SISTEMAS COMPUTACIONAIS 7

ORGANIZAÇÃO DE COMPUTADORES 7
Álgebra booleana 7
Sistemas de numeração e representação de dados 8
Aritmética computacional 8
Microprocessadores: Componentes 8
Microprocessadores: Arquitetura 9
Dispositivos periféricos 9
Conceitos de interrupções 10
Estruturas de endereçamento 10
Compiladores, Ligadores e Interpretadores 11
SISTEMAS OPERACIONAIS 12
Conceitos Básicos 12
Gerenciamento de Processos 13
Gerenciamento de Memória 13
Sistemas de Arquivos 15
Gerenciamento de Dispositivos 16
Concorrência 17
Concorrência: Deadlocks 17
REDES DE COMPUTADORES 19
Comunicação de dados 19
Meios de transmissão 19
Redes locais e de longa distância 20
Topologias 20
Serviços de comunicação 21
Protocolos 22
Equipamentos 22
Sistemas Distribuídos 23
Segurança 23
ARQUITETURA OSI DA ISO 24
Camada Física 24
Camada de Enlace de Dados 25
Camada de Rede 25
Camada de Transporte 26
Camada de Sessão 26
Camada de Apresentação 26
Camada de Aplicação 26
TCP/IP 26
Camadas do TCP/IP 27
Camada de Acesso à Rede (ou Camada de Interface) 27
Camada Internet (ou Camada de Inter-Rede) 27
Camada de Transporte 28
Camada de Aplicação 28
Protocolo IP 28
Protocolo TCP 28
Formato do Pacote IP 29
Classes de Endereços IP 29
Algoritmo de Transmissão de pacotes IP 30
Algoritmo de Recepção de pacotes IP 30
WINDOWS (2003/XP/2000/98) 31
Bibliotecas de ligações dinâmicas 31
Arquitetura Distribuída 32
MODELO CLIENTE SERVIDOR 32
2 camadas 32
3 camadas 32
N camadas 32
Chamadas remotas 32
Sincronismo e filas de mensagens 33
CONCEITOS DE INTERNET 33
Domain Name Service/System (DNS) 33

Default Gateway *33*
Backbone *34*
Intranet *34*
Extranet *35*
Firewall *35*

BANCO DE DADOS 36

CONCEITOS 36
Administração de Dados *36*
Sistemas Gerenciados de Bancos de Dados (SGBDs) *36*
Linguagem de definição de dados e Linguagem de manipulação de dados *38*
Dicionário de dados *38*
Arquitetura de banco de dados *38*
Bancos de dados relacionais *38*
MODELAGEM DE DADOS 39
Modelo Entidade-Relacionamento *39*
Modelo Relacional *40*
12 Regras de Codd *40*
Álgebra Relacional *41*
NORMALIZAÇÃO 41
Dependência Funcional *42*
Primeira Forma Normal *42*
Segunda Forma Normal *42*
Terceira Forma Normal *43*
Outras Formas Normais *43*
AMBIENTE OPERACIONAL 43
Conceito de transação *43*
Concorrência *43*
Recuperação *44*
Segurança *44*
Independência dos Dados *44*
Integridade *45*
Procedimentos ("Stored Procedures") *45*
Visões ("views") *45*
Gatilhos ("Triggers") *45*
Índices e otimização de acesso *45*
BANCOS DE DADOS DISTRIBUÍDOS 45
Características *46*
Fragmentação e Replicação *46*
BANCO DE DADOS DE OBJETOS 46
SQL (ANSI) 47
Principais instruções de manipulação de dados *47*
Uso do Join *47*
Subconsultas *48*
Elaboração de consultas SQL *48*

SOLUÇÕES DE SUPORTE À DECISÃO 50

CONCEITOS DE DATA WAREHOUSE E APLICAÇÕES 50
Estruturas de armazenamento para Data Warehouse *51*
Data Marts *52*
Metadados no ambiente de inteligência de negócios. *52*
Ferramentas de front-end: principais recursos e aplicações *52*
Processo de construção de um Data Warehouse *53*
OLAP (ON-LINE ANALYTICAL PROCESSING) 53
Arquiteturas OLAP *54*
CONCEITOS DE MODELAGEM DIMENSIONAL 54
Desenho de modelos dimensionais a partir de modelos transacionais normalizados *55*
EIS - ENTERPRISE INFORMATION SYSTEM 56
ECM - ENTERPRISE CONTENT MANAGEMENT 56
DATA MINING 57

ANÁLISE E PROJETO DE SISTEMAS 58

ANÁLISE E PROJETO ESTRUTURADO DE SISTEMAS 58
MODELAGEM DE DADOS 58
MODELAGEM FUNCIONAL 58
O Dicionário de Dados *58*
DFD e Modelagem de Fluxo de Dados *59*
MÉTODOS FUNCIONAIS X MÉTODOS OO 60
Métodos Funcionais *60*
Métodos OO (orientados por objetos) *61*
ANÁLISE ESSENCIAL 61
Análise Essencial x Análise Estruturada *62*
ANÁLISE E PROJETO ORIENTADO A OBJETOS 62
Modelagem Dinâmica *62*
UML *63*
Diagrama de casos de uso *63*
Agregação x composição *64*
Diagrama de estados *65*
Diagrama de seqüência *65*
Diagrama de atividades *66*
Diagrama de colaboração/comunicação *66*
Diagrama de implementação *67*
Diagrama de componentes *67*
Utilização dos diagramas *68*
PADRÕES DE PROJETOS (*DESIGN PATTERNS*) 69
USO/CONCEITOS DE FERRAMENTAS DE SUPORTE À ANÁLISE E PROJETOS ORIENTADOS A OBJETOS 70

ENGENHARIA DE SOFTWARE 71

PRINCÍPIOS DE ENGENHARIA DE SOFTWARE 71
Engenharia de sistemas *71*
Ciclo de Vida *71*
Processos de Software *71*
Análise *72*
Projeto (design) e Codificação *72*
Verificação x Validação *72*
Testes *72*
Garantia da qualidade *73*
Manutenção *73*
MODELOS DE CICLO DE VIDA 73
Modelo Clássico ou Cascata *73*
Modelo orientado a reuso *74*
Modelo Evolucionário ou Prototipagem *74*
Técnicas de quarta geração *74*
Modelo em Espiral *75*
RAD - Desenvolvimento rápido de aplicações *75*
TESTES 76
Tipos de Testes *76*
Walkthough *77*
Técnica de prova de corretitude *77*
Simulações *77*
Testes de caixa preta *77*
Testes de caixa branca *78*
ENGENHARIA DE REQUISITOS 78
Requisitos *79*
Processo da Engenharia de Requisitos *79*
Elicitação de requisitos *79*
ENGENHARIA DE USABILIDADE 80
Análise de requisitos de usabilidade *80*
Projeto de interfaces *81*
PROCESSOS DE SOFTWARE 81
ISO 12207 ..... *81*
Metodologias ágeis: Extreme Programming *82*
Metodologias ágeis: FDD *83*

MDA – Model Driven Architecture *84*
MDD – Model Driven Development *85*
RUP 85
Melhores Práticas do RUP *86*
Elementos Estáticos do RUP *87*
Disciplinas do RUP *88*
Fases do RUP *88*
MODELOS DE MELHORIA DE QUALIDADE DE PROCESSO E PRODUTO 88
CMM/CMMI *89*
ISO 9126 ..... *92*
Gestão da Qualidade *93*
Atividades da Qualidade Total (ISO 9001) *94*

MODELAGEM DE PROCESSOS DE NEGÓCIO 95

Conceitos básicos *95*
Identificação e delimitação de processos de negócio *95*
Técnicas de mapeamento de processos (modelos AS-IS) *96*
Técnicas de análise e simulação de processos *96*
Construção e mensuração de indicadores de processos *96*
Técnicas de modelagem de processos (modelos TO-BE) *96*
MODELAGEM DE PROCESSOS EM UML 96
Notação *96*
Artefatos *96*
Atividades *97*

GERÊNCIA DE PROJETOS 98

PMBOK 98
Conceitos *98*
Áreas de Conhecimento *98*
Análise de Risco *100*
Stakeholders *100*
Ciclo de Vida do Projeto *101*
Estrutura da Organização *101*
PROJETOS DE SOFTWARE 101
Estrutura de decomposição de trabalho (WBS) *101*
Planejamento *102*
Acompanhamento e Controle *102*
Gráfico de Gantt *102*
Ferramentas de Controle de Qualidade *103*
COLETA DE MÉTRICAS DE SOFTWARE 103
Estimativa de tamanho de software *104*
Análise de Pontos por Função (APF) *105*
COCOMO 106
Implementação Básica *106*
Implementação Intermediária *106*
Implementação Avançada *107*
ANÁLISE DE DESEMPENHO 107
Modelos Analíticos *107*
Simulação *108*
Medição de Desempenho *108*
Métricas de Desempenho *108*
Planejamento de Capacidade *109*

GESTÃO E RECURSOS INFORMACIONAIS 110

Serviço de TI *110*
Funções de TI *110*
MODELOS DE GOVERNANÇA EM TI: ITIL 111
Service Support *111*
Service Delivery *112*
MODELOS DE AUDITORIA EM TI – COBIT 113
Processos do COBIT *113*

Ferramentas de Gerenciamento do COBIT *114*
SISTEMAS DE GERENCIAMENTO ELETRÔNICO DE DOCUMENTOS (GED) 115
Aspectos Técnicos *115*
Evolução do GED *116*
SISTEMAS INTEGRADOS DE GESTÃO 116
ERP - Enterprise Resource Planning *116*
CRM - Customer Relationship Management *117*
AUTOMAÇÃO DE PROCESSOS DE TRABALHO (WORKFLOW) 118
Modelo de Referência *118*
Workflow no contexto do ERP *119*
GERENCIAMENTO DE PROCESSOS DE NEGÓCIO (BPM) 119
E-COMMERCE 120

TÉCNICAS E LINGUAGENS DE PROGRAMAÇÃO 121

LÓGICA 121
Lógica Formal *121*
ALGORITMOS E ESTRUTURAS DE DADOS 121
Noções de complexidade de algoritmo *121*
Tipos de dados *122*
Listas Encadeadas *122*
Pilhas *122*
Filas *123*
Vetores e Matrizes *123*
Árvores balanceadas *124*
Listas invertidas e Índices *125*
Métodos de acesso *125*
Métodos de ordenação e pesquisa *125*
Hashing *126*
PROGRAMAÇÃO 126
Programação Estruturada *126*
Modularização *126*
Sub-rotinas *127*
Escopo de Variáveis *127*
Tipos de dados *127*
Programação Orientada a Objetos *127*
Refactoring *128*
DESENVOLVIMENTO J2EE 128
Especificação J2EE *128*
Conceito de servidor de aplicação *128*
Container web e EJB *129*
Padrões e anti-padrões de projeto J2EE *130*
Padrão MVC de Projeto *130*

DESENVOLVIMENTO DE SISTEMAS WEB 132

HTML 132
CSS 133
JAVASCRIPT 133
Comandos do Javascript *133*
Eventos *134*
Funções do Javascript *134*
DHTML 135
ARQUITETURA DE APLICAÇÕES PARA AMBIENTE WEB 135
WEB SERVICES 135
Arquitetura Service-Oriented Architecture (SOA) *136*
Tecnologias dos Web Services *136*
PORTAIS CORPORATIVOS 137
Motivação e Características dos Portais *137*
Gestão de Conteúdo *138*
JSR 168 - Java Specification Request 168 Portlet Specification *140*
WSRP - Web Services for Remote Portlets *141*
ACESSIBILIDADE 142

Decreto nº 5296, de 02/12/2004 ..... *142*
Modelo de acessibilidade *142*
Cartilha técnica *143*
Recursos técnicos para implementação da acessibilidade em HTML *144*

SEGURANÇA DA INFORMAÇÃO 146

POLÍTICA DE SEGURANÇA 146
Regras gerais da Política de Segurança *146*
AMEAÇAS, ATAQUES E ANÁLISE DE VULNERABILIDADE 147
AUDITORIA DE SISTEMAS E SOLUÇÕES BASEADAS EM TECNOLOGIA DA INFORMAÇÃO 147
Requisitos de Segurança *148*
CERTIFICAÇÃO DIGITAL 148
Criptografia *149*
Criptografia Simétrica *149*
Criptografia Assimétrica *150*
Hashing *150*
Protocolos Criptográficos *150*
Assinatura Digital *151*

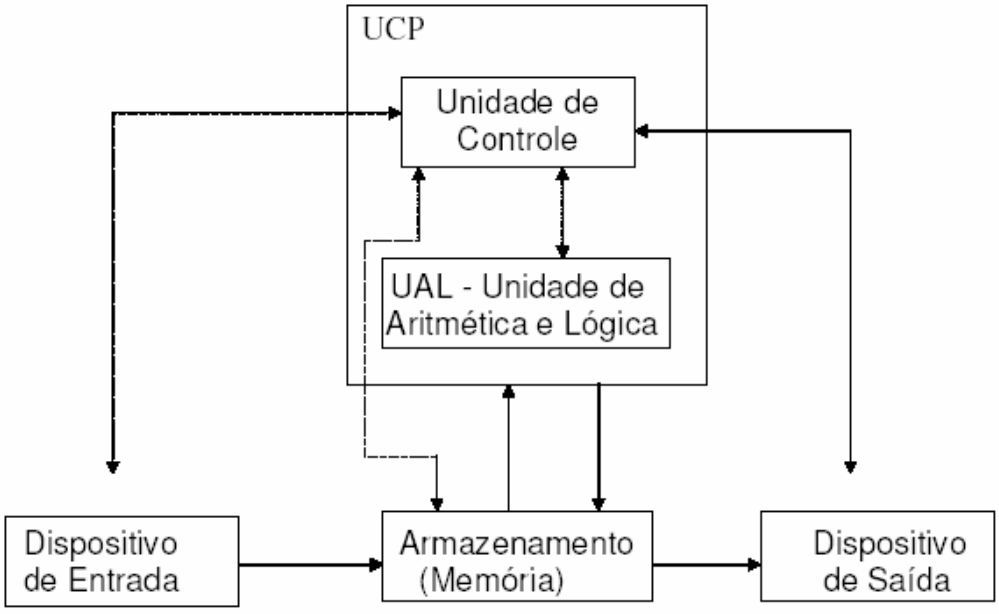
Sistemas Computacionais

Organização de Computadores

Um computador é organizado basicamente pela UCP, pela Memória e os Dispositivos de Entrada e Saída.

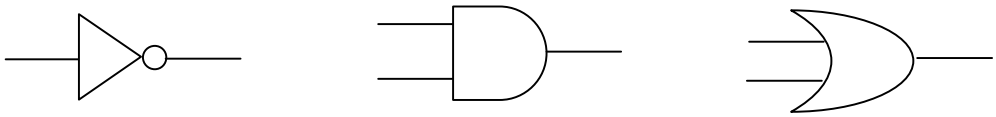
A UCP ou CPU (*Central Processing Unit*) é responsável pelo processamento e execução dos programas armazenados na memória principal. Ela é composta pela UAL (Unidade Aritmética e Lógica) e a UC (Unidade de Controle), que busca, interpreta e controla as instruções e demais componentes do computador.

Registradores são dispositivos de armazenamento temporário, localizados na UCP, extremamente rápidos, com capacidade para apenas um dado (uma palavra). Está localizado no chip, e os seus dados são voláteis.



Álgebra booleana

Os circuitos lógicos são construídos a partir das portas lógicas, que implementam fisicamente as funções booleanas básicas. A representação padrão destes três blocos lógicos é mostrada a seguir.



NOT AND OR

Os três principais operadores da álgebra *booleana* são os operadores NOT, AND e OR.

- O operador unário **NOT** é representado como $\bar{A}$. O resultado desta operação sobre uma variável é a inversão ou negação do valor da variável. Isto é, se a A = 1 então $\bar{A}$ = 0 e vice-versa.
- O operador **AND** é representado pelo símbolo •, como em A • B. O resultado da aplicação deste operador sobre variáveis *booleanas* é igual a 1 somente se todas as variáveis forem iguais a 1. Caso contrário, o resultado é 0. Esta operação é conhecida como produto lógico.
- O operador **OR** é representado pelo símbolo +, como em A + B. O resultado da aplicação deste operador sobre variáveis *booleanas* é igual a 1 se pelo menos uma das variáveis for igual a 1. Caso contrário, o resultado é 0. Esta operação é conhecida como soma lógica.

Existem várias leis descritas pela álgebra de *Boole* que são úteis no tratamento das equações lógicas:

- **Lei da identidade:** A + 0 = A e A • 1 = A;

- **Lei do zero e do um:** $A + 1 = 1$ e $A \cdot 0 = 0$;
- **Lei da inversão:** $A + \bar{A} = 1$ e $A \cdot \bar{A} = 0$;
- **Lei da comutatividade:** $A + B = B + A$ e $A \cdot B = B \cdot A$;
- **Lei da associatividade:** $A + (B + C) = (A + B) + C$ e $A \cdot (B \cdot C) = (A \cdot B) \cdot C$;
- **Lei da distributividade:** $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ e $A + (B \cdot C) = (A + B) \cdot (A + C)$.

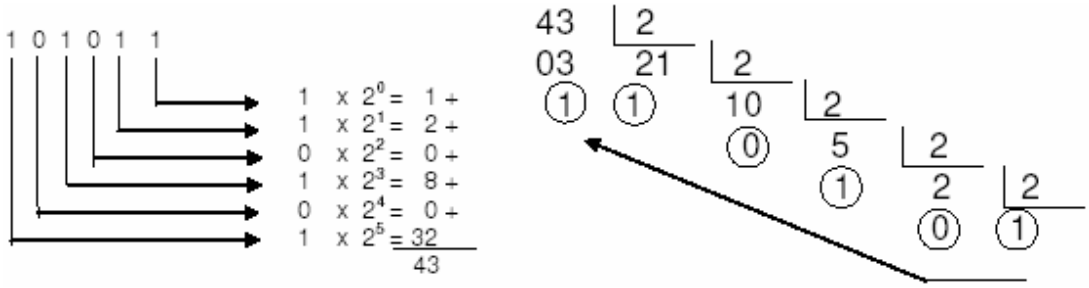
Além dessas leis existem dois teoremas conhecidos como **Teoremas de De Morgan**, cuja formulação é:

- $\overline{A+B} = \bar{A} \cdot \bar{B}$ e $\overline{A \cdot B} = \bar{A} + \bar{B}$

Sistemas de numeração e representação de dados

A nossa representação de números é baseada em um sistema de 10 algarismos (0 a 9), manipulados em múltiplos de 10. Esta representação é conhecida como a base 10.

Já os computadores utilizam apenas dois símbolos (0 e 1), representados pela voltagem elétrica (ligado ou desligado). Essa unidade de informação é chamada de bit, e é a abreviatura de *binary digit*, em inglês. Assim, um bit pode representar dois números, o 0 e o 1. Se usarmos dois bits, poderemos representar quatro estados diferentes, com três bits, já são oito estados, e assim por diante. Chamamos de um byte, o conjunto de oito bits, que pode representar 2^8 estados (256), ou os números de 0 a 255.



Aritmética computacional

As palavras de um computador são compostas por bits e podem representar números armazenados na memória. Estes números podem ter diferentes significados, como inteiros ou reais, serem positivos ou negativos:

- **Complementa Dois:** Os computadores manipulam tanto números positivos quanto números negativos, que são representados em complemento a 2. Nesta convenção os números que possuem 0s à esquerda são considerados positivos e os números com 1s à esquerda são considerados negativos. O complemento a 2 é obtido invertendo-se o número binário e depois somando 1 a este valor.
- **Adição e Subtração:** Numa soma os bits são somados um a um da direita para a esquerda, com os *carries* sendo passados para o próximo bit à esquerda. A operação de subtração usa a adição. O subtraendo é simplesmente negado antes de ser somado ao minuendo.
- **Multiplicação e Divisão:** A multiplicação precisa apenas de dois passos principais: o teste do produto e o seu deslocamento. A divisão é a operação recíproca da multiplicação. Dentre as operações aritméticas é a que aparece menos freqüentemente nos códigos dos programas.

Microprocessadores: Componentes

O processador (CPU) é o componente vital de um sistema de computação, responsável pela realização das operações de processamento (cálculos, entre outros) e de controle durante a execução de um programa.

O ciclo básico de uma instrução é: BUSCA -> DECODIFICÃO -> EXECUÇÃO. Outra divisão (para *pipelines*) é: Busca (*fetch*), Decodificação, Busca do Operando, Execução e Armazenamento do resultado.

O seu principal componente é a UC (Unidade de Controle). O processador conta também com uma Unidade Lógica e Aritmética (ALU), sendo que a sua ação é complementada pelo uso de registradores de processamento. E também possui um *clock*, fundamental para gerar pulsos e manter uma freqüência constante.

O processador também possui três barramentos: de endereço, de dados e de controle.

Alguns registradores importantes, que fazem parte da unidade de controle são: PC (*Program Counter*), IR (*Instruction Register*), MAR (*Memory Address Register*) e MBR (*Memory Buffer Register*).

As instruções executadas por um processador são as seguintes:

- **Operações lógicas e aritméticas:** somar, subtrair, multiplicar, dividir, and, or, xor, not,... ;
- **Movimentação de dados:** memória-CPU, CPU-memória, registrador-registrador, ...;
- **Desvios:** alteração da seqüência de execução das instruções;
- **Operações de entrada ou saída:** para comunicação com dispositivos de I/O.

Note que o hardware e o software são logicamente equivalentes, ambos podem executar as mesmas funções.

A função do processador é executar programas armazenados na memória principal. Para isso ele conta também com uma memória interna, os registradores. Revisando o passo a passo de execução de um programa:

1. Busca instruções da memória para o registrador de instrução (IR)
2. Atualiza o contador de programa (PC)
3. Determina o tipo de instrução, e (se usar) localiza os dados na memória
4. Busca os dados (se houver) para registradores
5. Executa a instrução, e armazena resultados
6. Volta ao passo 1 para a próxima instrução

Microprocessadores: Arquitetura

No início, mainframes, minicomputadores e microcomputadores possuíam claras distinções técnicas, em relação à quantidade de memória, palavras de 32, 16 e 8 bits (respectivamente), etc. Mas não existe nenhuma diferença conceitual entre eles, hoje em dia essa diferença é ainda menor.

As arquiteturas de microprocessadores são classificadas pelo número e quantidade de instruções:

- **RISC (*Reduced Instruction Set Computer*):** o conceito de processadores RISC foi levantado por John Cocke (da IBM), em 1974. O seu argumento era que os processadores só utilizavam 20% das instruções, e quanto menos instruções o processador conseguisse interpretar, menor o número de transistores (e menor o seu custo). Reduzindo o número de transistores e instruções a um mínimo possível, o processador conseguiria executar mais tarefas em um tempo menor. Exemplos são o MIPS, Power PC, Sparc e DEC Alpha.
- **CISC (*Complex Instruction Set Computer*):** Em contraste com o RISC, os chips CISC possuem uma grande quantidade de instruções diferentes e complexas. Os programadores executam um número menor de instruções do processador, pois cada instrução realiza tarefas mais complexas. Isso reduz o tamanho do código, diminuindo a necessidade de memória e armazenamento, e por conseguinte, diminuindo custos. Exemplos são os processadores da Intel.

