2006) comenta sobre os métodos que a Engenharia de software procura adotar:

- Abordagens estruturadas para desenvolvimento de software que incluem modelos de sistema, notações, regras, recomendações de projeto e guia de processo.
 - Descrições de modelo de sistema - Descrições de modelos gráficos que devem ser produzidos;
 - Regras - Restrições aplicadas aos modelos de sistema;
 - Recomendações - Recomendações de boas práticas de projeto;
 - Guia de processo - Quais atividades devem ser seguidas.

2.12 FASES DA ENGENHARIA

2.12.1 DETALHANDO ...

Origem: <http://engenhariadesoftware.blogspot.com/2007/02/ciclo-de-vida-do-software-parte-1.html>

Existem várias propostas e denominações para as fases do ciclo de vida de um software. Nossa proposta identifica 4 fases que são delimitadas por eventos típicos em diversos ciclos de vida. Cada fase inclui um conjunto de atividades ou disciplinas que devem ser realizadas pelas partes envolvidas. Essas fases são:

- Definição;
- Desenvolvimento;
- Operação;
- Retirada.

Fase de Definição

A fase de definição do software ocorre em conjunto com outras atividades como a **modelagem de processos de negócios** e **análise de sistemas**. Nesta atividade, diversos profissionais buscam o conhecimento da situação atual e a identificação de problemas para que possam elaborar propostas de solução de sistemas computacionais que resolvam tais problemas. Dentre as propostas apresentadas, deve-se fazer um **estudo de viabilidade**, incluindo **análise custo-benefício**, para se decidir qual solução será escolhida.

O resultado desta atividade deve incluir a decisão da aquisição ou desenvolvimento do sistema, indicando informações sobre hardware, software, pessoal, procedimentos, informação e documentação.

Caso seja decidido pelo desenvolvimento do sistema, no escopo da engenharia de software, é necessário elaborar o documento de **proposta de desenvolvimento de software**. Esse documento pode ser a base de um **contrato de desenvolvimento**.

Profissionais de engenharia de software atuam nesta atividade com o objetivo de identificar os **requisitos de software** e **modelos de domínio** que serão utilizados na fase de desenvolvimento. Os requisitos são também fundamentais para que o engenheiro possa elaborar um **plano de desenvolvimento de software**, indicando em detalhes os recursos necessários (humanos e materiais), bem como as estimativas de prazos e custos (cronograma e orçamento). Não existe um consenso sobre o que caracteriza o final da fase de definição. Isto varia de acordo com o modelo de processo adotado. Em algumas propostas, a fase de definição é considerada concluída com a apresentação da proposta de desenvolvimento apenas. Outros modelos de processo consideram que o processo apenas está completamente definido com a **especificação de requisitos** e com a elaboração do **plano de desenvolvimento de software**.

De acordo com o modelo de processo adotado, pode-se iniciar atividades da fase de desenvolvimento mesmo que a fase de definição não esteja completamente concluída.

Fase de Desenvolvimento

A fase de desenvolvimento ou de produção do software inclui todas as atividades que tem por objetivo a construção do produto. Ela inclui principalmente o design, a implementação e a verificação e validação do software.

Design

A atividade de design compreende todo o esforço de concepção e modelagem que têm por objetivo descrever como o software será implementado. O design inclui:

- Design conceitual;
- Design da interface de usuário;
- Design da arquitetura do software;
- Design dos algoritmos e estruturas de dados.

O **design conceitual** envolve a elaboração das idéias e conceitos básicos que determinam os elementos fundamentais do software em questão. Por exemplo, um software de correio eletrônico tradicional inclui os conceitos: mensagem, caixa de entrada, caixa de saída, etc. A mensagem, por sua vez, inclui os conceitos de para, cc, bcc, assunto, corpo, etc. Embora seja um design adotado pela maioria dos softwares, novos modelos conceituais podem vir a ser adotados. O conceito de conversação do Gmail é um exemplo.