Dispositivos periféricos

Em sistemas tais como microcomputadores e estações de trabalho, as interfaces de E/S são ligadas ao processador através de barramentos de endereço, dados e controle, de maneira semelhante à conexão entre memória principal e processador. A organização típica de um computador incluindo o sub-sistema de E/S é dado:

A comunicação entre a CPU e os periféricos é feita normalmente através de uma controladora ou interface, o que permite o CPU agir de maneira independente dos dispositivos, sem comunicação direta:

- **Polling**: O programa checa o estado do periférico manualmente. A CPU pode se manter ocupada até o término da operação. Uma solução é checar de tempos em tempos, executando concorrentemente.
- **Interrupção**: Ao terminar uma operação de E/S, uma interrupção é gerada para o processador. Isso elimina a necessidade de esperar pelo término, e várias operações podem ocorrer simultaneamente, através de um vetor de interrup-

ção. Quando a CPU é interrompida, ela pára e transfere a execução para um local fixo (rotina de serviço) para tratar a interrupção (divisão por zero, acesso inválido...)

- **Spooling:** Vêm de *Simultaneneous peripheral operation on-line*, e é uma técnica para aumentar a eficiência dos sistemas operacionais, base do sistema batch. Um exemplo é a utilização de impressoras.
- **Canais de DMA (*Direct Memory Access*):** é um recurso da placa mãe que permite que os dispositivos periféricos (placa de vídeo ou som, disco rígido, CD-ROM) acessem diretamente a memória RAM, sem consumir poder de processamento da CPU (apenas no início e no final da transferência). Cada dispositivo pode ocupar apenas um canal. Note que o processador não acessa a memória enquanto a controladora de DMA o faz (um só barramento), mas pode usar a memória cache.

Conceitos de interrupções

Inicialmente, quando começou-se a usar sistemas operacionais, um programa que estivesse sendo executado tomava o controle da máquina e só quando o programa encerrava é que o controle era devolvido para o sistema operacional. Quando ocorria algum problema (por exemplo, um erro no programa) não havia meios do sistema operacional retomar o controle da máquina, exceto com o usuário derrubando o sistema e recarregando o sistema operacional. Ficou então evidente a necessidade de haver um meio do sistema operacional poder retomar o controle da máquina, em situações de exceção.

Também ocorreu que as UCP's ficaram mais rápidas e mais eficientes, porém os dispositivos periféricos de E/S compatíveis progrediram menos em velocidade (ou seja, ficaram comparativamente mais lentos).

A resposta a estes problemas foi a utilização do mecanismo de INTERRUPÇÕES. Interrupções são modificações no fluxo de controle causadas por uma ação externa, geralmente relacionada a Entrada ou Saída. Uma interrupção é um sinal de controle enviado por um agente externo (um dispositivo) ao microprocessador, quando um determinado evento é detectado. A interrupção é um sinal de hardware.

Este mecanismo força o processador a tratar o evento externo. A detecção de uma interrupção faz com que o processador transfira o controle para uma rotina de tratamento de interrupção (*interrupt handler*). A rotina de tratamento de interrupções faz o processador executar as seguintes ações:

1. Detectar a fonte da interrupção (o dispositivo que interrompeu)
2. Executar as ações apropriadas (que dependem do dispositivo)
3. Retornar ao ponto do programa em que estava quando iniciou o atendimento à interrupção.

Interrupções de software (*traps* ou *exceptions*) são devidas a:

- Algum evento gerado pela execução de uma instrução, como uma divisão por zero, *overflow*, código de operação inválido, tentativa de acesso a uma área de memória protegida ou inexistente, etc
- A um evento programado.
- *Traps* são síncronas com o programa, enquanto interrupções associadas à E/S são assíncronas.

Estruturas de endereçamento

Os processadores podem acessar a memória de vários modos diferentes. A flexibilidade em seu acesso garante uma maior facilidade e performance ao trabalhar com variáveis, *arrays*, registros, ponteiros e outros dados.

- **Imediato:** a instrução contém o próprio operando. Mais rápido, mas a faixa de valores é menor.
- **Direto:** instrução contém o endereço de memória com o operando. Ex: mov a, 25h (do início da mem)
- **Indireto:** instrução contém o endereço de memória com o apontador para o endereço com o operando. Mais lento.
- **Com registrador:** instrução contém o registrador com o operando. Equivalente ao direto. Ex: *mov ax, bx*
- **Indexado:** instrução contém o (registrador + índice) que aponta para o endereço de memória com o operando.
- **Registrador-base:** Instrução contém o (reg. base + deslocamento) que aponta para o endereço com o operando.
- **Com pilha:** há um apontador de pilha, e utiliza-se instruções pop/push para obter o operando.

Compiladores, Ligadores e Interpretadores.

Existem dois tipos de aplicações para converter um código em uma linguagem para código de máquina:

- **Compilador:** traduz um programa escrito em uma linguagem de alto nível em código objeto. Esse código objeto é então ligado (*linked*) com os módulos de biblioteca de outros programas, para produzir um executável. O arquivo gerado pode ser então executado por um carregador, que também possui a função de resolver endereçamentos e outras tarefas que dependem do sistema operacional. Exemplo: C, C++, Pascal.
- **Interpretador:** interpreta e executa um código ao mesmo tempo. Ele não produz um arquivo executável, pois a sua saída depende do código fonte e dos valores de entrada. Exemplo: BASIC.

Sistemas Operacionais

O sistema operacional é responsável por gerenciar todos os recursos de uma máquina, facilitando a inter-face entre o hardware e o software do computador. Esses recursos incluem o processador (pode ser mais de um), a memória, os dispositivos de entrada e saída (impressora, monitor, teclado, etc), interfaces de rede, entre outros.

Esse gerenciamento de recursos de uma máquina é uma tarefa complexa, que é encapsulada pelo sistema. Imagine a tarefa de alocar um programa na memória. É preciso assegurar que há memória disponível suficiente, e que o local (endereço) onde o programa será alocado pode ser utilizado. Além disso, todas referências ao programa devem apontar para este mesmo endereço, e quando ele terminar, esse espaço deve ser liberado. Esta é apenas uma das tarefas do sistema operacional (SO).

Seria impraticável deixar a tarefa do gerenciamento de todos os recursos de um computador (principalmente o disco, processador e memória) aos programadores. Uma tarefa do sistema operacional é fornecer uma interface amigável para o programador, que pode utilizar funções de alto nível, como "Abrir Arquivo" ou "Ler Disquete" ao invés de ser obrigado a lidar com as interfaces complexas de discos rígidos ou unidades de disquetes.

Tanenbaum cita o exemplo de três programas tentando imprimir as suas saídas simultaneamente a uma mesma impressora. O resultado seria caótico, pois as linhas de cada programa poderiam sair intercaladas em uma mesma folha. O SO evitaria este cenário, imprimindo a saída de cada programa em sua devida ordem.

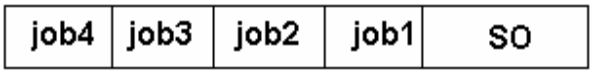
Conceitos Básicos

Como já explicado, o Sistema Operacional é um programa que atua entre o usuário e o hardware do computador, e proporciona um ambiente para executar programas de forma conveniente e eficiente.

Em resumo, o SO é um **alocador de recursos** e um **programa de controle**. A sua evolução é dada a seguir:

Sistema em batch: funcionava através da leitura de cartões perfurados, e sua memória possuía apenas o SO e um espaço para o programa do usuário.

Sistemas multiprogramados: a introdução dos discos permitiu que os *jobs* fossem escalonados pelo SO. Vários programas podiam residir na memória.



Sistemas de tempo compartilhado: permitiu a interação de vários usuários (multitarefa), adicionando o recurso de gerência de memória e proteção. O PC faz parte desta categoria.

Sistemas paralelos ou multiprocessados: com o objetivo de aumentar o *throughput*, mais de um processador compartilham o barramento, *clock*, memória, etc. Também há aumento da confiabilidade (tolerância à falhas). Um sistema multiprocessado possui uma programação mais complexa (**Fortemente Acoplado**).

Sistemas de tempo real: sistemas críticos, onde o processamento deve ser feito dentro de um limite de tempo.

Sistemas distribuídos: permitem que vários computadores, que NÃO compartilham memória ou *clock*, se comuniquem através de uma rede para completar uma tarefa. Um sistema distribuído é um conjunto de computadores que age como uma máquina única (**Fracamente Acoplado**).

A arquitetura dos computadores e sistemas operacionais em geral pode ser classificada quanto à sua história:

- **Primeira Geração:** válvulas e painéis. Toda a programação era feita em código absoluto.
- **Segunda Geração:** transistores. Computadores mais confiáveis, os sistemas *batch* já podiam ser implementados, leitura de cartões e linguagens de programação como FORTRAN ou linguagens de montagem.
- **Terceira Geração:** as técnicas de SPOOL, tempo compartilhado e multiprogramação foram implementadas. Vários terminais em uma máquina.
- **Quarta Geração:** circuitos integrados, surgindo a noção de computador pessoal. Tais máquinas passaram a se interconectar em rede, tiveram um grande aumento de seu poder de processamento e interatividade (*user-friendly*).
- **Quinta Geração:** começando em 1991, foram implementados sistemas especialistas, multimídia e distribuídos, a simplificação e miniaturização dos computadores. Os Pentiums da Intel se enquadram nessa categoria.
- **Próximas Gerações:** especulam sobre os computadores quânticos, que serão a "sexta" geração de computadores.

Gerenciamento de Processos

Um **programa** é uma seqüência de instruções que descrevem a execução de uma tarefa. Um **processo** é um programa em execução. O estado de um processo consiste do programa, da próxima instrução a ser executada, os valores das variáveis utilizadas, e o status dos dispositivos de entrada e saída.

Para guardar o estado de um processo, é criado um **vetor de estado**, que armazena todas as partes mutáveis do processo, ou seja, todos citados acima, menos o programa, que é imutável (em boas práticas de programação).

Para evitar que um programa acesse a área de outro, pode ser utilizado um registrador de base e outro de limite, com o espaço de endereçamento lógico do programa em execução. O próprio SO possui um mecanismo para se proteger de processos, que são os **bits de proteção** (modo usuário e modo privilegiado / monitor).

Processos precisam de tempo de CPU, memória, arquivos e dispositivos de I/O para realizar a sua tarefa. Estes recursos podem ser alocados durante a sua criação ou a sua execução. O Sistema Operacional deve:

- Criar e excluir processos de usuário e de sistema
- Suspender e retomar processos
- Dar mecanismo para a sincronização e comunicação de processos
- Tratar *deadlocks*

O **escalonamento de CPU** tem origem com a introdução de sistemas multiprogramados. Sempre que um processo precisa esperar (E/S), outro processo pode assumir o uso da CPU.

O escalonamento desse tipo é uma função fundamental do sistema operacional. Quase todos os recursos do computador são escalonados antes do uso, e esse escalonamento é vital para o projeto do Sistema Operacional.

Diferentes algoritmos de escalonamento de CPU possuem diferentes propriedades e podem favorecer uma classe dos processos em detrimento de outro. Alguns critérios de escalonamento são:

Ciclo de Burst CPU – E/S: Essa será a medida adotada para caracterizar cada processo em nosso algoritmo. A execução de um processo começa com um burst (surto) de CPU, que é seguido por um burst de E/S, que por sua vez, é seguido por outro burst de CPU, depois outro burst de E/S, e assim por diante. Por fim, o burst de CPU final termina com uma requisição do sistema para terminar a execução.

- **Turnaround (tempo de retorno):** também conhecido como tempo de execução. Do ponto de vista de um processo específico, o critério importante é o tempo necessário para executar esse processo. O intervalo desde o momento da submissão de um processo até o momento do término é o turnaround. O turnaround é a soma dos períodos gastos esperando para entrar na memória, esperando na fila de processos (*ready queue*), executando na CPU e realizando E/S.
- **Tempo de espera:** é a soma dos períodos gastos aguardando na fila de espera. Para sistemas interativos (como de tempo compartilhado) busca-se uma variância no tempo de resposta do que minimizar o tempo de resposta médio.

Alguns algoritmos de escalonamento de processos são:

FCFS (First Come First Served): neste escalonamento, o primeiro processo a chegar na fila é o primeiro a ser atendido, o que maximiza o seu *throughput*.

Round Robin: no escalonamento RR ou circular, os processos são organizados numa fila segundo sua ordem de chegada, sendo então despachados para execução. No entanto, ao invés de serem executados até o fim (completion), a cada processo é concedido apenas um pequeno intervalo de tempo (*time slice* ou quantum). Caso o processo não seja finalizado neste intervalo de tempo, ocorre sua substituição pelo próximo processo na fila de processos ativos, sendo o processo em execução interrompido e novamente colocado na fila de processos prontos, mas em seu fim. Isto significa que ao final de seu intervalo de tempo, isto é, de seu quantum, ocorre a preempção do processador, ou seja, o processador é designado para outro processo

Gerenciamento de Memória

A memória é o local onde os programas e dados são armazenados. A codificação binária (*bits*) foi escolhida para representar a informação, pois é mais confiável (e não mais eficiente!).

Uma **célula de memória** é a menor unidade endereçável. Cada célula tem um endereço único, e geralmente ocupam um *byte*, uma palavra, ou outra unidade.

Para evitar que dados sejam interpretados como instruções, ou vice-versa, um *bit* extra pode ser usado para cada célula para indicar o seu conteúdo. Na verdade vários *bits* extras podem existir com várias finalidades.

O processador central lê e grava na memória, assim como a controladora de DMA. Os programas devem ser mapeados para endereços absolutos, e instruções e dados são acessados. Além disso, vários programas ficam ao mesmo tempo na memória. O sistema operacional é responsável por:

- Manter um registro das áreas ocupadas da memória (e por quem)
- Decidir quais os processos que irão residir na memória
- Alocar e desalocar espaço na memória, conforme necessário

A CPU se comunica com a memória através dos registradores MAR e MBR, já explicados anteriormente.

A amarração de endereços lógicos (*binding* de programas executáveis) com físicos (da memória principal) é feita:

- **em tempo de compilação:** usado em arquiteturas antigas, onde o programa era sempre carregado na mesma posição da memória. Só faz sentido com monoprogramação
- **em tempo de carga:** executável contém mapa de relocação. O programa é interpretado, e os endereços corrigidos.
- **em tempo de execução:** há necessidade de hardware especial. Exemplo: registrador base cujo valor é sempre somado a endereço lógico.

Alocação Contígua Simples

A memória principal é dividida em duas partições: Sistema Operacional (parte baixa) e Processo do Usuário (restante da memória). O usuário tem controle total da memória, inclusive da área do SO (Ex.: DOS). A proteção (*bits* de proteção) foi posteriormente incluída, assim como registradores de base e limite.

Relocação é o processo de designar endereços de carga às várias partes do programa, ajustando código e dados para refletir os endereços designados.

Alocação Contígua Particionada

Imposta pela multiprogramação, existem múltiplas partições, e a memória é dividida em blocos. Cada partição pode receber um processo (programa), mas não considera a existência do *Swapping*. Para corrigir fragmentação interna nas partições, o seu tamanho passou a ser dinâmico, e não fixo. Mas isso também pode gerar fragmentação externa (pedaços livres da memória). Algoritmos para alocação contígua: *First Fit* (primeiro espaço), *Best Fit* (menor espaço livre) e *Worst Fit* (maior espaço).

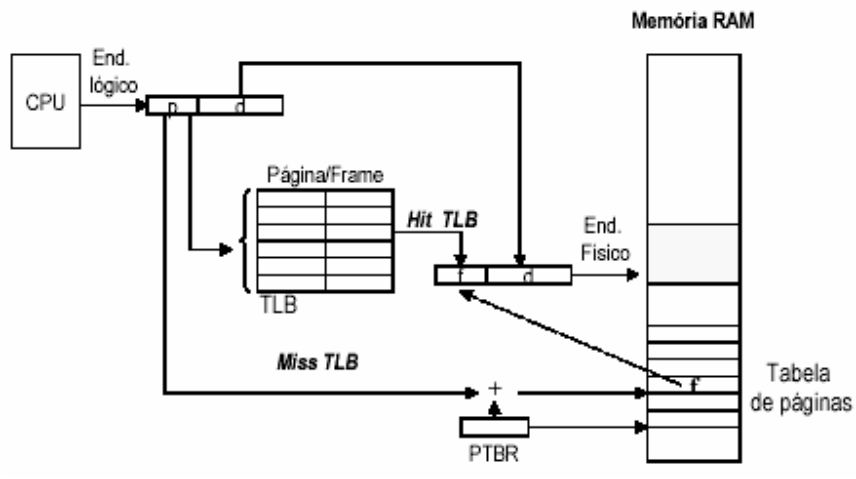
Swapping: é uma técnica para evitar a fragmentação externa, e lidar com a falta de espaço em memória, já que os processos precisam estar na memória principal para serem executados. Mas nunca deve ocorrer em processos com E/S pendentes.

Segmentação

- espaço lógico do programa é dividido em unidades que têm sentido para o programa: segmentos
- cada segmento pode ser alocado na memória física de forma independente

Paginação

- Mapeamento entre páginas físicas (*frames*) e páginas lógicas (*pages*)
- mapeamento de endereços através de uma tabela de páginas
- há fragmentação interna
- cache da tabela de páginas: TLB (*Translation Look-aside buffers*). Mas como o seu tamanho é limitado (de 8-2048 entradas), há a possibilidade de paginação multinível, para gerar tabelas de páginas menores
- **conjunto de trabalho:** páginas referenciadas por um programa. É desejável: espaço de endereçamento físico > conjunto de trabalho. Obs.: espaço de endereçamento virtual > espaço de endereçamento físico



Controle de espaço livre

- blocos de tamanho variável: listas encadeadas
- blocos de tamanho fixo: bitmaps

Memória Virtual

- Permite que o programa utilize um espaço de endereçamento virtual maior que a memória física total.
- parecido com *swapping* mas muito mais eficiente. Com *swapping*, o espaço de endereçamento de cada processo tem que ser menor que a MP; o que pode ser maior que a MP é a soma dos espaços de endereçamento de todos os processos.
- Apenas algumas das páginas virtuais têm um correspondente físico em cada instante; as demais páginas são marcadas como inválidas na tabela de páginas.
- Quando um programa faz um acesso a uma página que não está carregada na MP, a MMU (*Memory management Unit*) gera um trap para o sistema operacional, que entra em ação para carregar a página requisitada.

Sistemas de Arquivos

O Sistema Operacional fornece uma visão lógica uniforme do armazenamento. O conceito abstrato de arquivo serve para organizar os dados e seus acessos. O SO é responsável por:

- Criar e excluir arquivos ou diretórios
- Fornece suporte a primitivas para manipula-los
- Mapear arquivos no armazenamento secundário
- Fazer backup de arquivos nos meios estáveis (não-voláteis)

As informações são gravadas nos discos em "setores", distribuídos ao longo de "trilhas" concêntricas marcadas magneticamente como setores circulares no disco, conforme ilustração a seguir.

O processo de marcação magnética das trilhas e setores em um disco faz parte da "formatação" do disco. Esta formatação é dependente do sistema operacional que usará o disco. O sistema operacional DOS define que cada setor armazena 512 bytes. Todas as trilhas armazenam o mesmo número de bytes; desta forma, os dados na trilha mais interna estarão gravados com maior densidade, pois o espaço físico é menor.

Tempo de Acesso: o tempo de acesso aos dados de um disco é definido como o período decorrido entre a ordem de acesso e o final da transferência dos dados. O tempo de acesso não é constante, variando em função da posição relativa entre o braço atuador (que posiciona as cabeças de leitura e gravação) e o setor que será lido e portanto só tem sentido falar em tempo médio de acesso. Os tempos médios de acesso dos discos atuais são da ordem de 10 ms e é resultado das seguintes operações:

TEMPO DE ACESSO = TEMPO DE (SEEK + LATÊNCIA + TRANSFERÊNCIA)

- **Tempo de Seek:** seek ou busca é o tempo gasto na interpretação da localização do dado no disco (endereço do dado no disco) pela unidade de controle e no movimento mecânico do braço que sustenta a cabeça magnética, até alcançar a trilha desejada. Este tempo é variável de acesso para acesso. os tempos típicos de discos rígidos atuais podem variar de aproximadamente 0 ms (referente ao acesso a um setor localizado na mesma trilha onde no momento está a cabeça de leitura), 3 ms (para acesso a setores em trilhas adjacentes) a até 20 ms (referente ao acesso entre trilhas localizadas nas extremidades do disco). Este tempo é diretamente dependente da qualidade dos mecanismos eletromecânicos que comandam os braços atuadores. Discos de menores dimensões também tendem a ser mais rápidos.
- **Tempo de Latência:** também chamada latência rotacional, é o tempo gasto entre a chegada da cabeça de leitura / gravação sobre a trilha e a passagem do setor desejado na posição da cabeça. Como o disco permanece constantemente girando, a cabeça magnética só pode ler ou gravar um dado quando o setor endereçado está imediatamente abaixo dela. Portanto, há que aguardar que o disco gire até que o setor endereçado fique posicionado abaixo da cabeça. Esse tempo depende diretamente da velocidade com que o disco gira (5400, 7200 RPM, etc)
- **Tempo de Transferência:** é o tempo consumido na transmissão dos bits entre computador e disco e vice-versa. Este tempo depende da interface e do disco, que definem o throughput (taxa de transferência) do disco. Atualmente, dependendo da interface, o throughput seria da ordem de até 33 Mbytes/s. Como um setor tem 512 bytes, em 1 ms se poderia transferir cerca de 33 setores e o tempo de transferência de um setor seria da ordem de 15 ns.

Obs.: Os tempos relativos a unidades (*drivers*) de disquetes são muito maiores que os acima indicados para discos rígidos. *Drivers* de disquete giram a aproximadamente 300 rpm e o *throughput* é da ordem de 500 kbytes/s; os tempos de acesso médios são da ordem de 60 a 100 ms.

Após a formatação física, temos um HD dividido em trilhas, setores e cilindros. Porém, para que este disco possa ser reconhecido e utilizado pelo sistema operacional, é necessária uma nova formatação, chamada de formatação lógica. A formatação lógica consiste em escrever no disco a estrutura do sistema de arquivos utilizado pelo sistema operacional.

Um sistema de arquivos é um conjunto de estruturas lógicas e de rotinas que permitem ao sistema operacional controlar o acesso ao disco rígido. Diferentes sistemas operacionais usam diferentes sistemas de arquivos. Os principais são:

- **FAT16 (*File Allocation Table*):** utilizado pelo MS-DOS, Windows 95 e compatível com Windows 98, permitindo um máximo de 65526 clusters, que não podem ser maiores que 32 KB. A principal desvantagem desse sistema é o desperdício de espaço para partições maiores que 1 Gb, e o seu tamanho máximo por partição limitativo, de 2 Gb.
- **VFAT:** extensão da FAT16, que ao invés de permitir apenas arquivos com 11 caracteres (8.3), reserva uma área com nomes grandes, fazendo com que o MS-DOS interprete os arquivos como "Arquiv~1", "Arquiv~2", etc.
- **FAT32:** utiliza 32 bits de endereçamento para cada *cluster*, de apenas 4 Kb cada. Utilizado pelo Windows 95 e 98, a maior desvantagem é o alto número de *clusters*, diminuindo o desempenho. Tamanho máximo de 2 Terabytes.
- **NTFS (*New Technology File System*):** sistema de arquivos de 32 bits onde os setores do disco são endereçados diretamente. Cada unidade de alocação possui apenas 512 bytes, evitando o desperdício de disco. Utilizado pelo Windows NT em diante. É um sistema mais seguro e confiável que o FAT, além de flexível e adaptável.
- **EXT2:** é um sistema de arquivo utilizado apenas pelo Linux, que apresenta vários recursos avançados de segurança e suporte a partições de até 4 Terabytes. Outros sistemas relacionados são o EXT3, RaiserFS, etc.

Gerenciamento de Dispositivos

O Sistema operacional deve ocultar os detalhes de dispositivos de I/O de usuários. Ele deve:

- Fornecer uma interface geral de driver de dispositivos específicos
- Fornecer um componente de *buffering*, armazenamento em *cachê* e *spooling*.

Para evitar que a CPU fique presa por longos períodos em Entrada e Saída, podem existir processadores especializados de E/S, de baixo custo.

Além disso, o sistema operacional guarda uma **tabela de status de dispositivos**, com o registro de pedidos de I/O simultâneos.

O **DMA** é utilizado para evitar que transferências de I/O rápidas (síncronas), como fitas, disco ou rede de comunicação, não interrompam o CPU a cada byte transferido. O DMA transfere diretamente para a memória, sem interrupção da CPU. Uma interrupção é gerada apenas a cada bloco, para encontrar um *buffer* e configurar os registradores (com endereços inicial e fi-

nal). Como a memória só pode transferir 1 palavra / ciclo, a CPU às vezes concorre com o DMA para realizar operações de acesso à memória (apenas um barramento).

O código ASCII é um mapeamento de bits para caracteres de texto, por exemplo: **0** (47), **A** (65) e **a** (97).

Concorrência

A programação concorrente pode ocorrer de duas formas diferentes:

- **Processos:** ocupam espaços de endereços diferentes, e se comunicam usando *pipes* oferecidos pelo sistema operacional.
- **Threads:** ocupam o mesmo espaço de endereço de uma aplicação, e a comunicação é feita por um mecanismo da linguagem de programação (a JVM do Java oferece um mecanismo *multithreading*)

Em alguns tipos de aplicações, *threads* são essenciais, como no caso de aplicativos com interface gráfica, em que o programa pode esperar interação do usuário enquanto processa alguma tarefa. Outro exemplo são servidores que podem esperar por requisições de novos clientes enquanto lidam com requisições já enviadas.

Para que dois ou mais processos possam ter acesso a um mesmo recurso compartilhado, existe um mecanismo que controla este acesso, chamado de **mecanismo de sincronização**. Enquanto um processo estiver acessando determinado recurso, todos os outros que queiram acessá-lo deverão esperar. Isso se chama **Exclusão Mútua**.

A tentativa de implementar a exclusão mútua nos programas traz alguns problema. Os mais freqüentes são:

- **Velocidade de execução dos processos:** quando um processo mais rápido é obrigado à esperar que um lento use o recurso e o libere. Um gargalo gerado pela consistência dos processos onde o mais rápido ficará limitado à velocidade do mais lento. O sistema todo fica lento como conseqüência
- ***Starvation*:** o SO determina as prioridades dos processos, de duas formas diferentes: por escolha aleatória ou por prioridades. Quando a escolha é aleatória, existirá a probabilidade de um processo nunca ser escolhido, quando for uma escolha por prioridades, um processo de menor prioridade nunca receberá o acesso ao recurso, e ai este processo nunca executará sua rotina.
- **Sincronização condicional:** quando um recurso não está pronto para ser utilizado, o processo que vai acessar o recurso ficará em estado de espera até que o mesmo esteja pronto. Existe o risco deste recurso nunca ficar pronto por já estar com problemas. Então todo o sistema fica em espera.

Algumas soluções implementadas para lidar com este problema são:

- **Semáforos:** um semáforo é uma variável associada a um recurso compartilhado, indicando quando este está sendo acessado por um outro processo. Quando um processo entra ou sai da região crítica, o seu valor é alterado, de forma que outros processos possam entrar na região. Existe uma fila de espera associada ao semáforo, onde os processos terão acesso ao recurso na ordem de chegada.
- **Monitores:** são mecanismos de sincronização compostos de um conjunto de procedimentos, variáveis e estrutura de dados definidos dentro de um módulo cuja finalidade é a implementação automática da exclusão mútua entre seus procedimentos. A implementação é feita pelo compilador
- **Troca de mensagens:** é um mecanismo de comunicação e sincronização entre os processos, implementado pelo sistema operacional através de duas rotinas do sistema SEND (envio de mensagem para o processo receptor) e RECEIVE (recebimento do processo transmissor). No sistema de troca de mensagens, existe a possibilidade da mensagem se perder. Já o Endereçamento Indireto, que é o uso de uma área compartilhada para a troca de mensagens.

Concorrência: Deadlocks

Um problema da concorrência é quando métodos sincronizados chamam outros métodos sincronizados. Isso pode causar *deadlock*, quando uma *thread* A espera a liberação de um objeto, que está com a *thread* B. Mas esta só vai liberar o objeto após uma alteração de outra *thread* (A, por exemplo), monopolizando-a. Exemplos:

Cenário 1 (Não ocorre *deadlock*), saldo do caixa: R$0.00

- Cliente 1 é atendido (recebe acesso do caixa), deposita 1000 reais e libera o caixa
- Cliente 2 é atendido (recebe o acesso do caixa), saca 800 reais e libera o caixa.

Cenário 2: (Ocorre *deadlock*), cliente 2 chega antes de cliente 1

- Cliente 2 é atendido, tenta sacar 800 reais mas não há saldo. Cliente 2 espera haver saldo.
- Cliente 1 quer ser atendido (quer depositar 1000 reais) mas não é sua vez na fila. Ele espera a sua vez.

Solução: Evitar que métodos sincronizados chamem a outros métodos sincronizados. Caso não seja possível, controlar liberação e retorno dos acessos (hierarquia de chaves de acesso).

As quatro condições necessárias para ocorrer deadlocks (todas devem ser verdadeiras) são:

- **Exclusão mútua:** existe um recurso não-compartilhado (um processo por vez o usa)
- **Posse e Espera:** um processo mantém um recurso, e espera por outro de outro processo.
- **Não-Preempção:** um recurso só pode ser liberado voluntariamente.
- **Espera Circular:** um processo P0 espera por P1, P1 por P2, ..., Pn espera por P0.

Redes de Computadores

Os primeiros sistemas de computadores eram grandes e caros. Os seus usuários organizavam as tarefas a serem processadas em *jobs*, e o computador fazia o processamento em lote (*batch*). As tarefas eram executadas uma a uma, organizadas em uma fila de entrada. Na década de 80 surgiram os terminais remotos, que permitiram que os usuários pudessem acessar os computadores através de linhas de comunicação. Nesta mesma época surgiu o conceito de tempo compartilhado (*time sharing*), para otimizar o uso de *mainframes*. Os terminais passaram a ser ligados ao computador central por cabos e novos recursos surgiram utilizando esta interconexão.

As redes modernas têm raízes nos primeiros sistemas de telefones e telégrafos. Para lidar com o aumento no número de usuários, foi preciso criar novos meios de distribuir o processamento e utilização dos outros recursos pelas redes, criando o conceito de multi-processamento e sistemas de processamento distribuídos. Na década de 60, o governo dos EUA iniciou o projeto ARPA (*Advanced Research Project Agency*).

Uma rede de computadores é um conjunto de módulos processadores capazes de trocar informações. Seus objetivos são:

- **Compartilhamento de recursos:** Permite que programas, dados, área de armazenamento, periféricos, entre outros recursos, estejam disponíveis para qualquer um na rede, independente de sua localização física. Este é a principal motivação para o uso de redes de computadores.
- **Aumento na Confiabilidade:** Considerando que passa a existir redundância dos recursos compartilhados pela rede, a confiança de um sistema projetado para ter tolerância à falhas aumenta.
- **Redução de Custos:** O custo benefício de um microcomputador em relação ao seu processamento é muito mais baixo que o de um *mainframe*, logo a fragmentação do processamento é desejável.
- **Escalabilidade:** A possibilidade de aumentar o processamento do sistema gradualmente, a medida que cresce o volume do trabalho, adicionando novos módulos processadores, permite adiar custos.
- **Cooperação:** Pessoas distantes geograficamente podem trabalhar e cooperar de forma conjunta.

Comunicação de dados

A comunicação de dados em nível global tornou-se uma realidade do nosso tempo. O desafio que a comunicação de voz representou para as gerações passadas volta atualmente, sob a forma de aplicações de telemática, em várias modalidades de troca de informação entre computadores heterogêneos situados em ambientes remotos, interconectados através dos sofisticados meios oferecidos pela engenharia de telecomunicações.

A Comunicação de Dados diz respeito ao modo como os dados são manipulados, transformados e transportados entre pontos quaisquer. Exemplos disso são os modos *simplex* (unidirecional), *half-duplex* (uma direção de cada vez) *e full-duplex* de comunicação, as bandas passantes, as modulações de sinais e a criptografia.

As Redes de Computadores são mecanismos que utilizam a comunicação de dados e existem para prover: comunicação entre pontos, compartilhamento de recursos e confiabilidade da integridade dos dados transferidos.

Meios de transmissão

Basicamente existem dois tipos de tecnologias de meios de transmissão:

- **Redes em Difusão:** Apenas um canal de transmissão compartilhado por todas as máquinas.
- **Redes Ponto-a-Ponto:** Existem várias conexões entre pares individuais de estações.

A transmissão de dados, quando realizada nos dois sentidos é denominada duplex. No caso em que ela se realiza alternadamente, ou seja, ora num sentido, ora no outro, ela se denomina half-duplex. No caso em que ela se realiza simultaneamente nos dois sentidos, esta será denominada full-duplex.

Os bits podem ser transmitidos de forma **paralela**, onde são simultaneamente transportados por várias linhas (mais adequado à comunicação entre equipamentos próximos), ou **serial,** onde os bits são encaminhados através de uma única linha de comunicação (ideal para equipamentos muito distantes). A forma de delimitar os bits pode levar em conta duas diferentes filosofias — a transmissão serial síncrona e a serial assíncrona.

- **Transmissão síncrona:** os bits de dados são transmitidos segundo uma cadência pré-definida, obedecendo a um sinal de temporização (clock). O receptor, conhecendo os intervalos de tempo permitindo delimitar um bit, poderá identificar a seqüência dos bits fazendo uma amostragem do sinal recebido

- **Transmissão assíncrona:** a separação entre os bits é feita através de um sinal especial com duração variável. Um caso típico é a transmissão de caracteres; onde a cada grupo de bits constituindo um caractere são adicionados bits especiais para representar o início (*start bit*) e final deste (*stop bit*).

Redes locais e de longa distância

As redes são classificadas pela sua escala, que residem basicamente em três grupos:

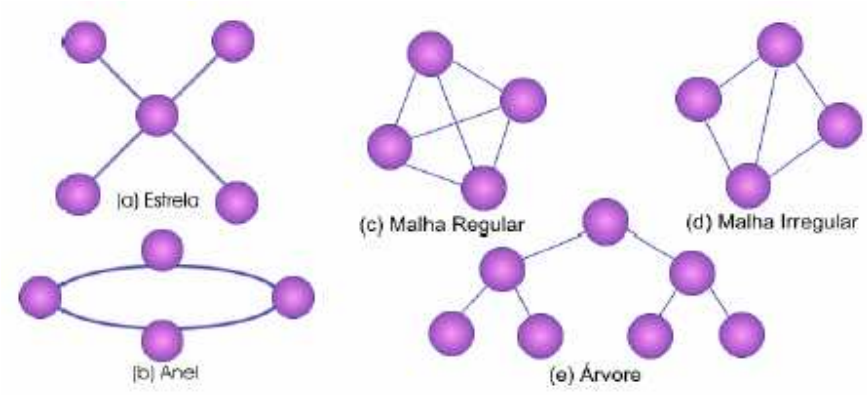
- **LAN** – Local Area Network ou Rede Local (normalmente uma sala, edifício ou campus).
- **MAN** – Metropolitan Area Network ou Rede Metropolitana (até o tamanho de uma cidade).
- **WAN** – Wide Area Network ou Rede Geograficamente Distribuída (ou de Longa Distância).

Topologias

A topologia de uma rede descreve como é o layout do meio através do qual ocorre o tráfego de informações e também como os dispositivos estão conectados a ele. Refere-se ao relacionamento físico e lógico de cada nó da rede (cada ponto de conexão com a rede pode ser chamado de nó, independente da função do equipamento representado por ele), ou seja, a forma como estão dispostos. Temos aqui então a divisão

- **Topologia lógica:** descreve como as informações devem transitar ao longo da rede, o formato dos dados, etc. É a forma como os protocolos operam no meio físico;
- **Topologia física:** refere-se à disposição dos cabos e componentes do meio físico, descrevendo onde cada nó da rede está situado fisicamente em relação aos demais, como é feita a distribuição da mídia de conexão (cabeamento de cobre, fibra óptica, wireless, etc) e mostra a configuração geral da rede através da planta de localização dos equipamentos.

O ambiente de funcionamento também influencia na escolha da topologia de uma rede. Ambientes ruidosos e com problemas de segurança têm requisitos mais exigentes quanto ao número máximo de nós, a separação máxima e mínima entre nós e a taxa máxima de informação transmitida. Alguns protocolos, por exemplo, levam em conta a distância máxima entre os nós da rede para seu perfeito funcionamento.



Ao lado estão relacionadas as topologias de redes ponto-a-ponto.

O envio de mensagens é feito de forma indireta, de forma que cada estação recebe o conteúdo integral da mensagem até ela alcançar o seu destinatário.

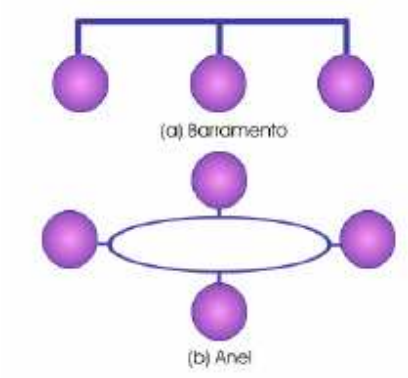
Tipos de Topologias	Ponto Positivos	Pontos Negativos
Topologia Estrela	É mais tolerante a falhas Fácil de instalar usuários Monitoramento centralizado	Custo de Instalação maior porque recebe mais cabos
Topologia Anel (*Token Ring*)	Razoavelmente fácil de instalar Requer menos cabos Desempenho uniforme	Se uma estação para todas param Os problemas são difíceis de isolar.
Topologia Barra-	Simples e fácil de instalar Requer menos cabos	A rede fica mais lenta em períodos de uso intenso.

mento	Fácil de entender	Os problemas são difíceis de isolar.

As redes em difusão, por outro lado, possuem apenas um barramento, sendo utilizadas em redes locais (curta distância). As mensagens são enviadas para todas as estações.

Um campo de endereço identifica a mensagem, e as estações que não são as destinatárias ignoram a mensagem. Uma característica deste tipo de rede é o *broadcasting*, ou a possibilidade de enviar uma mesma mensagem a várias estações, através de um código especial no campo de destinatário.

Outro exemplo de redes de difusão é a transmissão por satélites.



Serviços de comunicação

Define-se por serviço o conjunto de operações que uma camada é capaz de oferecer à camada superior. Ela define o que uma camada é capaz de executar sem se preocupar com a maneira como são executadas.

As camadas de uma arquitetura de rede podem oferecer diferentes classes de serviços às camadas superiores. Estes serviços podem ser orientados a conexão ou não (sem conexão).

- **Com conexão:** Análogo ao serviço telefônico, precisa iniciar uma conexão, para enviar a mensagem, e caracterizar término da conexão. Ela funciona como um canal virtual na realização do serviço.
- **Sem conexão:** Análogo ao sistema postal, cada mensagem contém o endereço do destinatário e pode ser encaminhada ao sistema independente de outras. O princípio básico é apenas enviar a mensagem.

Uma diferença nestes dois tipos de serviços, é que quando ele é orientado a conexão, as mensagens são sempre recebidas pelo destinatário na ordem de envio, o que pode não acontecer no serviço sem conexão.

Os serviços sem conexão são denominados serviços de **datagrama** (ex: correio eletrônico). Um datagrama é um pacote de informação que contém os dados do usuário, permitindo sua transferência numa rede de pacotes.

Outra característica de serviços de comunicação é a sua qualidade de serviço, podendo ser classificados em:

- **Confiável:** Os dados não são perdidos jamais (utilizam algum mecanismo de recuperação).
- **Não confiável:** Dados podem eventualmente se perderem e não recuperados pela camada em questão.

A implementação de serviços confiáveis geralmente utiliza mensagens de reconhecimento no destinatário, ou seja, uma confirmação que a mensagem foi recebida, o que nem sempre é desejável (performance, etc).

Um serviço é definido através de um conjunto de **primitivas** aos usuários, divididas em quatro classes:

- **Request:** Pedido enviado por uma entidade (um processo de uma camada) que solicita um serviço
- **Indication:** A entidade par é informada de uma solicitação de serviço
- **Response:** A entidade par responde ao pedido de serviço
- **Confirm:** A entidade solicitante é informada do resultado do serviço

Em pedidos não confirmados, apenas as duas primeiras primitivas são utilizadas (o pedido e a indicação). Ex:

1. Você disca o número do telefone de outra pessoa (CONNECT.request)
2. O telefone da outra pessoa toca (CONNECT.indication)
3. Ela atende dizendo “alô” (CONNECT.response)
4. Você ouve o “alô” da outra pessoa (CONNECT.confirm)
5. Você solicita uma informação(DATA.request)
6. A pessoa ouve a solicitação (DATA.indication)
7. Ela informa o que você solicitou (DATA.request)
8. Você ouve a informação (DATA.indication)

9. Você desliga seu telefone(DISCONNECT.request)
10. A pessoa ouve o sinal desconexão e faz o mesmo (DISCONNECT.indication)

Protocolos

O protocolo define um conjunto de regras que permitem especificar aspectos da realização do serviço. Basicamente indica o significado dos quadros, pacotes ou mensagens trocadas entres as entidades de cada camada. Existe um desacoplamento entre o conceito de protocolo e serviço, entre duas camadas.

Muitos protocolos conhecidos fazem parte do conjunto TCP/IP (*Internet Protocol Suíte*), como os seguintes:

- **IP:** Protocolo não orientado a conexão e não confiável, define o datagrama e endereçamento IP.
- **ICMP:** Utilizado para a verificação da rede, através de comandos PING, por exemplo.
- **IGMP (Internet Group Message Prococol):** Utilizado para a formação de grupos *multicast* em uma rede IP.
- **ARP (Adress Resolution Protocol):** Protocolo de resolução de endereços, associa endereços físicos com a numeração do IP. O RARP (*Reverse ARP*) faz o contrário, requisito o endereço IP de uma máquina a partir de um endereço físico MAC (*Media Access Control*).
- **TCP (Transmission Control Protocol):** Protocolo orientado a conexão e confiável,, possui informação de origem e destino (camada de transporte). Sua transmissão se dá através de segmentos.
- **UDP (User Datagram Protocol):** Não é orientado a conexão e nem confiável, mas muito leve.
- **Telnet:** Serviço de terminal virtual que permite sessões remotas sobre a rede (porta default 23).
- **FTP (File Transfer Protocol):** Serviço de transferência de arquivos (porta default 20 e 21).
- **DNS (Domain Name Service):** Serviço de tradução de nomes de hosts em endereços IP.
- **DHCP (Dynamic Host Configuration Protocol):** Controla e disponibiliza endereços IP para os clientes de rede que o requisitam. A vantagem é que o endereço IP não precisa ser fixo.
- **NFS (Network File System):** Sistema de arquivos remotamente acessível.
- **NNTP (Network News Transfer Protocol):** protocolo utilizado por programas clientes de notícias (que expiram).
- **SMTP (Simple Mail Transfer Protocol):** Protocolo para o envio de e-mails mais utilizado.
- **SNMP (Simple Network Management Protocol):** Gerencia redes conectadas em um único ponto.
- **POP3 (Post Office Protocol 3):** Protocolo para o recebimento de e-mails mais utilizado.
- **HTTP (HyperText Transfer Protocol):** Para transmissão de páginas Web (porta default 80).

Equipamentos

Um sistema de rede de computadores possui vários equipamentos (componentes) a sua disponibilidade:

- **(FIS) Mutiplexadores:** é um dispositivo cuja função é permitir que múltiplas estações de trabalho possam compartilhar uma linha de comunicação, pois a linha tem capacidade suficiente (banda) para suportar seu uso compartilhado. Com isso, multiplexadores reduzem o número de linhas de comunicação necessários, conseguindo uma redução nos custos pois diminuem o cabeamento necessário.
- **(FIS) Repetidores:** utilizados para estender a área de ação de uma rede, reforçando e regenerando os sinais físicos. São dispositivos bidirecionais, sem inteligência, e introduzem um atraso na propagação.
- **Hubs e Switches:** são dispositivos que provêem uma conexão central para as estações de trabalho, servidores e periféricos. Atuam como guardas de tráfego de dados, **evitando colisões e congestionamentos**. Estes dispositivos permitem que a rede cresça em número de equipamentos, até um limite aceitável de colisões, que depende do tipo de cabeamento e topologia da rede. Eles se encarregam de distribuir os sinais elétricos entre os vários equipamentos que compõem a rede, isolando os problemas de cada uma das estações e garantindo mais segurança e confiabilidade.
    - *(FIS) Hubs* ou **Concentradores**, trabalham com arquitetura de meio físico compartilhado. *Hubs* são multiplexadores inteligentes, possuem processador e um buffer onde armazenam os dados oriundos dos terminais

para envio posterior ao servidor (aonde é enviada a identificação do terminal), o que altera a velocidade de transmissão de dados. Os concentradores incluem um software de controle, com isso um grande número de linhas de baixa velocidade podem compartilhar um pequeno número de linhas de alta velocidade.

    - *(ENL) Switches* ou **Comutadores**, diferentemente dos *hubs* (barramento compartilhado), oferecem uma "linha comutada dedicada" a cada uma das suas conexões. Um *hub* que dispõe 10 Mbps, compartilha esta velocidade com todas as conexões, já o *switch* permitiria que cada conexão se comunicasse a velocidade total da LAN. Com isso os *switches* permitem comunicações paralelas, onde duas estações podem enviar seus dados em um mesmo intervalo de tempo sem riscos de colisões. Com isso evita à retransmissão e aumenta o rendimento da rede. O protocolo SNMP é usado pelo *switch* como uma das tarefas de gerenciamento.