O design conceitual exerce influência na interface de usuário e na arquitetura do software. O **design da interface de usuário** envolve a elaboração da maneira como o usuário pode interagir para realizar suas tarefas, a escolha dos objetos de interfaces (botões, menus, caixas de texto, etc.), o *layout* de janelas e telas, etc. A interface deve garantir a boa **usabilidade** do software e é um fundamental fator de sucesso do software.

O **design de arquitetura de software** deve elaborar uma visão macroscópica do software em termos de componentes que interagem entre si. O conceito de componente em arquitetura varia de acordo

com a visão arquitetônica adotada. São exemplos de visões arquitetônicas, a visão conceitual, visão de módulos, visão de código e visão de execução.

Na **visão conceitual**, os componentes de software são derivados do design conceitual. Estes componentes são abstrações que devem definir outros elementos menos abstratos. Exemplos são arquiteturas cliente-servidor e arquitetura em camadas. Na **visão de código**, deve-se determinar como as classes e/ou funções estão organizadas e interagindo entre si. Estas classes implementam os componentes abstratos ou conceituais. Na **visão de módulos**, deve-se determinar quais são os módulos que serão utilizados na implementação e como eles organizam as classes e/ou funções. Na **visão de execução**, a arquitetura deve descrever os diferentes processos que são ativados durante a execução do software e como eles interagem entre si. Enquanto as anteriores oferecem uma visão estática, esta é uma visão dinâmica do software.

O **design de algoritmos e estrutura de dados**, também conhecido como design detalhado, visa determinar, de maneira independente da linguagem de programação adotada, as soluções algorítmicas e as estruturas de dados associados. Deve-se decidir, por exemplo, como as informações podem ser ordenadas (algoritmo de bolha ou quicksort) e em qual tipo de estrutura de dados (array, lista encadeada) elas vão ser armazenadas.

Implementação

A **implementação** envolve as atividades de codificação, compilação, integração e testes. A codificação visa traduzir o design num programa, utilizando linguagens e ferramentas adequadas. A codificação deve refletir a estrutura e o comportamento descrito no design. Os componentes arquiteturais devem ser codificados de forma independente e depois integrados. Os testes podem ser iniciados durante a fase de implementação. A depuração de erros ocorre durante a programação utilizando algumas técnicas e ferramentas. É fundamental um controle e gerenciamento de versões para que se tenha um controle correto de tudo o que está sendo codificado.

Verificação e validação

Verificação e validação destinam-se a mostrar que o sistema está de acordo com a especificação e que ele atende às expectativas de clientes e usuários. A validação visa assegurar se o programa está fazendo aquilo que foi definido na sua especificação (**fazendo a coisa certa**). A verificação visa verificar se o programa está correto, isto é, não possui erros de execução (**fazendo certo a coisa**). Existem diferentes formas de verificação e validação. Inspeção analítica e revisão de modelos, documentos e código-fonte são formas que podem ser usadas antes mesmo que o programa seja completamente codificado. Os testes de correção, desempenho, confiabilidade, robustez, usabilidade, dentre outros, visam avaliar diversos fatores de qualidade a partir da execução do software. Diferentes técnicas de testes podem ser aplicadas para cada um destes fatores

Fase de Operação

A fase de operação envolve diferentes tipos de atividades:

- Distribuição e entrega;
- Instalação e configuração;
- Utilização;
- Manutenção.

A distribuição e entrega pode ser feita diretamente pelo desenvolvedor (em caso de software personalizado), ou em um pacote a ser vendido em prateleiras de lojas ou para ser baixado pela Internet (em caso de software genérico).

O processo de instalação e configuração, normalmente, pode ser feito com a ajuda de software de instalação disponibilizado pelos fabricantes dos ambientes operacionais.

A atividade de utilização é o objeto do desenvolvimento do software. A qualidade da utilização é a usabilidade do software.