- **(ENL) Bridges:** equipamentos que têm como finalidade segmentar uma rede local em várias sub-redes, e com isto conseguem diminuir o fluxo de dados. Os *Bridges* também podem converter padrões, como por exemplo de *Ethernet* para Token-Ring. Eles manipulam pacotes, não retransmitindo ruídos, erros, e pacotes mal formados. Fazem parte do nível de ligação lógico (MAC). São suas funções:
    - ler o endereço do pacote e retransmití-lo;
    - filtrar as mensagens, de modo que pacotes com erros não seja retransmitidos;
    - armazenam os pacotes quando o tráfego for muito grande;
    - memorizar os endereços lógicos dos dispositivos, e gerar lista dos dispositivos ligados a cada porta;
    - deve-se evitar *loops* ativos (envio de um pacote a dois *bridges*, que retransmitem vários *frames* repetidos).
- **(RED) Roteador:** existem duas atividades que são básicas a um roteador. São elas: a determinação das melhores rotas e o transporte dos pacotes. Determinar a melhor rota é definir por qual enlace uma determinada mensagem deve ser enviada para chegar ao seu destino de forma segura e eficiente. Para realizar esta função, o roteador utiliza dois conceitos muito importantes: o conceito de métrica e o conceito de tabelas de roteadores. Transportar os pacotes pela rede é uma função relativamente simples realizada pelos roteadores. Consiste em verificar o endereço de rede para quem a mensagem está destinada, determinar se conhece este endereço. E, por fim, traduzir para um novo endereço físico e enviar pacote.
- **(TRA) Gateways:** são equipamentos utilizados para se fazer a comunicação entre duas redes com arquiteturas diferentes. Na comunicação entre arquiteturas diferentes surgem diversos problemas, como por exemplo:
    - tamanho máximo de pacotes;
    - forma de endereçamento;
    - técnicas de roteamento;
    - controle de acesso;

Sistemas Distribuídos

Um sistema distribuído é formado por um conjunto de módulos processadores interligados por um sistema de comunicação (rede de computadores). Enquanto uma rede é construída para o compartilhamento de recursos, um sistema distribuído possui como principais metas a obtenção de um maior desempenho e confiabilidade.

Normalmente cada rede interligada em um sistema distribuído (*intranets*) é protegida por um firewall, pois o sistema como um todo pode ter a sua segurança comprometida, já que aumenta o número de entradas (e brechas) da rede, principalmente se existir um acesso à Internet.

Segurança

A segurança no universo computacional se divide em segurança lógica e segurança física, presente tanto em computadores *standalones* (PC's individuais) como em computadores ligados em rede (Internet ou *intranets*).

A **segurança física** deve atentar para ameaças sempre presentes, como incêndios, desabamentos, relâmpagos, alagamentos, problemas na rede elétrica, acesso indevido de pessoas ao CPD, treinamento inadequado de funcionários, etc. Medidas de proteção física, tais como serviços de guarda, uso de no-breaks, alarmes e fechaduras, circuito interno de televisão e sistemas de escuta são realmente uma parte da segurança de dados. As medidas de proteção física são freqüentemente citadas como "segurança computacional". O ponto-chave é que as técnicas de proteção de dados não têm serventia nenhuma se a segurança física não for garantida.

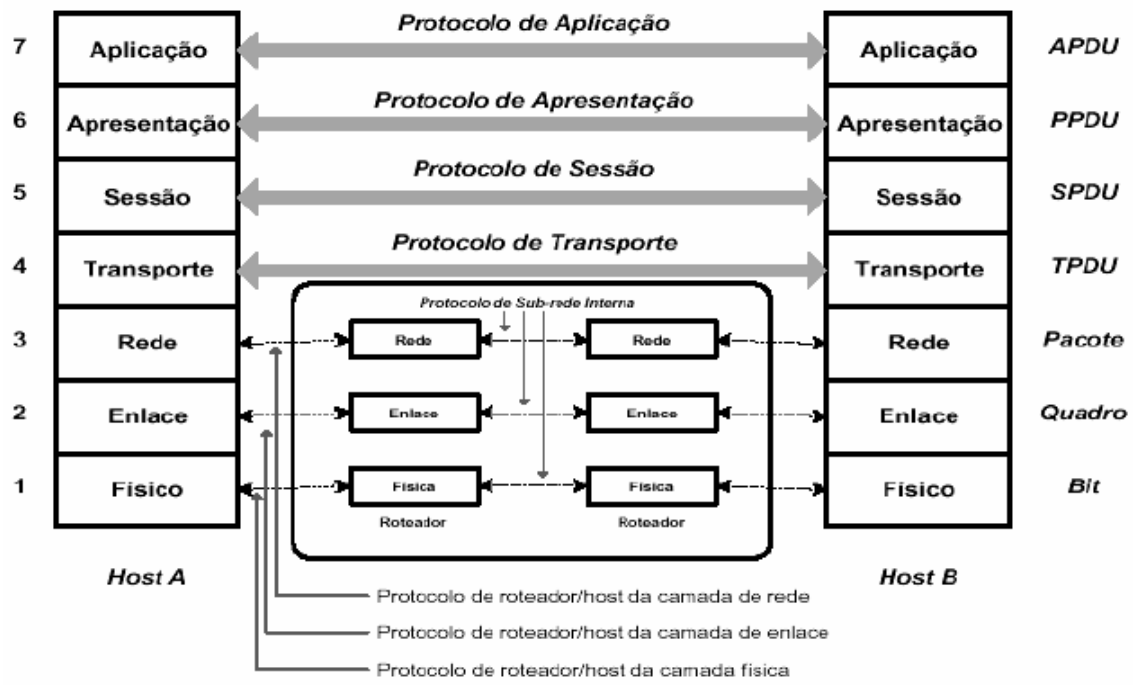
A **segurança lógica** requer um estudo maior, pois envolve investimento em softwares de segurança ou elaboração dos mesmos. Deve-se estar atento aos problemas causados por vírus, acesso de invasores de rede, programas de backup desatualizados (ou feito de maneira inadequada), distribuição de senhas de acesso, etc.. Um recurso muito utilizado para se proteger dos bisbilhoteiros da Internet (administrador de sistemas), é a utilização de um programa de criptografia que embaralha o conteúdo da mensagem, de modo que ela se torna incompreensível para quem não seja nem o receptor ou provedor da mesma.

Arquitetura OSI da ISO

O modelo OSI foi criado seguindo a filosofia das arquiteturas multicamadas, com os seguintes princípios:

- Cada camada corresponde a um nível de abstração necessário no modelo;
- Cada camada possui suas funções próprias e bem definidas;
- As funções de cada camada foram escolhidas pela definição dos protocolos internacionais;
- As fronteiras entre as camadas foram definidas de modo a minimizar o fluxo de dados nas interfaces;
- O número de camadas deve ser suficientemente grande para que funções distintas não sejam colocadas na mesma camada, e suficientemente pequeno para que não seja difícil o controle da arquitetura.

Além disso, o modelo foi projetado para redes geograficamente distribuídas (WAN).



Camada Física

É responsável pela transferência de bits num circuito de comunicação. A sua função é garantir que cada bit enviado de um lado será recebido do outro lado sem ter alterado o seu valor. Neste nível resolvem-se:

- os modos de representação dos bits 0 e 1 de maneira a evitar ambigüidades ou confusões (valor da tensão em volts, duração de cada sinal representando um bit, a codificação dos sinais, etc.)
- os tipos de conectores utilizados nas ligações (número de pinos, funções associadas a cada pino, etc.)
- como as conexões se estabelecem para a iniciação de um diálogo e como é feita a desconexão ao fim
- modo de transmissão adotado (unidirecional, bidirecional)
- modo de conexão adotado (ponto-a-ponto, multiponto)
- modo de tratamento dos erros (detecção, tratamento, etc).

Em resumo, esta camada relaciona-se às interfaces elétricas e mecânicas, preocupando-se com o transporte de dados (representados por bits) entre dois equipamentos terminais, via um suporte de transmissão. Alguns protocolos padrões são: IEEE

802, ISO 2110, etc. O **Ethernet** é um dos padrões mais populares para a conexão física de uma LAN. Utiliza um procedimento de compartilhamento de cabos denominado de CSMA/CD e trata das colisões de dados que podem ocorrer quando diferentes nós da rede tentam enviar dados simultaneamente. É chamada de topologia baseada em competição, pois as estações de trabalho competem pela banda passante do canal, representa um meio econômico de obter transmissão de alta velocidade (10 ou 100 Mbps), e surporta diversas configurações de cabeamento, sendo fácil de instalar.

Repetidores, multiplexadores e *Hubs* utilizam esta camada.

Camada de Enlace de Dados

Tem por função principal a transformação do meio de comunicação "bruto" em uma linha livre de erros de transmissão para a camada de Rede. Ela efetua esta função através do fracionamento das mensagens recebidas do emissor em unidades de dados denominadas quadros, que correspondem a algumas centenas de bytes. Estes quadros são transmitidos seqüencialmente e vão gerar quadros de reconhecimento enviados pelo receptor. Nesta camada, as unidades de dados são enriquecidas com um conjunto de bits adicional (no início e fim de cada quadro) de modo a permitir o reconhecimento destes.

Uma outra função desta camada é evitar uma alta taxa de envio de dados da parte do emissor no caso do sistema emissor não ter capacidade de absorver a informação à mesma taxa. Ela está dividida convencionalmente em duas subcamadas:

- **Controle de Acesso à Mídia (Media Access Control - MAC):** controla os meios pelos quais vários dispositivos compartilham o mesmo canal de transmissão.
- **Controle de Link Lógico (Logical Link Control - LLC):** estabelece e mantém links (enlaces) entre dispositivos em comunicação.

Com o objetivo de permitir um controle de erro eficiente, a camada de enlace decompõe as mensagens em porções menores denominadas **quadros** (*frames*), onde são adicionados códigos especiais de controle de erro. O controle de erros de transmissão é uma das funções mais importantes asseguradas pela camada de enlace. As responsabilidades desta camada são:

- a forma como os bits provenientes da camada Física serão agrupados em quadros (*frames*)
- os mecanismos de detecção e correção de erros implantados, uma vez que as informações trocadas através da camada física não são isentas de erros de transmissão, pelos fatores que já foram levantados
- mecanismos de controle de fluxo para limitar o volume de informação trocados entre fonte e destino
- a gestão das ligações entre as entidades de Rede
- mapeamento do endereçamento MAC, gravado na ROM do adaptador de rede (48 bits)

Alguns dos componentes desta camada são o *Bridge*, *Switch*, o *ISDN Router*, etc.

Camada de Rede

A camada de rede é responsável da gestão de sub-redes. A principal função da camada de rede é efetuar o encaminhamento dos pacotes trocados entre duas entidades oferecendo uma comunicação fim-a-fim.

Os caminhos a serem utilizados podem ser definidos em função de tabelas estáticas ou determinados dinamicamente no momento de cada diálogo em função das condições de tráfego, devendo ainda efetuar a gestão dos problemas de congestionamento pela presença de uma quantidade excessiva de pacotes de dados na rede.

A camada de rede é a camada mais baixa que trata a transmissão fim-a-fim. As duas funções essenciais da camada de rede são **roteamento** e **controle de congestionamento**. Além dessas, também é responsável por:

- multiplexação, endereçamento e seqüenciação
- mapeamento entre endereços de rede e endereços de enlace
- estabelecimento e liberação de conexões do serviço de rede
- transmissão de unidades de dados do serviço de rede (pacotes)
- segmentação e blocagem de SDUs/PDUs
- detecção e recuperação de erros

Os serviços da camada de rede foram projetados para serem independentes da tecnologia de sub-rede. Alguns de seus componentes são o *Roteador*, o *ATM Switch*, *Switch* nível 3, entre outros.

Camada de Transporte

Esta camada cria normalmente uma conexão de rede para cada conexão de transporte requerida pela camada de sessão, e determina também o tipo de serviço oferecida à esta. Ela implementa um verdadeiro diálogo fim-a-fim, ou seja, o programa executando no sistema fonte dialoga com o programa executando na máquina destino através dos cabeçalhos e informações de controle contidas nas mensagens deste nível.

Dado que esta camada é responsável do estabelecimento e término das conexões de rede, ela deve definir um mecanismo de endereçamento que permita a um sistema indicar com qual sistema ele deseja dialogar. Finalmente, ela deve implementar um mecanismo de controle de fluxo fim-a-fim para evitar que o sistema fonte envie mensagens numa taxa superior àquela com a qual o sistema destino pode consumi-las.

De maneira similar à camada de Rede, a de Transporte pode fornecer dois tipos de serviço: sem conexão e orientados à conexão. Ela também permite um "isolamento" em relação às camadas superiores, pois é o último dos níveis que seriam mais orientados a transporte efetivo das informações (e não à aplicações).

A camada de transporte **supre as deficiências** entre a qualidade de serviço (QOS) que a camada de sessão necessita e aquilo que a camada de rede pode oferecer. Alguns parâmetros são: o retardo (e probabilidade de falha) no estabelecimento de uma conexão, o desempenho (*thoughput*), retardo de trânsito, taxa de erros residuais, proteção, prioridade e resiliência (probabilidade de finalizar uma conexão espontaneamente).

Um componente usado a partir desta camada é o *Gateway*, e os protocolos usados são: TCP, ARP, RARP, etc.

Camada de Sessão

A principal função desta camada é oferecer aos seus usuários meios para o estabelecimento das conexões (sessões), de modo que estes possam trocar dados. Uma sessão é utilizada para permitir a conexão à distância a um computador (através de terminais, para uma transferência de arquivo, para o carregamento de programas, etc).

A camada de sessão é responsável dos estabelecimentos de sessões de diálogo para os usuários da rede. Além disso define a gestão do diálogo (se deve ser efetuado em modo uni ou bidirecional).

É importante notar aqui que a camada de Sessão oferece unicamente as ferramentas para a solução dos problemas de erros e incoerência por sincronização / resincronização. Na realidade, quem ativa estas ferramentas quando da ocorrência de um problema são as entidades das camadas superiores.

Camada de Apresentação

A Camada de Apresentação utiliza algumas funções freqüentemente necessárias de modo a poupar o usuário deste trabalho. Esta camada assume particularmente as funções associadas à sintaxe e à semântica dos dados transmitidos. Um exemplo típico das funções efetuadas por esta camada é a codificação da informação num padrão bem definido (ASCII, EBCDIC, etc.).

Esta camada pode ainda suprir outras funções associadas à compreensão dos dados, se utilizando do conhecimento do significado da informação para reduzir a quantidade de informação enviada, inclusive para implementar funções de confidencialidade e de autenticação.

Camada de Aplicação

A camada de Aplicação tem por função o gerenciamento dos programas de usuário (programas de aplicação) que executam em máquinas conectadas e utilizam o sistema de comunicação para a troca de informações.

Esta camada implementa um conjunto de protocolos diversificado e orientado a aplicações bem definidas. Exemplos são o protocolo de terminal virtual e de transferência de arquivo. Ela mantém o contato direto com os usuários da arquitetura de comunicação, abrindo caminho para os serviços oferecidos pelas camadas inferiores.

TCP/IP

Nos anos 70, o DoD (*US Department of Defense*), diante de um inventário de diferentes computadores que não podiam interagir, foi o pioneiro no desenvolvimento de protocolos de software para redes, que poderiam funcionar em mais de uma mar-

ca e modelo de computador. O principal conjunto estabelecido pelo DoD é o *Transmission Control Protocol/Internet Protocol* (TCP/IP). Estes protocolos são acordos sobre como devem ocorrer as transmissões nas redes.

A arquitetura Internet também é organizada em camadas, e composta por dois protocolos principais, o TCP (transporte fim-a-fim confiável) e o IP (encaminhamento de pacotes de dados através de várias sub-redes.

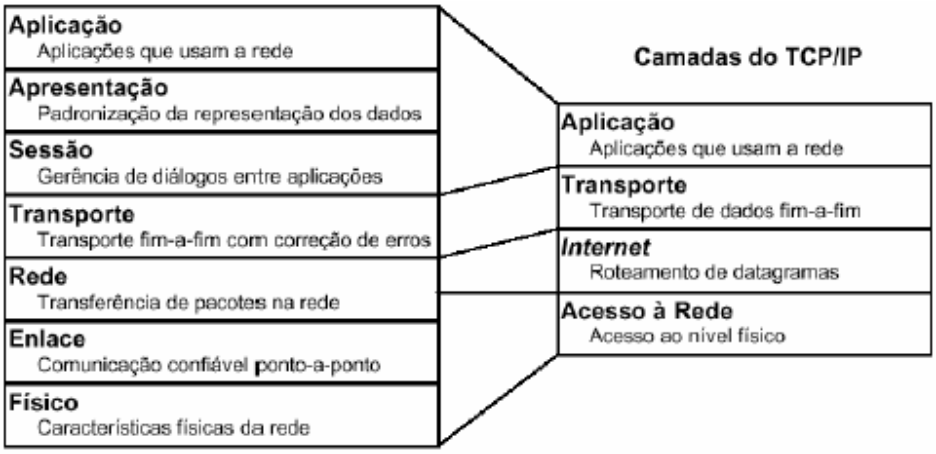
O conjunto TCP/IP pode oferecer um serviço relativamente confiável. Ao utilizar o protocolo TCP/IP, caso um datagrama TCP enviado por um Host (Computador) seja danificado em um segmento entre dois roteadores, antes de chegar ao Host destinatário, o computador remetente reenvia o datagrama porque não recebe uma confirmação de recebimento pelo destinatário.

O sucesso e a popularidade do protocolo não se deve apenas à imposição das agências militares americanas, mas também ao fato der ter sido o primeiro protocolo a atingir a importante meta da comunicação de dados com abrangência mundial:

- TCP/IP é um protocolo aberto, público e independente de equipamentos e de sistemas operacionais;
- TCP/IP não define protocolos para o nível físico, possibilitando sua implementação sobre uma grande variedade de protocolos já existentes, tais como: Ethernet, Token Ring e X.25;
- O esquema de endereçamento do TCP/IP permite designar univocamente qualquer máquina, mesmo em redes globais como a Internet;
- TCP/IP inclui protocolos do nível de aplicação que atendem à demanda de serviços imposta.

Camadas do TCP/IP

Não existe uma correspondência de camadas entre o modelo OSI e o TCP/IP que seja aceita universalmente, pois o TCP/IP foi criado com o compromisso da funcionalidade, e o OSI foi projetado academicamente.



Da mesma forma que no modelo de referência OSI, os dados descem a pilha de protocolos para chegar a rede e sobem para chegar às aplicações. Cada camada da pilha de protocolos adiciona um cabeçalho com informações de controle e trata o que recebe da camada superior como sendo dados.

Camada de Acesso à Rede (ou Camada de Interface)

É a mais baixa da hierarquia de protocolos TCP/IP. Os seus protocolos provêem meios para que os dados sejam transmitidos a outros computadores na mesma rede física. Esta camada pode abranger as três primeiras camadas do modelo de referência OSI: física, de enlace e de rede. Entretanto, a camada de acesso à rede do TCP/IP não define propriamente os protocolos para estes três níveis, mas sim como utilizar os protocolos já existentes para suportar a transmissão de um datagrama IP. Outros protocolos existem nessa camada, como o IPX (da Novell), que faz parte do protocolo IPX/SPX. As principais funções da camada de acesso à rede são: o encapsulamento de datagramas IP em frames para transmissão e a tradução de endereços IP em endereços físicos de rede. Estas duas funções apresentam implementações específicas para cada tipo de rede.

Camada Internet (ou Camada de Inter-Rede)

Fica exatamente sobre a camada de acesso à rede. O Internet Protocol (IP), é o coração desta camada. Ele provê um serviço básico de datagrama sobre o qual as redes TCP/IP são implementadas. Todos os protocolos das camadas superiores a esta fazem uso do protocolo IP.

Outros protocolos da Camada Internet são o ICMP (*Internet Message Control Protocol*) para a obtenção de informações sobre a rede, e o ARP (*Address Resolution Protocol*) para resolução de endereços IP.

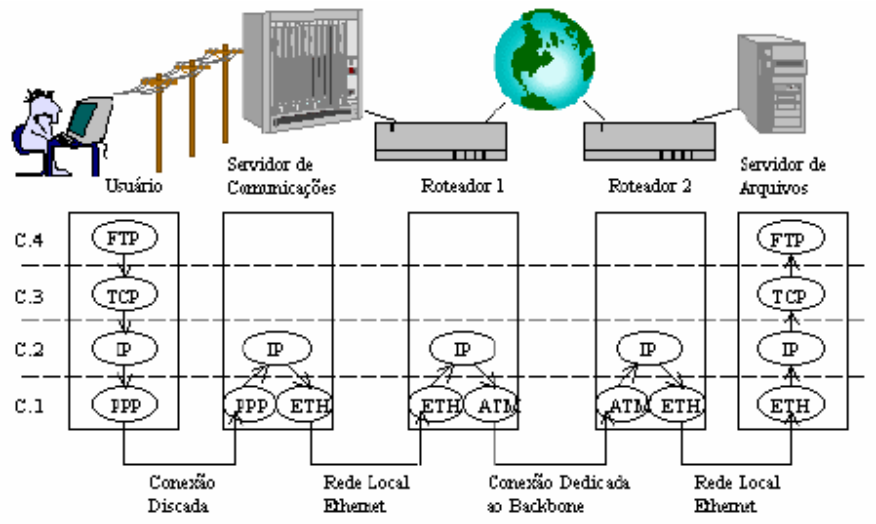
Camada de Transporte

Esta camada fim-a-fim está localizada exatamente sobre a camada Internet na hierarquia TCP/IP. Os principais protocolos desta camada são: TCP (*Transmission Control Protocol*), o UDP (*User Datagram Protocol*)

O TCP é um protocolo orientado a conexão com detecção e correção de erros fim-a-fim. O UDP é um protocolo não orientado a conexão e não confiável, sendo portanto muito leve. Ambos os protocolos passam dados entre as camadas de aplicação e Internet. Cada aplicação deve escolher o que melhor se adapta a sua natureza.

Camada de Aplicação

Fica no topo da pilha TCP/IP, incluindo todos os processos que utilizam serviços das camadas inferiores para transmitir dados através da rede. Alguns de seus protocolos são: Telnet, FTP, SMTP, DNS, NFS, etc.



Protocolo IP

O IP é um protocolo não orientado a conexão, ou seja, não existe negociação prévia de uma conexão para a transmissão de dados. Isto não impede a existência de protocolos orientados à conexão nas camadas superiores, mas eles deverão negociar o estabelecimento de conexões por si próprios. Além de ser não orientado à conexão, o protocolo IP também é não confiável, uma vez que não suporta mecanismos de detecção e recuperação de erros. Em outras palavras, o protocolo IP não verifica se um datagrama foi recebido corretamente, deixando esta responsabilidade para os protocolos das camadas superiores.

As principais funções do protocolo IP são:

- definir o datagrama IP, que é a unidade básica de transmissão de dados da arquitetura TCP/IP;
- definir o esquema de endereçamento IP;
- passar dados da camada de acesso à rede à camada de transporte;
- rotear datagramas IP;
- fragmentar e remontar datagramas IP

Protocolo TCP

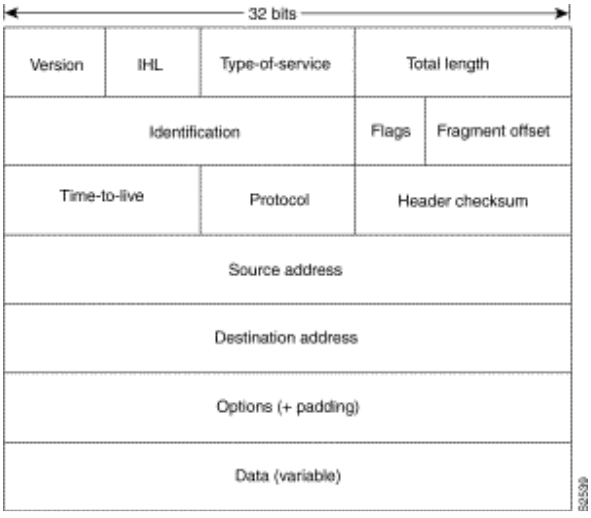
O protocolo TCP trabalha no mesmo nível que o protocolo UDP, mas oferece serviços mais complexos, que incluem controle de erros e de fluxo, serviço com conexão e envio de fluxo de dados. O TCP utiliza o mesmo conceito de porta do UDP. Para o TCP, uma conexão é formada pelo par (Endereço IP de Origem, Porta de Origem) e (Endereço IP de Destino, Porta de Destino).

O protocolo TCP oferece as seguintes características:

- Controle de Fluxo e Erro fim-a-fim (a comunicação é *full-duplex* fim-a-fim).

- Serviço confiável de transferência de dados.
- A aplicação necessita apenas enviar um fluxo de bytes.
- Desassociação entre quantidade de dados enviados pela aplicação e pela camada TCP.
- Ordenação de mensagens.
- Multiplexação de IP, através de várias portas.
- Opção de envio de dados urgentes.

Formato do Pacote IP



Versão: Atualmente IPV4. A IPV6 é cogitada.

IP Header Length: tamanho do cabeçalho

Type-of-Service: Nível de importância, etc.

Total Length: tamanho do pacote IP inteiro.

Identification: Número, para seqüências.

Flags: Controla fragmentação (3 bits).

Fragment Offset: Posição relativa do dado fragmentado do datagrama original.

Time-to-Live: Evita loops infinitos ao rotear um pacote (é um contador que é decrementado).

Protocol: Indica o protocolo (sup) a ser usado.

Header Checksum: Integridade do cabeçalho.

Source Address: indica o nó remetente.

Destination Address: indica o nó destinatário

Options: Permite que o IP suporte vários opções, como segurança.

Data: Contém as informações para a camada superior, que será utilizada pelo protocolo especificado.

Classes de Endereços IP

O endereço IP está dividido em 5 classes diferentes, identificadas pelos 4 primeiros bits:

- **Classe A** primeiro octeto inicia com 0xxx, ou 1 ate 126 decimal.
- **Classe B** primeiro octeto inicia com 10xx, ou 128 ate 191 decimal.
- **Classe C** primeiro octeto inicia com 110x, ou 192 ate 223 decimal.
- **Classe D** primeiro octeto inicia com 1110, ou 224 ate 239 decimal. (Classe para Multicast)
- **Classe E** primeiro octeto inicia com 1111, ou 240 ate 254 decimal. (Classe para Research)

O formato da máscara do endereço IP nas três primeiras classes é (N para *Network* e h para *Hosts*):

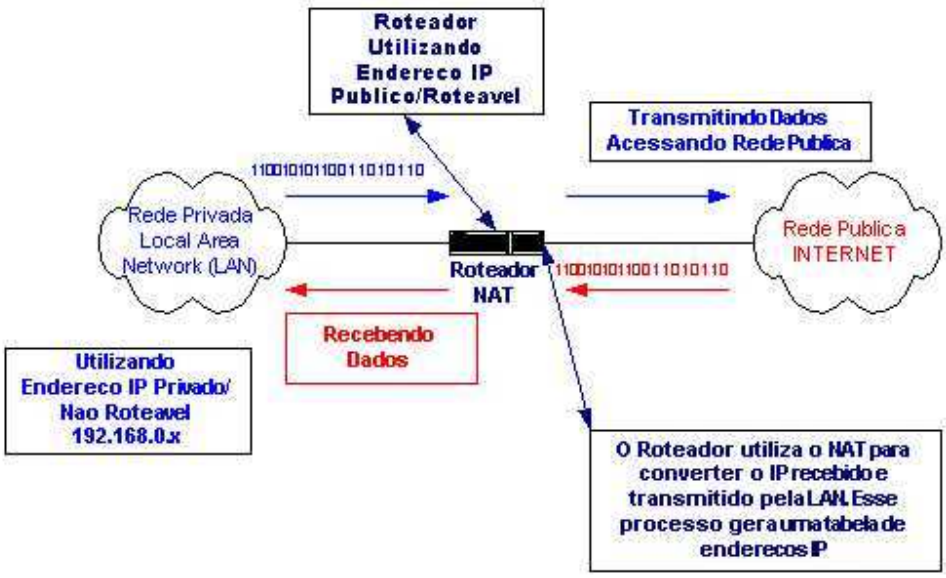
- Classe A (Redes grandes) -- NNNNNNNN.hhhhhhhh.hhhhhhhh.hhhhhhhh
- Classe B (Redes médias) -- NNNNNNNN.NNNNNNNN.hhhhhhhh.hhhhhhhh
- Classe C (Redes locais) -- NNNNNNNN.NNNNNNNN.NNNNNNNN.hhhhhhhh

Existem alguns endereços especiais, como o *Loopback* (127.0.0.1), e outros reservados das classes, como o 10.0.0.0/8 da Classe A (de 10.0.0.0 a 10.255.255.255) e o 192.168.0.0/16 da Classe C (até 192.168.255.255), não roteados para Internet.

- **Classe A:** como exemplo, o endereço 126.1.12.2 pertence à classe A. São possíveis 126 redes (o primeiro número vai de 1 a 126), e 16.777.214 hosts para a rede (os três últimos números, 256 * 256 * 256 – 2).

- **Classe B:** como exemplo, o endereço 128.126.12.34 pertence à classe B. São possíveis 16.384 redes (os dois primeiros números, de 128 a 191), e 65.534 hosts (para os dois últimos números, 256*256 – 2).
- **Classe C:** como exemplo, o endereço 192.168.0.34 pertence à classe C. Permite a existência de cerca de 2.097.152 redes (três primeiros números, variando de 192 até 233) e 254 hosts por rede (256 – 2).

Segue um exemplo de uma rede local utilizando um endereço IP privado Classe C para acessar a Internet:



Algoritmo de Transmissão de pacotes IP

```
1. Datagrama pronto para ser transmitido
2. Caso:
      Endereço Destino == Endereço Transmissor
            Entrega datagrama pela interface loopback (127.0.0.1)
            Fim
      Endereço de rede do destino == endereço de rede local
            Descobre o endereço físico do destino (ARP)
            Transmite datagrama pela interface correta
            Fim
      Endereço de rede do destino != endereço de rede local
            Verifica tabela de rotas
            Descobre rota que se encaixa com a rede destino
            Descobre o endereço físico do gateway (ARP)
            Transmite o datagrama para o gateway
            Fim
3. Fim
```

Algoritmo de Recepção de pacotes IP

```
1. Datagrama recebido da camada intra-rede, defragmentado e testado
2. Caso:
      End. Destino = End. do Host / outras interfaces do Host / Broadcast
```

```
            Passa datagrama para níveis superiores -> FIM
      Caso:
            Máquina que recebeu não é roteador
                  Descarta datagrama -> FIM
            Máquina é roteador (possui mais de uma interface IP)
            Caso:
                  Endereço IP destino = Rede IPcom interface direta
                        Descobre o endereço físico do destino (ARP)
                        Transmite datagrama pela interface respect. -> FIM
                  Caso Endereço de rede do destino endereço de rede local
                        Verifica tabela de rotas
                        Descobre o endereço físico do gateway (ARP)
                        Transmite o datagrama para o gateway -> FIM
3. Fim
```

Windows (2003/XP/2000/98)

A Microsoft (empresa fundada por Bill Gates e Paul Allen) criou e mantém o Windows, um sistema operacional comercial muito popular, que possui este nome por ter a sua interface baseada em janelas (*windows*).

As versões iniciais do Windows (até a versão 3.11 inclusive) rodavam em cima do DOS, um sistema operacional de 16 bits. As versões posteriores (Windows 95, 98, NT, etc.) suportam 32 bits, enquanto versões mais avançadas, como o Windows XP e o Windows Vista, estão preparadas para 64 bits.

Bibliotecas de ligações dinâmicas

O Windows possui arquivos que contém uma biblioteca de funções e procedimentos designados para serem chamados por outros programas rodando no sistema operacional. Estes arquivos, com a extensão dll (*dynamic linked libraries*), quando compilados, utilizam um mecanismo conhecido como ligação dinâmica. A ligação dinâmica permite que os procedimentos incluídos no arquivo sejam disponibilizados para outras aplicações.

Ao compilar um programa (por exemplo, em Delphi ou Visual Basic) que acessa um destes procedimentos, a aplicação varre o código fonte, lista todas as referências aos procedimentos que não são parte de sua própria biblioteca de procedimentos, e encontra as DLLs que os possuem.

Quando o programa for executado, o Windows carregará o arquivo de biblioteca de ligações dinâmicas que contém o procedimento referenciado. Todos os procedimentos públicos no arquivo DLL tornam-se acessíveis e os endereços de memória são especificados e dinamicamente ligados ao executável.

A técnica de ligação dinâmica possui vantagens sobre a ligação estática (onde todo o código fonte dos procedimentos é incorporado ao executável). Uma das vantagens é a economia de memória, pois a DLL é carregada apenas uma vez, mesmo que vários programas façam uso de seus procedimentos. Outra vantagem é a economia em espaço em disco, pelo mesmo motivo. Porém, estas vantagens podem se tornar um problema, se as DLLs possuírem muitos procedimentos não utilizados, pois estes seriam carregados em memória e ocupariam espaço em disco inutilmente.

Outra vantagem é que melhorias ou correções às bibliotecas de procedimentos podem ser feitas naturalmente, e os programas que utilizam os seus procedimentos não precisam ser recompilados.

Como desvantagem, pode ocorrer de um programa utilizar uma DLL desatualizada ou mesmo referenciar uma DLL inexistente no sistema (todas as bibliotecas devem ser copiadas junto com o executável).

Arquitetura Distribuída

Principais conceitos e componentes

Modelo Cliente Servidor

O modelo cliente / servidor foi criado tendo como base a descentralização dos dados e recursos de processamento, em oposição ao modelo centralizado (*mainframe*). Neste modelo existem uma ou mais máquinas que atuam como servidores (Servidor de Arquivos, Banco de Dados, Servidor de Impressão), e fornecem os serviços para outras máquinas clientes, através de uma rede de computadores.

- **Vantagens:** Podem ser utilizados equipamentos baratos, há uma divisão mais eficiente do trabalho, diminuição do tráfego da rede, maior versatilidade de sistemas operacionais e plataformas, maior segurança e integridade (fornecido por um gerenciador de banco de dados central), entre outras.
- **Desvantagens:** Aumento do custo administrativo de suporte à rede e banco de dados, maior dificuldade em encontrar pontos problemáticos do sistema, custo maior no hardware / software do servidor.

2 camadas

No modelo de 2 camadas toda a lógica do negócio fica no cliente. Em cada máquina cliente deve ser instalado o programa (desenvolvido em VB, Delphi, etc), que possui todas as regras de acesso a um banco de dados, por exemplo. Naturalmente, a camada de apresentação também fica no cliente..

A desvantagem do modelo de 2 camadas é a dificuldade de manutenção, pois as regras de negócio estão sempre mudando (evolução do mercado, alterações na legislação), exigindo alteração em todos os clientes.

3 camadas

O modelo em 3 camadas veio solucionar o problema das 2 camadas, retirando as regras de negócio do cliente e centralizando-as em um determinado ponto, chamado Servidor de Aplicações. O acesso ao banco de dados é feito através das regras contidas no Servidor de Aplicação.

- **Apresentação:** Esta camada fica no cliente, fornecendo a interface visual da aplicação. Alterações à ela são mais raras, mas exigem a atualização de todos os clientes.
- **Lógica:** São as regras de negócio, centralizadas em um Servidor de Aplicações, fáceis de manter.
- **Dados:** Os dados permanecem em um servidor de Banco de Dados.

N camadas

Outra evolução aos modelos Cliente / Servidor anteriores é o modelo de 4 camadas, que centraliza a camada de apresentação em único ponto, normalmente um servidor *Web*. Assim existe uma camada adicional chamada **Cliente**, que representa o navegador (*browser*) utilizado pelos clientes em suas estações.

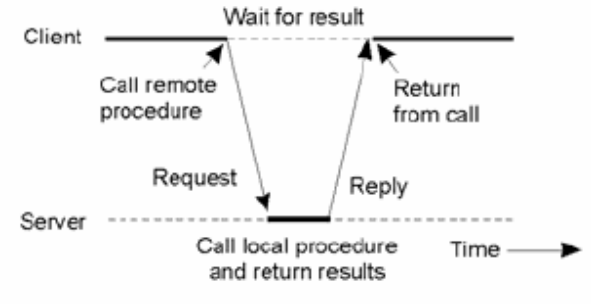
A camada de apresentação passa a ser um servidor *Web* composta de páginas HTML, ASP, JSP ou outras.

São evidentes as vantagens ao dimensionar um sistema em 3 ou mais camadas. Mas é preciso levar em conta os custos com os equipamentos, desempenho, etc. Se os servidores ficarem responsáveis por grande parte do processamento e uso de memória, os seus custos aumentam, e por isso o número de usuários, recursos do software, nível de exigência de cada usuário, e outros fatores devem ser considerados.

Chamadas remotas

O mecanismo de chamadas remotas RPC (*Remote Procedure Call*) integra o protocolo usado para comunicação Cliente / Servidor com as linguagens de programação convencionais, permitindo clientes comunicar com servidores através de chamadas de procedimentos.

A chamada remota segue o modelo da chamada local, sendo que o procedimento chamado executado em um processo diferente, normalmente em um computador diferente. Os passos são:



Cliente chama *stub* cliente

Stub cliente constrói mensagem, e chama SO

SO cliente envia mensagem para SO remoto

SO remoto entrega mensagem p/ *stub* servidor

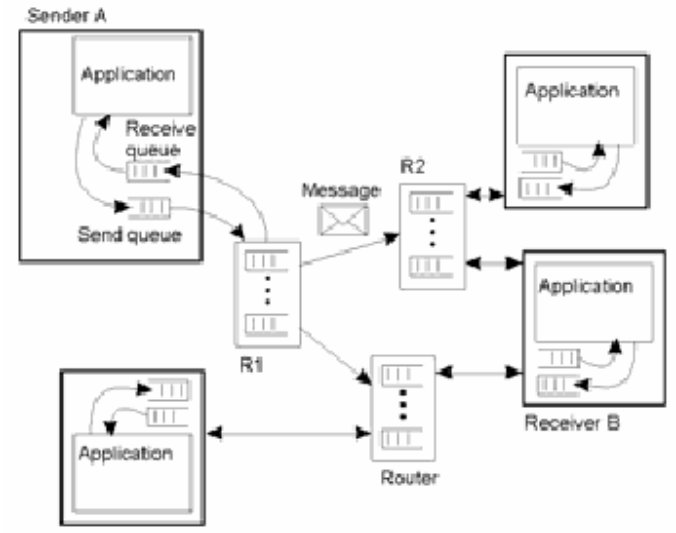
Servidor realiza trabalho, retorna para *stub*

SO servidor envia mensagem para SO cliente

SO cliente entrega mensagem para *stub* cliente

Stub cliente retorna resultado para o cliente

Sincronismo e filas de mensagens



Conceitos de Internet

Domain Name Service/System (DNS)

O DNS faz a conversão do nome do Domínio para o numero de IP e vise-versa.

Quando usamos a Internet estamos acostumados a lembrar dos nomes dos domínios ao contrario dos números dos IPs. Alias a grande maioria dos usuários nem mesmo sabem que um nome do domínio esta amarrado a pelo menos um numero de IP. Para facilitar a nossa vida o DNS faz o papel intermediador convertendo do nome do Domain para o numero do IP.

- **Exemplo:** www.google.com após convertido fica assim - numero IP: 216.239.35.100

Quando o endereço acima é digitado no navegador, uma mensagem é enviada ao DNS *server* solicitando a localidade do Domain. Se o mesmo existir, o DNS *server* retornará uma outra mensagem com o exato numero do IP para que o navegador consiga acessar o domínio desejado.

Os 13 DNS *Root Servers* são identificados pelas letras A-M, e a maioria está localizada nos Estados Unidos.

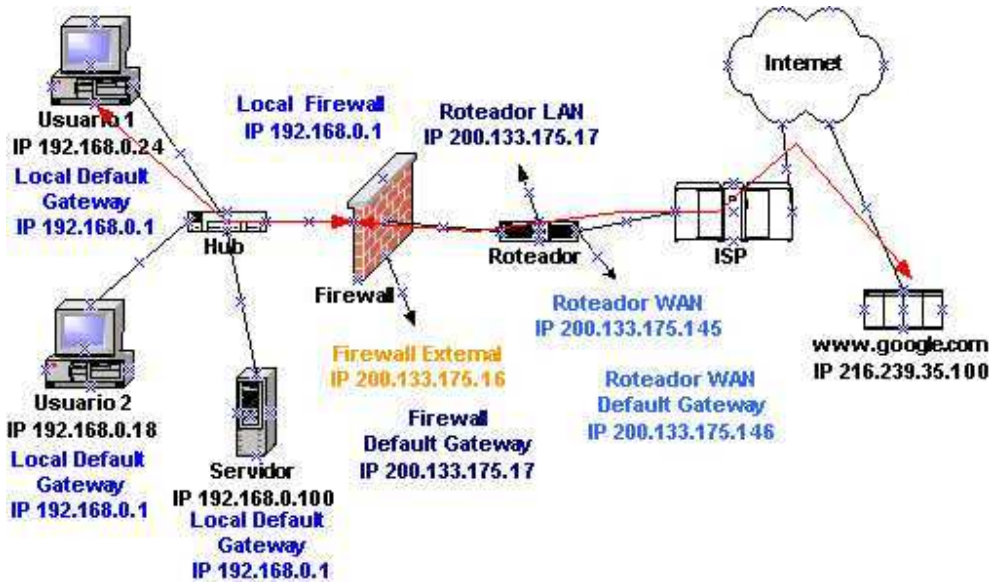
Default Gateway

Default Gateway é aquele que serve como intermediador entre redes, sempre presentes em redes locais com acesso a Internet. Quando estamos em uma rede local e queremos acessar um servidor local só é necessário descobrir o IP do servidor em enviar o pacote para o mesmo. Já quando queremos acessar a Internet já muda um pouco a figura. Neste caso não sabemos o IP do servidor que queremos acessar, então temos que fazer um pedido para o DNS como já estudamos anteriormente. O Default

Gateway entra em cena aqui, porque quando se emite um pacote IP que não pertence a rede local alguém tem que tomar providencia e encaminhá-lo para a devida rede que ele pertence. O seu papel é fazer a transição de uma lado para outro, no nosso caso ele manda o pacote que foi gerado pela rede interna para a rede externa Internet.

Default Gateway pode ser um roteador, um firewall, um computador etc., tudo isso vai depender do ponto de vista do layout de rede, certamente em muitos dos casos o roteador e o firewall serão Default Gateway em uma determinada rede. O roteador sendo Default Gateway da rede externa (ISP) e o firewall sendo Default Gateway da rede interna (rede local).

- **Exemplo:** O usuário na Maquina de IP 192.168.0.24 está acessando o google.com na Internet, e para isso ele teve que percorrer vários Default Gateway. O desenho tenta ilustrar o básico do que acontece quando se tenta acessar um endereço que esteja fora da rede local. Para que o acesso seja completado, o computador do cliente, firewall e o roteador devem estar com seus respectivos Default Gateway configurados. O desenho não está ilustrando o acesso ao DNS para adquirir o IP do google.com e nem como é obtido os endereços de IP internos do Gateway - ARP request.



Backbone

Backbones são 'portas' de acesso à Internet. No Brasil, poucas empresas são realmente proprietárias de *backbones* de Internet, como é o caso da EMBRATEL, da TELEFÔNICA e da IMPSAT. Estas empresas vendem a conexão com os *backbones* para os provedores de acesso à Internet. Estes provedores então, vendem o acesso discado via linha telefônica aos usuários finais.

O *backbone* ("espinha dorsal") é o trecho de maior capacidade dessa rede, com uma infra-estrutura de alta velocidade e que proporciona a conexão com várias redes menores. Localmente, o *backbone* é uma linha – ou conjunto de linhas – à qual as redes se conectam para formar uma WAN. Na Internet ou em outras WANs, o *backbone* é um conjunto de linhas com as quais as redes locais ou regionais se comunicam para interligações de longa distância.

Cada país tem uma rede principal para transmitir pacotes da Internet. Geralmente um país tem poucos *backbones* ou até mesmo um só. Um deles, pelo menos, é mantido pelo Estado. Mas existem também *backbones* em empresas particulares. O primeiro *backbone* da Internet criado no Brasil foi a RNP (Rede Nacional de Pesquisa), começou atendendo entidades, faculdades ou universidades que queriam se conectar à rede.

Intranet

As empresas descobriram que podem criar redes como a Internet, porém privadas, as *Intranets*, que cumprem o papel de conectar entre si filiais, departamentos, fornecedores, clientes, etc, mesclando (com segurança) as suas redes particulares de informação com a estrutura de comunicações da Internet.

Basicamente a montagem de uma intranet consiste em usar as estruturas de redes locais existentes na maioria das empresas, e em instalar um servidor Web. O servidor é a máquina que faz o papel de repositório das informações contidas na intranet, onde são buscadas as páginas HTML, e-mails ou outros tipos de arquivos.

A Intranet simplifica a interação do usuário, tornando fácil o acesso a aplicações e a informações estáticas e dinâmicas, não importando onde esteja ou qual a plataforma utilizada. Também viabiliza uma publicação em tempo real, com informação muito mais atual, favorecendo o desempenho dos funcionários da empresa. E finalmente, auxilia no processo de descentralização das informações, da distribuição de dados e do desenvolvimento de aplicações, além de permitir maior participação do usuário final na criação de aplicações.

Extranet

A rigor uma intranet pode operar apenas como uma rede corporativa dentro dos limites da empresa, porém pode ser vantajoso a ligação da intranet com a internet, neste caso chamada de extranet. A Extranet é a disponibilização da Intranet (no todo ou em parte) para ser acessada de fora da empresa.