A manutenção normalmente ocorre de duas formas: corretiva e evolutiva. A manutenção corretiva visa a resolução de problemas referentes à qualidade do software (falhas, baixo desempenho, baixa

usabilidade, falta de confiabilidade, etc.). A manutenção evolutiva ou adaptativa visa à produção de novas versões do software de forma a atender a novos requisitos dos clientes, ou adaptar-se às novas tecnologias que surgem (hardware, plataformas operacionais, linguagens, etc). Mudanças no domínio de aplicação implicam em novos requisitos e incorporação de novas funcionalidades. Surgimento de novas tecnologias de software e hardware e mudanças para uma plataforma mais avançada também requerem evolução.

Fase de retirada

A fase de retirada é um grande desafio para os tempos atuais. Diversos softwares que estão em funcionamento em empresas possuem excelentes níveis de confiabilidade e de correção. No entanto, eles precisam evoluir para novas plataformas operacionais ou para a incorporação de novos requisitos. A retirada desses **softwares legados** em uma empresa é sempre uma decisão difícil: como abrir mão daquilo que é confiável e ao qual os funcionários estão acostumados, após anos de treinamento e utilização?

Processos de **reengenharia** podem ser aplicados para viabilizar a transição ou **migração** de um software legado para um novo software de forma a proporcionar uma retirada mais suave.

2.12.2 RESUMINDO ...

- Fase de Definição = Foco no **QUE**
 - Deve-se **analisar os requisitos**, recursos e restrições para
 - **Apresentar propostas iniciais de soluções;**
 - **Estudar a viabilidade;**
 - **Planejar e gerenciar** o desenvolvimento;
 - Realizada a partir de **estimativas** e **análise de riscos** que se utilizam de **métricas**;
 - Etapas:
 - Análise / Especificação do Sistema;
 - Planejamento do Projeto de Software (Estudo de viabilidade, estimativas);
 - Análise de Requisitos;
 - Esta fase encerra-se com o **contrato de desenvolvimento**.

- Fase de Desenvolvimento = Foco no **COMO**
 - Etapas:
 - Design de Software - Design conceitual, design da interface de usuário, design da arquitetura de software, design de algoritmos e estruturas de dados;
 - Codificação / Implementação e integração - Codificação, compilação, integração e verificação de programas (testes, inspeção, depuração);
 - Validação da qualidade - Testes beta, avaliação de usabilidade, avaliação de desempenho, etc.

- Fase de Operação
 - Distribuição e entrega;
 - Instalação e configuração;
 - Utilização e administração;
 - Manutenção = Foco na **TROCA** (correções e melhorias).

- Corretiva – correção de erros;
- Evolutiva ou adaptativa – novas versões.
 - Novos requisitos;
 - Novas situações de operação – hardware, sistemas operacionais.
- Fase de Retirada
 - Migração;
 - ReEngenharia;
 - Engenharia Reversa.

2.13 DICAS DE LEITURA

- PRESSMAN, Roger. **Engenharia de software**. 1995. Makron Books. Capítulo 1 – Software e Engenharia de Software;
- PRESSMAN, Roger. **Engenharia de software**. 2006. McGraw Hill. Capítulo 2 – Processo: Uma visão Genérica;
- SOMMERVILLE, Ian. **Engenharia de software**. 2006. Pearson Education. Capítulo 1 – Uma introdução à engenharia de software;
- PÁDUA FILHO, Wilson. Engenharia de Software: Fundamentos, Métodos e Padrões. 2003. LTC. Capítulo 1 - Engenharia de Software;
- Notas de Aula – Princípios da Engenharia de Software. Link: <http://www.garcia.pro.br/07-02-ULBRA-Engenharia/ULBRA-ENGENHARIA-1-2-3.pdf>;
- Notas de Aula – Engenharia de Software, prof. Ricardo Falbo. Link: <http://www.inf.ufes.br/%7Efalbo/download/aulas/es-g/2006-2/NotasDeAula.pdf>;
- Material variado sobre Engenharia de Software - Blog do Prof. Jair C Leite, UFRN: http://engenhariadesoftware.blogspot.com/2007_03_01_archive.html;
- Texto sobre a situação atual de projetos de sistema (indicação de André de Mattos Ferraz): <http://meiobit.pop.com.br:80/meio-bit/software/engenharia-de-software-anatomia-de-um-projeto-fracassado>.