O usuário doméstico que acessa a intranet de uma empresa não percebe que está na intranet. A diferença é percebida somente em termos de velocidade pelos funcionários, quando estes saem da intranet e acessam a internet do computador de sua seção.

- **Conectando a intranet à Internet:** Usa-se um roteador para encaminhar as informações da internet para a rede corporativa e vice-versa. Para obter esta ligação é necessário a contratação de um canal de dados junto a uma empresa. Também é preciso registrar um Domínio e obter um endereço IP.
- **Protegendo a Intranet:** É necessário proteger a Web corporativa contra a invasão de intrusos. Isso é feito por um computador dedicado que serve de porteiro, que supervisiona o trânsito das informações entre a intranet e a extranet e vice-versa. Esse computador geralmente roda um *firewall*.
- **VPN (Virtual Private Network):** é uma rede corporativa implementada através de redes públicas (Internet).

Firewall

Firewall pode ser definido como uma barreira de proteção, que controla o tráfego de dados entre um computador e a Internet (ou entre a rede onde o computador está instalado e a Internet). Seu objetivo é permitir somente a transmissão e a recepção de dados autorizados. Existem firewalls baseados na combinação de hardware e software e firewalls baseados somente em software. Este último é o tipo recomendado ao uso doméstico e também é o mais comum.

Há mais de uma forma de funcionamento de um firewall, que varia de acordo com o sistema, aplicação ou do desenvolvedor do programa. No entanto, existem dois tipos básicos de conceitos de firewalls:

- **Filtragem de pacotes:** é muito utilizado em redes pequenas ou de porte médio. Por meio de um conjunto de regras estabelecidas, esse tipo de firewall determina que endereços IPs e dados podem estabelecer comunicação e/ou transmitir/receber dados. Alguns sistemas ou serviços podem ser liberados completamente (por exemplo, o serviço de e-mail da rede), enquanto outros são bloqueados por padrão, por terem riscos elevados (como softwares de mensagens instantâneas, tal como o ICQ). O grande problema desse tipo de firewall, é que as regras aplicadas podem ser muito complexas e causar perda de desempenho da rede ou não serem eficazes o suficiente.
- **Filtragem de aplicação:** (exemplos de aplicação: SMTP, FTP, HTTP, etc) são instalados geralmente em computadores servidores e são conhecidos como proxy. Este tipo não permite comunicação direto entre a rede e a Internet. Tudo deve passar pelo firewall, que atua como um intermediador. O proxy efetua a comunicação entre ambos os lados por meio da avaliação do número da sessão TCP dos pacotes. Este tipo de firewall é mais complexo, porém muito seguro, pois todas as aplicações precisam de um proxy. Permite um acompanhamento mais preciso do tráfego.

A seguir são citadas as 3 principais razões para se usar um firewall:

- O firewall ajuda a impedir que a rede ou computador seja acessado sem autorização. Assim, é possível evitar que informações sejam capturadas ou que sistemas tenham seu funcionamento prejudicado pela ação de hackers;
- O firewall é um grande aliado no combate a vírus e cavalos-de-tróia, uma vez que é capaz de bloquear portas que eventualmente sejam usadas pelas "pragas digitais" ou então bloquear acesso a programas não autorizados;
- Em redes corporativas, é possível evitar que os usuários acessem serviços ou sistemas indevidos, além de ter o controle sobre as ações realizadas na rede, sendo possível até mesmo descobrir quais usuários as efetuaram.

Banco de Dados

Conceitos

Bancos de dados (ou bases de dados) são conjuntos de dados com uma estrutura regular que organizam informação. Essas estruturas costumam ter a forma de tabelas: cada tabela é composta por linhas e colunas. Informações utilizadas para um mesmo fim são agrupadas num banco de dados.

Segundo [KORTH], um Banco de Dados pode ser definido como uma coleção de dados inter-relacionados, cujo conteúdo informativo representa a qualquer instante, o estado de uma determinada aplicação.

Segundo [Navathe], um Banco de Dados é uma coleção de dados interligados por uma semântica comum.

Administração de Dados

Administrar dados significa envolvimento direto com o negócio. O Administrador de dados deve ser um profissional especialista em técnicas de modelagem de dados para sistemas OLTP (*On-Line Transactional Processing*) e sistemas OLAP (*On-Line Analytical Processing*), mas também, em um caso ótimo, ser conhecedor das principais regras que regem o negócio da empresa. Em outras palavras, uma mistura de Analista de Sistemas (especialista em modelagem) e Analista de Negócio. Algumas de suas responsabilidades são:

- Criação e manutenção de um modelo de dados corporativo
- Auditoria dos modelos de dados para eliminação de falhas de modelagem
- Padronização e completude na dicionarização dos dados
- Modelagem em relação ao escopo do sistema
- Integração do modelo analisado com os demais modelos de dados da corporação.

Em parceria com um DBA, é possível identificar também:

- Alterações no modelo de dados segundo a ótica de otimização da utilização de recursos do sistema gerenciador de banco de dados (SGBD) utilizado no sistema
- Conformidade com o Padrão de Banco de Dados (padrão para nomes de tabelas, atributos, etc.) adotado pela corporação, e otimização do modelo de dados, objetivando aumento de performance

Sistemas Gerenciados de Bancos de Dados (SGBDs)

Em sistemas computacionais, bases de dados são geridas por um sistema gestor de bancos de dados, ou SGBD. A apresentação dos dados pode ser semelhante à de uma planilha eletrônica, porém os sistemas de gestão de BDs possuem características especiais para o armazenamento, classificação e recuperação dos dados:

- **Manipulação de Dados:** organizar o conteúdo dos dados inserindo, atualizando, deletando e recuperando dados;
- **Definição de Dados:** estruturar os elementos de dados em esquemas lógicos e físicos;
- **Restrições de Integridade:** garantir a segurança, integridade e concorrência dos dados.

Administrar um banco é, de maneira simplista, instalar, configurar, monitorar e solucionar problemas de um SGBD. Esmiuçando este conceito, um Administrador de Banco de Dados tem as seguintes responsabilidades:

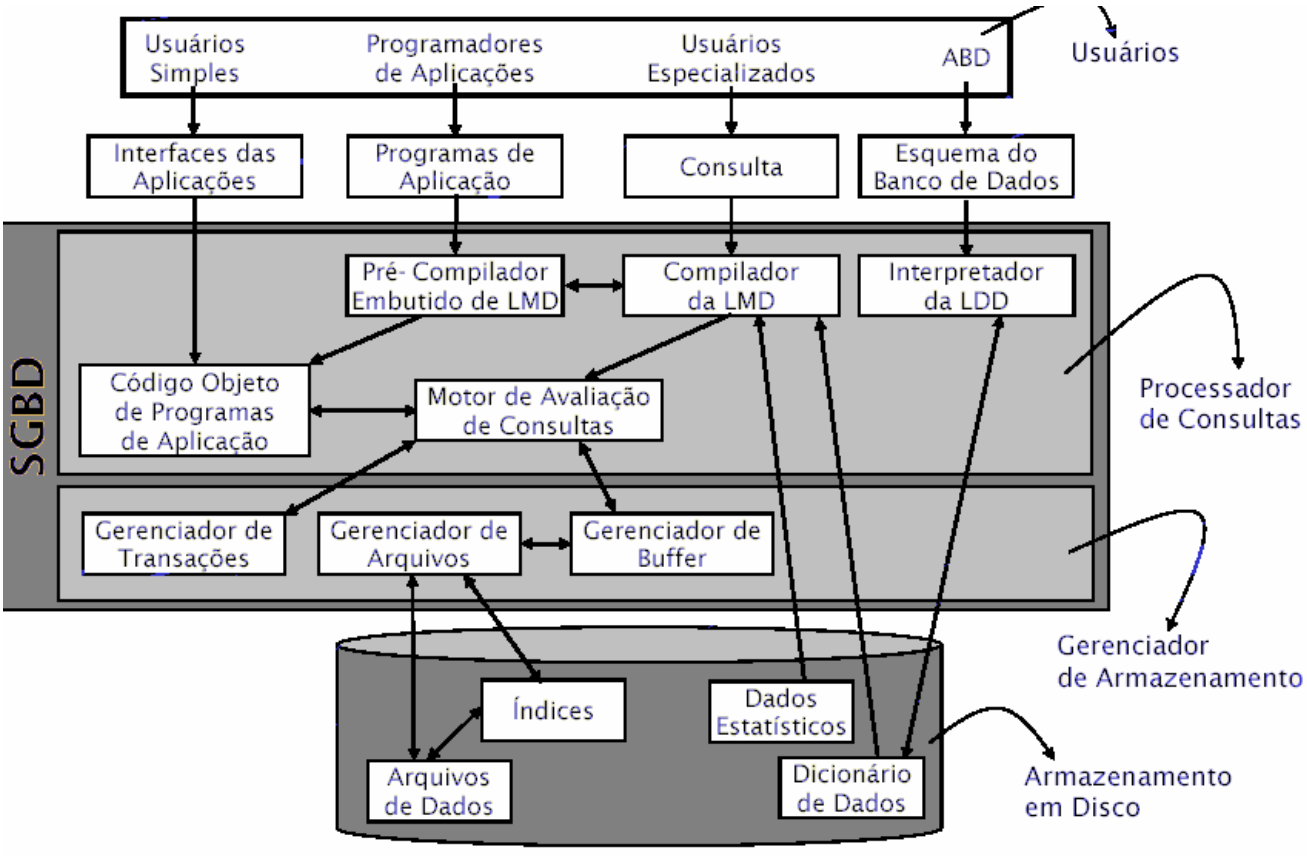
- Projeto lógico do banco de dados: criação do esquema lógico usando a DDL
- Definição de checagem de segurança e integridade
- Decisão de como os dados são representados na base de dados armazenada
- Projeto físico da base de dados
- Definição de procedimentos de recuperação
- Monitoração do desempenho

- Contato com usuários para averiguação de disponibilidade dos dados por eles requisitados e ajuda na determinação e resolução de problemas
- Ajustes apropriados à medida que ocorram mudanças de requisitos

A administração deve prever a utilização do SGBD ao longo de vários anos, garantindo a ausência de problemas físicos futuros que impeçam a disponibilidade dos dados.

Segundo [SILBERSCHATZ], o Sistema Gerenciador de Banco de Dados é uma coleção de dados inter-relacionados e uma coleção de programas para acesso a esses dados, proporcionando um ambiente conveniente e eficiente para armazenamento e recuperação de informações. A estrutura geral do SGBD é dividida nos seguintes componentes funcionais:

- **Processador de consultas**:
  - Compilador e pré-compilador da Linguagem de Modelagem de Dados
  - Interpretador da Linguagem de Definição de Dados
  - Motor de Avaliação de consultas.
- **Gerenciador de armazenamento**:
  - Gerenciador de Integridade e Autorização
  - Gerenciador de Transações
  - Gerenciador de Arquivos
  - Gerenciador de Buffer
- **Estruturas de Dados da Implementação do sistema físico:**
  - Arquivos de Dados
  - Dicionários de Dados
  - Índices
  - Dados estatísticos



Linguagem de definição de dados e Linguagem de manipulação de dados

Os elementos da DML (*Data Manipulation Language* - Linguagem de Manipulação de Dados). A DML é um subconjunto da linguagem usada para selecionar, inserir, atualizar e apagar dados. No nível físico precisa-se definir algoritmos que permitam um acesso eficiente aos dados. Nos níveis mais altos de abstração, é dada ênfase à facilidade de uso, fornecendo uma interação humana eficiente com o sistema.

A DDL (*Data Definition Language* - Linguagem de Definição de Dados) permite ao usuário definir tabelas novas e elementos associados. A maioria dos bancos de dados de SQL tem extensões proprietárias no DDL. O resultado da compilação de comandos de uma DDL é um conjunto de tabelas que são armazenadas em um arquivo chamado dicionário de dados. Este dicionário de dados contém "dados sobre dados", e é consultado antes que os dados sejam lidos ou modificados no sistema de banco de dados.

Dicionário de dados

O Dicionário de dados armazena os metadados relativos à estrutura do banco de dados. Ele é muito usado, e portanto grande ênfase é dada ao desenvolvimento de um bom projeto com uma implementação eficiente do dicionário.

Uma notação bastante utilizada para representar o dicionário de dados é sublinhar o atributo identificador (chave), e os tipos de atributos podem ser T (texto), I (inteiro), S/N (valor lógico), D (data), entre outros.

Atributo	Descrição	Tipo
Razão Comercial	Razão Comercial ou nome, em caso de pessoa física	T(50)
Pessoa Jurídica	Identifica se a pessoa é jurídica ou não.	S/N
CGC/CPF (0,1)	CGC (Código Geral do Contribuinte) para pessoa jurídica ou CPF (Código de Pessoa Física)	T(14)
Inscrição Estadual(0,1)	Inscrição Estadual quando a pessoa for jurídica	T(14)
Endereço	Endereço residencial/comercial	T40
Complemento	Complemento do endereço	T(30)
Bairro	Bairro onde reside	T(30)
CEP	Código de Endereçamento Postal da rua	I(8)
Telefones (0,2)	Telefones de contato	T(20)
Fax	Fax de contato	T(20)

Arquitetura de banco de dados

São três os modelos abordados na arquitetura ANSI/X3/SPARC:

- **Modelo conceitual:** definido pelo administrador de dados, e refere a uma visão lógica e global dos dados, ele deve abordar todos os objetos do sistema de informação da empresa independente da aplicação.
- **Modelo externo:** definido pelo administrador de aplicação. Visa um subconjunto do modelo conceitual para uma aplicação específica da empresa, podendo existir várias visões externas.
- **Modelo interno:** definido pelo Administrador de Banco de Dados. Se preocupa com a implementação física dos dados através de um banco de dados.

O **Projeto Lógico** do Banco de Dados engloba o Modelo conceitual e externo. O **Projeto Físico** consiste do Nível Interno.

Bancos de dados relacionais

Os bancos de dados relacionais são indiscutivelmente os mais utilizados atualmente. Alguns conceitos são:

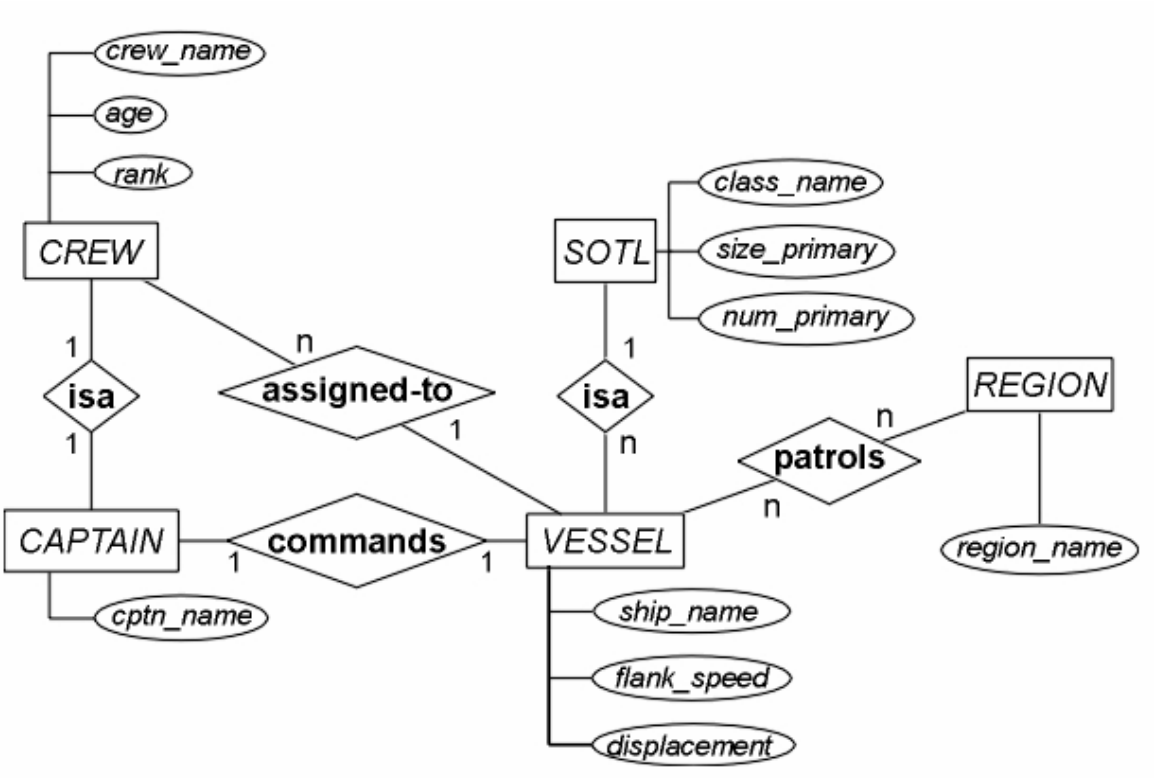
- **Tabelas:** são as relações do banco, representadas como uma tabela de dados, composta por colunas.
- **Colunas:** cada coluna na tabela possui um nome único, e contém um tipo de dados associados. Também são denominadas campos ou atributos da tabela.
- **Linhas:** cada linha em uma tabela representa um registro diferente. Por causa do formato tabular, todas elas têm os mesmos atributos. As linhas também são comumente chamadas de registros ou tuplas.

- **Valores:** cada linha consiste em um conjunto de valores individuais que correspondem a colunas. Cada valor deve ter o tipo de dados especificado pela sua coluna.
- **Chaves:** são utilizadas para identificar um registro específico, geralmente feito através de um valor inteiro único. Um número artificialmente gerado é a melhor forma de garantir a unicidade da chave, pois poucas informações reais possuem essa propriedade.
- **Relacionamentos:** as chaves estrangeiras representam um relacionamento entre dados em duas tabelas. Existem três tipos básicos de relacionamentos em um banco de dados relacional:
    - **um para um:** significa que há uma de cada coisa no relacionamento.
    - **um para muitos:** uma linha em uma tabela é vinculada a muitas linhas na outra tabela.
    - **muitos para muitos**: muitas linhas em uma tabela são vinculadas a muitas linhas de outra tabela. Esse tipo de relacionamento normalmente obtém uma tabela inteira para si próprio, logo uma terceira tabela que só contém as chaves das outras tabelas como chaves estrangeiras em pares serve para mostrar quais registros estão associados.
- **Metadados:** um banco de dados é autodescritível, e contém uma descrição de sua própria estrutura como parte integrante do mesmo. Metadados são armazenados na forma de tabelas (tabelas do sistema).

Modelagem de Dados

Modelo Entidade-Relacionamento

O Modelo de Dados, também conhecido como E-R (Entidade-Relacionamento) é uma forma de representação gráfica do conhecimento que se tem sobre o ambiente. Mostra uma visão estática das informações de interesse (entidades) e dos vínculos (relacionamentos) entre elas. Serve para comunicar o analista de dados e o usuário. É considerado um modelo **semântico**.



- **Entidade (Retângulo):** É algo real ou abstrato, sobre o qual nos interessa armazenar dados. Entidades fracas (que dependem de outras) são representadas por um retângulo duplo.
- **Atributo (Elipse):** É um dos itens de dados que armazenamos sobre uma entidade. Atributos multivalorados são representados por elipses duplas.

- **Relacionamento (Linhas):** Um relacionamento bi-direcional é uma relação entre dois tipos de entidades, existindo também o multi-direcional, auto-relacionamento, etc.
- **Chave de Identificação:** Atributo (ou conjunto de) cujo valor individualiza uma ocorrência da entidade. Por exemplo, para identificar a entidade EMPREGADO, o atributo-chave é a MATRÍCULA.
- **Domínio:** São os possíveis valores que um atributo pode assumir. Ex.: SEXO = [M | F]
- **Ocorrência:** O número de vezes que determinado atributo aparece em outra entidade.

Existem entidade independentes (a chave está em seus atributos), dependente (chave depende de outra) e associativa (a chave é obtida através da concatenação das chaves de identificação das entidades que ela associa. Representações para esses s outras características do banco surgiram em um modelo estendido (Teorey).

Modelo Relacional

Um banco de dados relacional consiste em uma coleção de tabelas, cada uma das quais com um nome único. Uma linha em uma tabela representa um relacionamento entre um conjunto de valores. Uma vez que essa tabela é uma coleção de tais relacionamentos, há uma estreita correspondência entre o conceito de tabela e o conceito matemático de relação, a partir do qual se origina o nome desse modelo de dados. Logo o Modelo Relacional está formalmente baseado na Álgebra Relacional.

Seguindo a terminologia do modelo relacional, tratamos os nomes dessas colunas como atributos. Para cada atributo há um conjunto de valores permitido, chamado domínio do atributo em questão. Como as tabelas em essência são relações, podemos usar os termos matemáticos relação e tupla no lugar de tabelas e linhas.

Três objetivos básicos motivaram fortemente as pesquisas que resultaram no modelo relacional:

- **Independência dos dados:** definição clara e precisa dos limites entre os aspectos físicos e lógicos de um gerenciador de banco de dados.
- **Comunicabilidade:** permitir um modelo estrutural simples de forma que usuários de várias categorias tivessem um entendimento comum dos dados e pudessem se comunicar através do banco de dados.
- **Linguagem de alto nível:** existência de uma linguagem que permitisse a manipulação de um conjunto de dados através de apenas um simples comando.

Atualmente esses objetivos já se encontram incorporados ao cotidiano do ambiente de desenvolvimento de sistemas. Já passa despercebido, por exemplo, o conceito de independência dos dados, pois todos os bancos de dados possuem essa característica. Mas além destas existem as 12 regras de Codd.

12 Regras de Codd

Um banco de dados, para que seja considerado "totalmente relacional", deve atender as 12 regras definidas por E. F. Codd, o criador do modelo relacional para banco de dados.

As doze regras de Codd estão baseadas na regra zero, que determina o seguinte: "Qualquer sistema considerado, ou que deseja ser, um sistema gerenciador de banco de dados relacionam deve ser capaz de gerenciar, por completo, bases de dados através de sua capacidade relacional". Essa regra determina que um SGBDR não permite exceções quanto ao modelo relacional de gerenciamento de bases de dados. As 12 são as seguintes:

- **Regra 1 - Representação de valores em tabelas:** Todas informações do BD relacional são representadas de forma explícita no nível lógico e exatamente em apenas uma forma - por valores em tabelas.
- **Regra 2 – Acesso Garantido:** Cada um e qualquer valor atômico (datum) em um banco de dados relacionam possui a garantia de ser logicamente acessado pela combinação do nome da tabela, do valor da chave primária e do nome da coluna.
- **Regra 3 – Tratamento sistemático de nulos:** Valores nulos devem ser suportados de forma sistemática e independente do tipo de dado para representar informações inexistentes e / ou inaplicáveis.
- **Regra 4 – Dicionário de dados ativo baseado no modelo relacional:** A descrição do banco de dados é representada no nível lógico da mesma forma que os dados ordinários, permitindo que usuários autorizados utilizem a mesma linguagem relacional aplicada aos dados regulares.
- **Regra 5 – Linguagem Detalhada:** Um sistema relacional pode suportar várias linguagens e várias formas de recuperação de informações. Entretanto, deve haver pelo menos uma linguagem, com uma sintaxe bem definida e ex-

pressa por conjuntos de caracteres, que suporte de forma compreensiva todos os seguintes itens: definição de dados, definição de *views*, manipulação de dados (interativa e embutida em programas), restrições de integridade, autorizações e limites de transações.

- **Regra 6 – Atualização de Views:** Todas as visões ("views") que são teoricamente atualizáveis devem também ser atualizáveis pelo sistema.
- **Regra 7 – Atualização de alto nível:** A capacidade de manipular um conjunto de dados (relação) por de um simples comando deve-se estender às operações de inclusão, alteração ou exclusão de dados.
- **Regra 8 – Independência Física:** Programas de aplicação permanecem logicamente inalterados quando ocorrem mudanças no método de acesso ou na forma de armazenamento físico.
- **Regra 9 – Independência Lógica:** Mudanças nas relações e nas *views* provocam pouco ou nenhum impacto nas aplicações.
- **Regra 10 – Independência de Integridade:** As aplicações não são afetadas quando ocorrem mudanças nas restrições de integridade.
- **Regra 11 – Independência de Distribuição:** As aplicações não são logicamente afetadas quando ocorrem mudanças geográficas dos dados. Devem permanecer inalterados quando são distribuídos em meios ou máquinas diferentes.
- **Regra 12 – Não Subversão:** Se um sistema possui uma linguagem de baixo nível, essa linguagem não pode ser usada para subverter as regras de integridades e restrições definidas no nível mais alto.

Além dessas doze regras básicas, o modelo relacional também define nove regras estruturais que tratam da definição de chaves primárias, chaves estrangeiras, *views*, tabelas etc.; dezoito regras de manipulação que definem as operações de "*join*", "*union*", "*division*" etc.; e três regras de integridade: integridade de entidade, integridade referencial e a capacidade de definir outras regras de integridade sem introduzir dependência estrutural. A integridade de entidade define que uma chave primária não pode ter valores duplicados ou nulos.

A integridade referencial determina que o valor de uma chave estrangeira deve ter obrigatoriamente correspondência em uma chave primária de uma outra relação.

Álgebra Relacional

A álgebra relacional é uma linguagem de consultas procedural, que consiste em um conjunto de operações tendo como entrada uma ou duas relações e produzindo, como resultado, uma nova relação.

- **Seleção (σ):** seleciona tuplas que satisfaçam uma determinada condição. O resultado é uma relação com a mesma estrutura da tabela original, e que contém as linhas que satisfazem à condição.
- **Projeção (π):** retorna uma relação apenas com os atributos selecionados. Duplicatas são eliminadas.
- **União ($\cup$):** une dois conjuntos de relações. Ex: $\pi_{\text{nome}}(tab\_clientes) \cup \pi_{\text{nome\_cliente}}(tab\_devedor)$
- **Interseção ($\cap$):** encontra as tuplas que estão tanto em uma relação quanto em outra.
- **Diferença (-):** encontra as tuplas que estão em uma relação, mas não em outra.
- **Produto Cartesiano (x):** combina as informações de duas relações, contendo todos os pares de tuplas possíveis. O número de tuplas resultante será o produto entre o número de tuplas de cada relação.
- **Junção ($\otimes$):** esta operação fundamental une duas relações através de uma coluna em comum entre elas, efetivando os relacionamentos entre as entidades de um banco de dados.
- **Divisão ($\div$):** usada quando a consulta emprega a frase "para todos", pois responde perguntas do tipo "quais fornecedores fornecem todas as peças?". Útil em relacionamentos "muitos para muitos".

Normalização

Normalização é uma técnica de projeto amplamente utilizada no desenho de bancos de dados relacionais.

A teoria da normalização é baseada no conceito de formas normais. Uma tabela relacional é dita estar numa determinada forma se ela satisfaz um certo conjunto específico de restrições.

Normalização é o processo de remoção de dados redundantes de tabelas relacionais, através de sua decomposição em tabelas menores. O objetivo da normalização é criar um conjunto de tabelas relacionais livres de dados redundantes e que pode ser modificada de forma consistente e correta.

Dependência Funcional

Um atributo Y é funcionalmente dependente de um atributo X se cada valor de X tenha associado a ele precisamente um valor de Y. Quando o atributo X é uma chave primária, então todos ao atributos são, por definição, dependentes de X, pois não podem existir dois registros com o mesmo valor para X.

Notação: R.x → R.y (lê-se a coluna x da tabela relacional R funcionalmente determina (identifica) a coluna y.

A dependência funcional pode ser classificada em:

- **Total:** um atributo é totalmente dependente de outro se ele for funcionalmente dependente do outro e não dependente de um subconjunto de outro
- **Parcial:** um atributo é parcialmente dependente de outro se ele for funcionalmente dependente de um subconjunto de outro

Considere a tabela abaixo, onde a chave primária é formada pelos atributos código-func + código-curso. O atributo avaliação é dependente total da chave composta. Já o atributo descrição-curso tem dependência parcial com relação à esta chave, pois depende somente de parte dela, ou seja, de código-curso.

código-func	*código-curso*	*descrição-curso*	*avaliação*	*data-conclusão*
00001	ENG01	ENG. CIVIL	A	01/01/2005

Primeira Forma Normal

Definição: Uma tabela relacional está na primeira forma normal se todos os valores das colunas são atômicos, ou seja, assumem apenas um único valor.

FORNECIMENTO DE PEÇAS				
Código Fornecedor	*Status Cidade*	*Cidade*	*Código Peça*	*Quantidade*
S1	20	LONDRES	P1	300
S1	20	LONDRES	P2	200
S1	20	LONDRES	P3	400
S1	20	LONDRES	P4	200
S1	20	LONDRES	P5	100
S1	20	LONDRES	P6	100
S2	10	PARIS	P1	300
S2	10	PARIS	P2	400
S3	10	PARIS	P2	200
S4	20	LONDRES	P2	200
S4	20	LONDRES	P4	300
S4	20	LONDRES	P5	400

- Elimine atributos multivalorados criando tantos conjuntos de entidades quantos forem os atributos multivalorados

A tabela ao lado anteriormente possuía valores multivalorados para Código-Peça (P1,P2,P3,P4,P5) e Quantidade (300,200,400,200,100). Agora está na forma 1FN.

Entretanto, ela ainda contém dados redundantes, pois a informação da CIDADE e STATUS CIDADE precisa ser repetida para todas as peças.

Não é possível cadastrar um certo fornecedor para uma cidade até que ele forneça uma peça. Isso não é desejável.

Segunda Forma Normal

Definição: Uma tabela relacional está na segunda forma normal se está na primeira forma normal e todas as colunas não-chave são totalmente dependentes da chave primária inteira.

FORNECEDOR		
Código Fornecedor	*Status Cidade*	*Cidade*
S1	20	LONDRES
S2	10	PARIS
S3	10	PARIS
S4	20	LONDRES

- Crie tabelas separadas para conjuntos de valores que se referem a múltiplos registros e relacione essas tabelas através de uma chave estrangeira

As informações CIDADE e STATUS CIDADE não são totalmente dependentes da chave primária total, por isso foi criada a tabela FORNECEDOR para guardar esses dados, tendo como chave primária o Código Fornecedor.

Note que ainda há o problema de se relacionar o Status com a Cidade, pois deve existir obrigatoriamente um fornecedor nela.

Terceira Forma Normal

Definição: Uma tabela relacional está na terceira forma normal se está na segunda forma normal e todas as colunas não-chave são funcionalmente dependentes apenas da chave primária.

CIDADE	
Cidade	***Status Cidade***
LONDRES	20
PARIS	10

- Crie tabelas separadas para colunas que não dependem da chave primária.

Foi criada a tabela CIDADE com o nome da cidade como chave primária. Desta forma, a tabela FORNECEDOR só precisa guarda o Código Fornecedor (chave) e o valor da Cidade, que se relaciona com esta tabela.

Um esquema de dados nessa situação pode facilmente lidar com o projeto de um banco de dados para uma empresa inteira.

Outras Formas Normais

Em um relacionamento muitos-para-muitos, entidades independentes não podem ser armazenadas na mesma tabela. A quarta forma normal (4FN) lida com este problema, criando tabelas de relacionamentos apenas com chaves primárias e estrangeiras, removendo entradas duplicadas em outras tabelas.

A quinta forma normal (5FN) afirma que deve ser possível reconstruir a tabela original a partir das tabelas em que ela foi dividida. O benefício de aplicar esta regra é assegurar que nenhuma coluna estranha ao contexto foi criada e que todas as tabelas criadas são tão grandes quanto necessariamente precisam ser.

É uma boa prática aplicar estas regras, mas, a menos que você esteja trabalhando num esquema de dados muito grande, você provavelmente não precisará delas.

Ambiente Operacional

Conceito de transação

É um conjunto de procedimentos que é executado num banco de dados, que para o usuário é visto como uma única ação. A integridade de uma transação depende de 4 propriedades, conhecidas como ACID.

- **Atomicidade:** Uma transação não pode ser executada pela metade, isto é, ou se executa ela por inteiro, ou se retorna para o estado anterior a transação, onde nada foi executado.
- **Consistência:** Uma transação deve ser efetuada como um programa que preserva a consistência do BD. Sendo assim, ela é de responsabilidade do programador que codifica a transação. Ela só executa se o estado do Banco de Dados permanecer consistente após seu fim.
- **Isolamento:** Sua necessidade surge em execuções concorrentes, as diversas transações que ocorrem simultaneamente, não podem ser intercaladas de forma a gerar um estado inconsistente.
- **Durabilidade:** Quando ocorre falha no banco de dados, apos a execução com sucesso de uma transação, a durabilidade garante por algum mecanismo a recuperação das informações perdidas.

UM EXEMPLO DE *DEADLOCK*	
Sessão 1	**Sessão 2**
postgres=# BEGIN WORK; BEGIN	
	postgres=# BEGIN WORK; BEGIN
postgres=# UPDATE tbl_usuario SET nome='Mauricio' WHERE usuid=2;	
	postgres=# UPDATE tbl_usuario SET nome='Juliano' WHERE usuid=1;
postgres=# UPDATE tbl_usuario SET nome='Julio' WHERE usuid=1;	
	postgres=# UPDATE tbl_usuario SET nome='Marcio' WHERE usuid=2;

Concorrência

Controle de concorrência é um método usado para garantir que as transações

são executadas de uma forma segura e segue as regras ACID. Os SGBD têm de ser capazes de assegurar que nenhuma ação de transações submetidas (*committed transactions*) serão perdidas ao desfazer transações abortadas (*rollback*). Uma transação é uma unidade que preserva consistência.

Deadlocks acontecem quando duas diferentes aplicações tentam e alteram os mesmos dados na mesma hora. Neste ponto, ambas as sessões são bloqueadas, pois cada uma está aguardando a outra para serem liberadas. Os SGBDS atuais tratam o *deadlock* através da sua detecção, encerrando uma das sessões (com *rollback*), e informando o erro por de uma mensagem.

Outros problemas que podem ocorrer com um banco de dados, devido à concorrência de transações são:

- **Problema da Atualização Perdida:** quando duas transações acessam os mesmos itens em operações intercaladas, e tornam o valor de um dos itens de forma incorreta.
- **Problema da Atualização Temporária:** uma transação atualiza um item, que é acessado por outra transação, mas a atualização no item falha por alguma razão, e volta ao seu valor.
- **Problema do Sumário Incorreto:** ocorre quando uma transação aplica uma função agregada (estatística, como contagem ou média) para um grupo de registros, enquanto outras transações fazem atualizações no grupo.

Existem técnicas para impedir que esses problemas ocorram:

- **Bloqueio em 2 fases:** consiste de aplicar bloqueios em operações de atualização ou consulta, sendo do tipo:
    - *Bloqueio Compartilhado (Shared):* permite que outras transações leiam o item
    - *Bloqueio Exclusivo (Exclusive):* controlado por uma transação única
- **Ordenação por *Timestamp*:** o *timestamp* é um identificador único para a transação, criado pelo SGBD.

Recuperação

É a capacidade de recuperação automática no caso de falhas e de ferramentas de restauração do banco de dados. Aplicativos de importação e exportação de dados (tabelas, outros objetos) devem estar disponíveis.

As falhas são classificadas como:

- **Transação:** erros de lógica, divisão por zero, *overflow*, parâmetros inválidos, etc.
- **Sistema:** erros que afetam o sistema como um todo, como *crash* do sistema, falta de energia, *deadlocks*...
- **Mídia:** normalmente relacionado ao hardware, como falha ao gravar na memória principal, problemas de disco...

Para se recuperar de falhas que afetam as transações, o sistema mantém um **log**, mantido em disco.

Segurança

É a existência de mecanismos de bloqueio a acessos não autorizados. Algumas funcionalidades:

- Controlar o acesso ao banco de dados
- Conceder acesso a objetos específicos no banco de dados.
- Confirmar privilégios concedidos e recebidos com o dicionário de dados
- Criar sinônimos para os objetos de banco de dados

Independência dos Dados

As aplicações não deverão sofrer alteração em função de mudanças físicas no banco de dados. Também deverão ser imunes a reestruturações lógicas do banco.

A habilidade de modificar a definição de um esquema em um nível sem afetar a definição de esquema num nível mais alto é chamada de independência de dados. Existem dois níveis:

- **Independência física de dados:** habilidade de modificar o esquema físico (interno) sem a necessidade de reescrever os programas aplicativos. As modificações no nível físico melhoram o desempenho.

- **Independência lógica de dados:** habilidade de modificar o esquema conceitual sem a necessidade de reescrever os programas aplicativos. As modificações neste nível são necessárias quando a estrutura lógica do banco é alterada.

Integridade

É a capacidade do banco de garantir em qualquer instante a consistência do banco de dados, através de integridade referencial, integridade de domínio e integridade de entidade.

Procedimentos ("Stored Procedures")

Procedimento armazenado ou Stored Procedure é uma coleção de comandos em SQL para gerenciamento de Banco de dados. Encapsula tarefas repetitivas, aceita parâmetros de entrada e retorna um valor de status (para indicar aceitação ou falha na execução). O procedimento armazenado pode reduzir o tráfego na rede, melhorar a performance, criar mecanismos de segurança, etc.

- **Exemplo: (MS-SQL Server):**

```
Create procedure busca
@nomedebusca varchar (50)
as select nome1, nome2
from nome_da_tabela
where nome = @nomedebusca
```

Visões ("views")

Uma visão (*view*) é uma tabela lógica ou virtual composta pelo resultado de um conjunto de *queries* pré-compiladas. Diferente de tabelas normais em um banco de dados relacional, uma *view* não faz parte do esquema físico, mas sim uma tabela dinâmica e virtual, computada ou coletada de dados do banco. Alterar os dados em uma visão altera os dados do banco, embora o SGBD possa ter meios de impedir isso (*read only views*).

Gatilhos ("Triggers")

Gatilho ou trigger é um recurso de programação presente na maioria dos sistema de gerenciamento de banco de dados, utilizado para associar um procedimento armazenado a um evento do banco de dados (leitura, inclusão, exclusão, atualização de registro, por exemplo) de modo que o procedimento armazenado seja executado automaticamente sempre que o evento associado ocorrer.

Índices e otimização de acesso

O Índice é um arquivo auxiliar associado a uma tabela. Sua função é acelerar o tempo de acesso às linhas de uma tabela, criando ponteiros para os dados armazenados em colunas especificas. O banco de dados usa o índice de maneira semelhante ao índice remissivo de um livro, verificando um determinado assunto no índice e depois localizando a sua posição em uma determinada página.

Bancos de Dados Distribuídos

Os Bancos de Dados Distribuídos surgiram com a fusão das tecnologias de banco de dados e de rede de comunicação de dados. Um BDD pode ser entendido como uma coleção de múltiplos bancos de dados, logicamente inter-relacionados, distribuídos por uma rede de computadores. Um Sistema Gerenciador de Banco de Dados Distribuído é um sistema de software que gerencia um Banco de Dados Distribuídos enquanto torna a distribuição transparente para o usuário.

Características

Um sistema de banco de dados distribuídos consiste em uma coleção de nós, cada qual podendo participar na execução de transações que fazem acesso a dados em um ou diversos nós. A diferença principal entre sistemas de bancos de dados centralizados e distribuídos é que no primeiro os dados estão localizados em um único local, enquanto no último os dados residem em diversos locais, e há a necessidade de comunicação entre eles.

Uma vantagem é que se um nó falhar em um sistema distribuído, os nós remanescentes podem ser capazes de continuar operando. Em particular, se itens de dados são duplicados em diversos nós, uma transação que necessita de um item de dado particular pode achá-lo em diversos nós. Assim, a falha de um nó não implica necessariamente no desligamento do sistema.

A principal desvantagem do sistema de banco de dados distribuídos é a complexidade adicional requerida para assegurar a própria coordenação entre os nós. Uma vez que os nós que formam o sistema distribuído operam em paralelo, é mais difícil assegurar que os algoritmos estão corretos, e há um grande potencial para defeitos. Pode existir também sobrecarga de processamento, já que há troca de mensagens entre os nós.

- Transparência na distribuição, replicação e fragmentação
- Melhora na confiabilidade, disponibilidade e desempenho
- Fácil de expandir (extensibilidade)
- Complexidade no controle

Os BDDs podem implementar operações adicionais que se aproveitam da distribuição dos nós para melhorar algum aspecto do banco. Uma dessas operações é a **Semi-junção**, que visa reduzir o número de tuplas em uma relação antes de transferi-la para outro site. Outra questão no processamento de consultas é onde executá-las: no cliente ou no servidor:

- **Query Shipping:** transporte da consulta do cliente para o servidor, para ser executada. Vantajosa quando o servidor possui grande capacidade de armazenamento.
- **Data Shipping:** transporte da consulta do servidor para o cliente, para ser executada pelo cliente. É vantajosa quando há disponibilidade de processamento nos clientes, porém há um alto custo de comunicação na rede (sem *cache*).
- **Hybrid Shipping:** é uma combinação das anteriores, com o objetivo de reduzir ao máximo o custo da execução.

Fragmentação e Replicação

"Qual site deve ser usado para armazenar quais partes do Banco de Dados?", supondo que não haja replicação.

- **Fragmentação Horizontal:** bancos armazenam um subconjunto de tuplas (ex: onde DNO < 5)
- **Fragmentação Vertical:** bancos armazenam um subconjunto de colunas
- **Fragmentação Mista:** ou híbrida, combinação das anteriores

A replicação de dados entre os bancos de dados distribuídos implica na sua redundância entre os sites, sendo útil na melhoria da disponibilidade dos dados, pois se um site ficar fora, o dado pode ser buscado em outro. Isso é ainda mais forte na arquitetura completamente replicada, que possui alta disponibilidade e capacidade de recuperação. O desempenho das consultas aumenta, já que não é necessário navegar pela rede para buscar um dado replicado. Mas o desempenho em atualizações diminui.

Banco de Dados de Objetos

Os Banco de Dados de Objetos (BDOs) são projetados de modo que possam ser diretamente integrados com os softwares que estão sendo desenvolvidos, e que utilizam linguagens de programação orientadas a objetos.

Um objeto possui dois componentes: o seu **estado** (valor) e o seu **comportamento** (operações). Ao converter um modelo relacional para um outro orientado a objetos, alguns itens podem ser mapeados diretamente, como os atributos para variáveis de instância, e os relacionamentos entre objetos são representados por referências inversas (OIDs, integridade referencial).

Um BDOO fornece uma identidade única (OID) para cada objeto armazenado.

O estado de um objeto pode ser formado por outros objetos, pelo uso de alguns **construtores de tipo**. Alguns deles são:

- **Atômico:** números inteiros, números reais, cadeia de caracteres, booleanos.
- **Tupla (*tuple*):** é um tipo estruturado, agregando informações afins. Corresponde ao *struct* do C++.

- **Conjunto (*set*):** nesse tipo de dado não pode haver dois elementos com o mesmo valor (na *bag* isso é possível).
- **List, Bag e Array:** são do tipo "coleção". O tipo *List* e *Array* são ordenados, enquanto o *Bag* é desordenado.

Outros conceitos que devem ser levados em conta em BDOs são:

- Encapsulamento de Operações
- Compatibilidade com linguagens de programação
- Hierarquias de tipo e Herança (aumenta a reusabilidade e diminui a quantidade de código escrito).
- Suporte a objetos complexos
- Polimorfismo e sobrecarga de operador
- Criação de versões (suporte para várias versões do mesmo objeto)
- ODL (Linguagem de Definição de Objetos) e OQL (Linguagem de Consulta de Objetos, similar ao SQL)

SQL (ANSI)

SQL (*Structured Query Language*) é uma linguagem de pesquisa declarativa de banco de dados, utilizada para manipulação de dados e criação ou manutenção de bancos relacionais. SQL foi criada originalmente pela IBM, e foi adotada como um padrão pela ANSI em 1986 e pela ISO no ano seguinte.

A maioria dos bancos de dados relacionais utilizam o SQL como linguagem padrão. Apesar disso, o SQL não pode ser considerável portável entre os vários bancos de dados, pois muitos bancos de dados implementam extensões do SQL (como o PL/SQL da Oracle e o Transact-SQL da Microsoft), e muitos também omitem ou modificam algumas implementações de certos comandos ou tipos de dados.

A linguagem SQL é baseada na Álgebra Relacional, com algumas modificações e extensões.

Principais instruções de manipulação de dados

SELECT é o comumente mais usado do DML, comanda e permite ao usuário especificar uma *query* como uma descrição do resultado desejado. A questão não especifica como os resultados deveriam ser localizados.

INSERT é usada para adicionar uma linha (formalmente uma tupla) a uma tabela existente.

UPDATE para mudar os valores de dados em uma tupla de tabela existente.

DELETE permite remover tuplas existentes de uma tabela.

BEGIN WORK (ou START TRANSACTION, dependendo do dialeto SQL) pode ser usado para marcar o começo de uma transação de banco de dados que pode ser completada ou não.

COMMIT envia todos os dados das mudanças permanentemente.

ROLLBACK descarta as mudanças nos dados existentes desde o último COMMIT ou ROLLBACK.

COMMIT e ROLLBACK interagem com áreas de controle como transação e locação. Ambos terminam qualquer transação aberta e liberam qualquer cadeado ligado a dados. Na ausência de um BEGIN WORK ou uma declaração semelhante, a semântica de SQL é dependente da implementação.

Uso do Join

Uma junção (*join*) é uma *query* que combina linhas de várias tabelas ou visões. Sempre que são especificadas mais de uma tabela na cláusula FROM, uma junção é feita.

- A **junção natural** é baseada em todas as colunas com o mesmo nome em ambas as tabelas.

```
SELECT location_id, city, department_name
FROM locations NATURAL [INNER] JOIN departments;
```

- A **junção cartesiana** (ou *cross join*) acontece quando todas as combinações linha a linha são geradas.

```
SELECT a, b FROM cont CROSS JOIN dept
```

- As **junções externas** (ou *outer joins*) podem ser à direita, à esquerda, ou completas, e são usadas para retornar linhas baseadas em uma condição, assim como as linhas que não atendem à condição da tabela à direita, ou à esquerda, ou de ambas.

```
-- Mostra todos os países, independente da existência das localida-
des
SELECT country_name, city
FROM  locations NATURAL RIGHT OUTER JOIN countries


SELECT e.emp_id, e.emp_name, d.dept_id, d.dept_name
FROM emp e FULL OUTER JOIN dept d ON e.dept_id = d.dept_id;
```

Subconsultas

Subconsultas (ou *subqueries*) são *queries* dentro de outras *queries*. A *query* mais interna é processada primeiro, e retorna um conjunto de resultados para a *query* externa.

Subconsultas utilizadas na cláusula FROM geram o que é chamado de uma *inline view*.

Subconsultas utilizadas na cláusula WHERE são chamadas de subconsultas aninhadas.

Exemplos:

```
SELECT last_name, first_name, salary FROM emp
WHERE  salary = (SELECT MAX(salary) FROM emp);


SELECT name FROM city
WHERE  (cnt_code, st_code) IN
   (SELECT cnt_code, st_code FROM  state
    WHERE  st_name = 'TEXAS');
```

Elaboração de consultas SQL

Outros operadores utilizados em SQL ANSI são os operadores de conjuntos:

- **UNION:** retorna todas as linhas únicas selecionadas por qualquer uma das *queries*.
- **UNION ALL:** retorna todas as linhas (incluindo duplicatas) retornadas por qualquer uma das *queries*.
- **INTERSECT:** retorna linhas selecionadas de ambas as *queries* (linhas em comum).
- **MINUS:** retorna linhas únicas selecionadas da primeira *query*, que não estejam presente na segunda.

A operação de divisão é uma das consultas mais difíceis utilizando o SQL.

-- S é a tabela (A B E) e R é a tabela (A B).

-- Divisão de S por R

SELECT S.E FROM S WHERE NOT EXISTS

(SELECT * FROM R WHERE NOT EXISTS

(SELECT * FROM S WHERE R.A = S.A AND R.B = S.B

Soluções de Suporte à Decisão

Apoio à Decisões é a entrega de informações a usuários finais que precisam de dados para tomar as suas decisões de negócio. Para fornecer esta funcionalidade, os dados de múltiplos sistemas transacionais são integrados em um único sistema, e analisados na forma de uma informação de apoio à decisões. Este sistema é chamado SAD (Sistema de Apoio à Decisões).

Para garantir um suporte eficiente às organizações, o SAD foi desenvolvido em diferentes fases. Originalmente, companhias criaram os relatórios de gerenciamento, que eram entregues em papel. Porém, ficou óbvio que este método de gerenciamento de informações para usuários era ineficiente. Eles eram difíceis de manter, e possuem uma acessibilidade limitada, além de ser suscetível a erros humanos.

Para sobrepor esta limitação, relatórios passaram a ser gerados em um ambiente de linguagem de terceira geração (3GL), onde o programador precisaria coletar e entender os dados desejados pelo usuário, e então definir o formato apropriado (como cabeçalhos, rodapés e quebras de controle). Após projetar o relatório, o programador precisava codificá-lo e testá-lo. A chance de erros humanos diminuiu, porém o processo todo consumia bastante tempo, além de não suportar relatórios customizáveis pelos requerimentos de vários usuários (como o departamento de *marketing*, vendas, etc). O código para obter dados de vários sistemas era muito complexo.

Para superar os problemas dos ambientes 3GL, algumas organizações criaram sistemas de informações executivas (EISs), que foram projetados para fornecer um alto gerenciamento, com suporte automatizado à decisões. Estes sistemas foram úteis para autoridades na empresa, como executivos seniores. Os empregados de nível mais baixo, como o pessoal de produção, não tinha acesso à esses sistemas. Além disso, o custo, complexidade e inflexibilidade destes sistemas reduziu a sua eficiência.

Com o advento da tecnologia cliente / servidor (1980), novas ferramentas foram desenvolvidas para permitir aos clientes criarem seus próprios relatórios, embora ainda não fossem amigáveis. O usuário precisaria conhecer as estruturas de dados armazenadas nos bancos, às vezes gerando *queries* de baixa performance (com vários *joins*), pois esses bancos não foram projetados visando o suporte à decisões.

Desta forma foram criados bancos de dados especializados, projetados para a análise das informações. Esses bancos foram conhecidos como *Data Warehouse*, que utilizavam processamento on-line analítico (OLAP) para produzir os resultados da análise dos dados.

Os DAS atuais são projetados para ajudar uma organização responder seis tipos de perguntas para suportar suas decisões: Quem?, O Quê?, Quando?, Onde?, Por quê? e Como? Alguns exemplos: "Quem é o cliente?", "O quê eles compram da nossa empresa?", "Por quê eles compram de nossos concorrentes?". Responder a essas perguntas ajuda os gerentes a entender as necessidades de seus clientes e da sua própria organização.

Conceitos de Data Warehouse e Aplicações

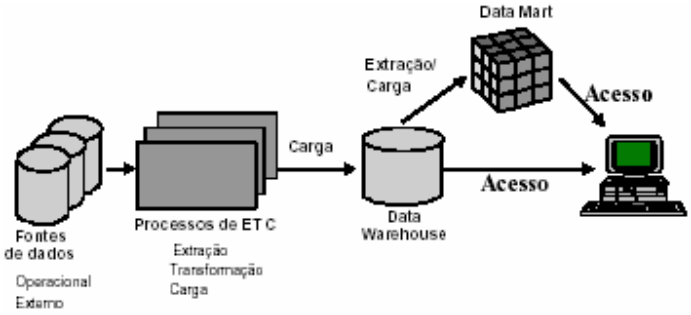
O termo *Data Warehouse* (DW) é utilizado para nomear um "banco de dados histórico, separado do ambiente de produção da empresa e projetado para apoio à decisão". Um DW é uma coleção de dados integrados, orientados por assunto, não-voláteis e variáveis em relação ao tempo, de apoio às tomadas de decisão gerenciais.

Nos últimos dez anos, alguns projetos de DW fracassaram, e o que observamos hoje é a presença de uma manobra mercadológica tentando vender o mesmo produto com nome diferente. Um DW deve:

- Tornar as informações de uma organização acessíveis: o conteúdo de um DW deve ser compreensível e navegável e o acesso aos dados deve ser feito com bom desempenho.
- Tornar as informações de uma organização consistentes: as informações oriundas de diferentes áreas da organização devem ter garantidas a integridade semântica, ou seja, combinadas sem problemas de nomes iguais para coisas diferentes, ou nomes diferentes para a mesma coisa.
- Ser uma fonte de informações flexível e adaptável: um DW deve suportar contínuas modificações estruturais para inserção de novos dados sem comprometer a estrutura já existente.
- Ser o alicerce para a tomada de decisão: o DW deve conter os dados certos na forma certa para suportar a tomada de decisão. Só há uma verdadeira saída de um DW: as decisões que forem tomadas após o DW apresentar as evidências. O DW antes de tudo é um SAD (Sistema de Apoio à Decisões).

Um termo relacionado é o BI, abreviação para *Business Intelligence*, ou Inteligência Aplicada aos Negócios. Podemos dizer que BI é um "conjunto de tecnologias que permitem a análise do desempenho de um negócio".

Ferramentas de BI sempre existiram e fizeram parte de um projeto de DW. Veremos que a maioria das soluções de BI apresentadas precisam de um banco de dados histórico e separado do ambiente de produção (DWs setoriais ou *Data Marts*). Para analisar o desempenho do negócio com eficiência, deve-se usar um banco de dados histórico separado das bases operacionais. A maioria dos produtos existentes no mercado sempre montam um "DW" para esse tipo de análise.



Para implantar ferramentas de BI com bom desempenho, como um pacote CRM (*Customer Relationship Management*), por exemplo, é necessária a implementação de um DW, que é a base para aplicações de BI.

A arquitetura de um ambiente de data warehouse abrange estruturas de armazenamento, mecanismos de integração, comunicação, processamento e apresentação da informação para o usuário final. É composta de:

- **Arquitetura de Dados:** descreve o conteúdo do data warehouse – os dados que são importantes para o negócio, as estruturas de armazenamento que compõem o ambiente do DW e as fontes que as alimentam, os modelos de dados lógicos e físicos, agregações e hierarquias, tudo baseado nos requisitos levantados. Define a granularidade dos dados, o volume e a distribuição dos dados no ambiente.
- **Arquitetura Técnica:** abrange os processos e as ferramentas que atuam sobre os dados. É ela que se preocupa com a forma com a qual os dados serão extraídos da fonte, como serão tratados de modo a atender os requisitos do negócio, e como fazer para que eles se tornem acessíveis para o usuário. É responsável pelo gerenciamento das atividades que criam e mantêm as informações do DW.
- **Administração e Gerenciamento:** É responsável por toda a infra-estrutura do ambiente de data warehouse (hardware, rede, etc.) e pelo gerenciamento dos serviços que contribuem para manter o data warehouse atualizado e consistente. Atividades: segurança, monitoramento da performance, dimensionamento do *hardware*, *backup* e recuperação, checagem da qualidade de dados, etc.
- **Metadados:** consistem em informações sobre dados que compõem o data warehouse. Eles são extremamente importantes dentro do ambiente, pois representam uma visão integrada das bases de dados que fazem parte deste ambiente. Eles são utilizados para construir, manter, gerenciar e utilizar o DW.

Estruturas de armazenamento para Data Warehouse

Para habilitar a tomada de decisões orientada a negócios em uma organização, os usuários precisam de acesso facilitado aos dados de negócio relevantes. As características de um *data warehouse* permitem isso.

Um *data warehouse* é definido como uma coleção de dados orientados por assunto, integrados, que variam com o tempo e não voláteis, que suportam as capacidades de tomada de decisão dos usuários do negócio. Em um modelo relacional, um sistema de faturamento possui inclusões, deleções e atualizações de dados diariamente. Já no *Warehouse*, acontecem somente cargas de dados e consultas, ou seja, falando tecnicamente, há somente SELECTs e INSERTs, e não há UPDATEs.

- **Orientado a assuntos:** implica que os dados relevantes dos maiores assuntos de uma organização, como clientes, produtos e contas, são recolhidos juntos, em único lugar. Por exemplo, em um sistema bancário OLTP, aplicações independentes são mantidas para as diferentes transações do cliente (depósitos, empréstimos, etc). Para criar um relatório financeiro para um cliente, precisaríamos acessar cada sistema OLTP separadamente. Porém, um *data warehouse* pode coletar os dados independentes para fornecer esta informação para o cliente. Como o DW é organizado por temas (assuntos), os dados são menos agrupáveis por *joins*.
- **Variável com o tempo:** a dimensão do tempo é um tema comum onde os dados do DW são construídos. Tempo é um dos critérios primários para filtragem de várias atividades. Por exemplo, um analista pode gerar uma *query*, em uma dada semana, um dado mês ou ano. Podem ser comparadas as vendas dos primeiros meses com anos anteriores. Pode responder a pergunta "Vale à pena lançar essa campanha no inverno?". A causa de uma subida ou declínio nas vendas pode ser analisada, assim como outros dados históricos.
- **Integrado:** os dados são carregados de várias fontes, e antes de serem lidos pelo DW, necessitam ser convertidos em um formato que seja preciso, significativo e consistente. Exemplo: três sistemas (vendas, catálogo e vendas especiais), cada sistema identifica um produto de forma diferente (letras, números). Para que a informação sobre os produtos seja carregada pelos três sistemas de forma significativa, eles devem ser convertidos para um código em comum.

- **Não-Volátil:** implica que os dados não são alterados após o seu carregamento pelo DW. Porém, esses dados devem ser precisos e atualizados, pois são o alvo de *queries* para a busca de informações. Considere o exemplo de uma cadeia de lojas alimentícias. A movimentação de um produto entre as lojas é monitorado e atualizado por um sistema OLTP. O status desta movimentação muda em cada estágio. O status do produto é volátil até ele ser encaminhado para o depósito de uma loja, quando o seu status se torna completo, e pode ser armazenado no DW.

Data Marts

Acessar dados do DW pode consumir bastante tempo, por causa do grande número de usuários e volume de dados. Para lidar com esse problema, utilizam-se *Data Marts*.

Um *data mart* é uma implementação de uma estratégia de particionamento de dados, que permite os usuários acessar dados por departamento. Podem existir DW que possuem informações de todos os departamentos de uma organização, mas também um *data mart* que contém dados apenas do dept. financeiro. Os *data marts* podem ser o ponto de início de um DW.

Se um *data mart* é referenciado por um *data warehouse*, então ele é chamado de dependente. Caso seja carregado diretamente de sistemas OLTP, então é chamado de independente.

Com o aparecimento de data mart ou warehouse departamental, a abordagem descentralizada passou a ser uma das opções de arquitetura data warehouse.Os data marts podem surgir de duas maneiras. A primeira é top-down e a outra é a botton-up.

- **Top-down:** é quando a empresa cria um DW e depois parte para a segmentação, ou seja, divide o DW em áreas menores gerando assim pequenos bancos orientados por assuntos departamentalizados.
- **Botton-up:** é quando a situação é inversa. A empresa, por desconhecer a tecnologia, prefere primeiro criar um banco de dados para somente uma área. Com isso os custos são bem inferiores de um projeto de DW completo. A partir da visualização dos primeiros resultados parte para outra área e assim sucessivamente até resultar num DW.

Implementar um *data mart* traz alguns benefícios, como a localização (geográfica), fazendo que seja específico para uma determinada região, além de ser menor e mais fácil de gerenciar que o DW organizacional. Outro benefício de implementar um *data mart* é que ele reduz a demanda de dados no DW.

A tecnologia usada tanto no DW como no Data Mart é a mesma, sendo a principal diferença é a de que os Data Marts são voltados somente para uma determinada área, já o DW é voltado para os assuntos da empresa toda. Portanto, cabe a cada empresa avaliar a sua demanda e optar pela melhor solução. Como um *data mart* possui uma quantidade significativamente menor de dados que um DW, o número de usuários que o acessam também é menor, reduzindo o tráfico na rede.

Metadados no ambiente de inteligência de negócios.

Metadado é definido como um dado sobre um dado. Ele é importante para todos os aspectos do desenvolvimento, operação e uso de um *data warehouse.* Um metadado de um DW contém:

- Regras de transformação e extração de dados antes de serem carregados para o DW.
- Descrição detalhada da estrutura, conteúdo, chaves primárias e índices dos dados do DW.
- Informação sobre a localização e significado do dado. Ajuda usuários a navegar pelos dados do DW.
- Informação utilizada para mapear dados entre os sistemas de origem e o *data warehouse*.
- Um guia para os algoritmos utilizados ao sumarizar os dados.

Os metadados devem estar sempre atualizados, para garantir que o DW funcione com eficiência.

Ferramentas de front-end: principais recursos e aplicações

É o conjunto de produtos para acesso e análise de dados do DW. Permitem ao usuário navegar entre os dados, resumi-los, compará-los, etc. São ferramentas projetadas para trabalhar de acordo com o modelo de dados implementado. Portanto, a escolha do modelo de dados do DW é fator crítico para o *front-end*.

São três as principais ferramentas de desenvolvimento de DW disponíveis no mercado: Data Mart Suite (Oracle), Impromptu e PowerPlay (Cognos), e SQL Server 7.0 (Microsoft)

Processo de construção de um Data Warehouse

Exemplo de Caso: Cinemas. Acompanhar a evolução do público e valor arrecadado em nível de região do país, estado e cidade, classificados por gênero de filme e sala de cinema. Avaliar a evolução de filmes por ator participante, diretor, etc.

Necessidades Executivas: Quais diretores atraem maior público e em que gênero está esse público? Quais períodos do ano possuem maior público por gênero, ator/diretor, e geograficamente?

Como começar?: Analisar o modelo atual do sistema transacional e, a partir deste, construir o esquema estrela. Modelar o Data Mart a partir das informações colhidas de acordo com suas necessidades.

Construindo o Esquema Estrela: Para construir um modelo dimensional, inicialmente, é necessária total independência do modelo transacional. Analisar cada solicitação cuidadosamente para identificar fatos.

Identificando o Fato: O que nos dá idéia de ação nas solicitações? O que é medido? Quais indicadores de negócios? Resposta: A evolução de público e valor arrecadado. Logo, analisando as solicitações, temos que "Exibição de Filmes" é nosso fato. Pois: Quem tem público para ser acompanhada a evolução? Quem tem valor arrecadado? O fato que tem os indicadores de negócio é exibição de filmes.

Identificando as Dimensões: Onde acontece o fato? Quando acontece o fato? Quem realiza o fato? O que acontece no fato?

Dimensões:

- Onde: está sendo solicitada a evolução de medidas de exibição de filmes por região, estado e cidade.
- **O Quê:** Gênero do Filme é o O QUE do fato. Resumindo: o que deve ser acompanhado. Saber se cresce ou não o público de determinado tipo de filme em determinada região. Exemplo: será que filmes com linguagem escrachada são bem aceitos no Sul do Brasil?
- **Quem:** A Sala de Exibição é o QUEM do nosso problema. Pois é Quem exibe uma sessão de cinema.
- **Quando:** A granularidade desta dimensão está bem disposta nos requisitos: MENSALMENTE. Porém, se temos MÊS, podemos facilmente estruturar nossa Dimensão Tempo como: Ano e Mês. Porém quando analisamos os requisitos com mais detalhes, podemos ver que será necessária uma análise de PERÍODO DE FÉRIAS. Este tipo de análise não pode ser feita com o grão MÊS. Para Contornar este problema modelaremos nossa dimensão TEMPO como: Trimestre, Mês, Data, Ano

Como Tratar Hierarquia no Modelo Dimensional: membros da hierarquia são atributos da mesma dimensão. Criar uma entidade dimensão para cada nível hierárquico (*SnowFlake*).

Definição de Chaves: chaves utilizadas em cada entidade dimensão não devem ser as mesmas do modelo transacional, pois para o Data Mart é importante manter o histórico dos dados. Cada dimensão deve ter uma chave exclusiva nesse ambiente.

Validação do Modelo: para validar este modelo só resta estruturar tabelas com simulação de dados. Por meio de observação poderemos confirmar se é possível obter resultados que satisfaçam as necessidades.

OLAP (On-line Analytical Processing)

A natureza multidimensional dos problemas de negócios resultaram em uma evolução na tecnologia OLAP. Ela permite a análise de dados históricos para a tomada de decisões. São vários os seus benefícios:

- Fornece um meio de acessar, visualizar e analisar dados, com grande flexibilidade.
- Permite lidar com *queries* multidimensionais, que são mais complexas.
- Apresenta dados aos usuários finais de forma natural e intuitiva, navegando pelos modelos de dados de forma efetiva, entendendo a informação. A organização reconhece o valor dos seus dados.
- Acelera a entrega da informação para usuários finais, pois os valores computados são preparados na fase de projeto.
- OLAP garante a segurança dos dados, enquanto vários usuários compartilham dados confidenciais.

As ferramentas OLAP são as aplicações que nossos usuários finais têm acesso para extraírem os dados de suas bases com os quais gera relatórios capazes de responder as suas questões gerenciais. Elas surgiram juntamente com os sistemas de apoio a decisão para fazerem a extração e análise dos dados contidos nos *Data Warehouses* e *Data Marts*. Algumas características:

- **Consultas ad-hoc:** são consultas com acesso casual único e tratamento dos dados segundo parâmetros nunca antes utilizados, geralmente executado de forma iterativa e heurística. O próprio usuário gera consultas de acordo com suas necessidades de cruzar as informações de uma forma não vista e com métodos que o levem aquilo que procura.

- **Slice-and-Dice:** essa característica é de extrema importância. Permite analisar as informações de diferentes prismas limitados somente pela nossa imaginação. Utilizando esta tecnologia conseguimos ver a informação sobre ângulos que anteriormente inexistiam sem a confecção de um DW e a utilização de uma ferramenta OLAP.
- **Drill Down/Up:** consiste em fazer uma exploração em diferentes níveis de detalhe das informações. Com o Drill Down você pode "subir ou descer" dentro do detalhamento do dado, como por exemplo analisar uma informação tanto diariamente quanto anualmente, partindo da mesma base de dados.
- **Geração de Queries:** a geração de queries no OLAP se dá de uma maneira simples, amigável e transparente para o usuário final, o qual precisa ter um conhecimento mínimo de informática para obter as informações que deseja.

Cada uma destas tecnologias e técnicas tem seu lugar no mercado de Suporte à Decisão e apóia diferentes tipos de análises. É importante lembrar que as exigências do usuário devem ditar que tipo de *Data Mart* você está construindo. Como sempre, a tecnologia e técnicas devem estar bem fundamentadas para atenderem da melhor maneira possível essas exigências.

Arquiteturas OLAP

Para suportar os requerimentos analíticos, existem quatro tipos de arquiteturas OLAP que podem ser implementadas, a multidimensional (MOLAP), relacional (ROLAP), *desktop* (DOLAP) ou híbrida (HOLAP).

- **DOLAP:** ferramentas que disparam uma instrução SQL, de um cliente qualquer, para o servidor e recebem o micro-cubo de informações de volta para ser analisado na workstation. O ganho com essa arquitetura é o pouco tráfego que se dá na rede, visto que todo o processamento OLAP acontece na máquina cliente, e a maior agilidade de análise, além do servidor de banco de dados não ficar sobrecarregado, sem incorrer em problemas de escalabilidade. A desvantagem é que o tamanho do micro-cubo não pode ser muito grande, caso contrário a análise passa a ser demorada e/ou a máquina do cliente pode não suportar em função de sua configuração.
- **ROLAP:** possuem uma engenharia de acesso aos dados e análise OLAP com uma arquitetura um pouco diferente. Nesse caso a consulta é enviada ao servidor de banco de dados relacional e processada no mesmo, mantendo o cubo no servidor. O que podemos notar nesse caso é que o processamento OLAP se dará somente no servidor. A principal vantagem dessa arquitetura é que ela permite analisar enormes volumes de dados, em contra partida uma grande quantidade de usuários acessando simultaneamente poderá causar sérios problemas de performance no servidor causando, inclusive, o travamento do mesmo.
- **MOLAP:** processa-se da seguinte forma: com um servidor multidimensional o acesso aos dados ocorre diretamente no banco, ou seja, o usuário trabalha, monta e manipula os dados do cubo diretamente no servidor. Isso traz grandes benefícios aos usuários no que diz respeito à performance, mas tem problemas com escalibilidade além de ter um custo alto para aquisição.
- **HOLAP:** essa nova forma de acessar os dados nada mais é do que uma mistura de tecnologias onde há uma combinação entre ROLAP e MOLAP. A vantagem é que com a mistura de tecnologias pode-se extrair o que há de melhor de cada uma, ou seja, a alta performance do MOLAP com a escalibilidade melhor do ROLAP.

Conceitos de modelagem dimensional

A técnica antiga de se criar bancos de dados simples e compreensível foi aperfeiçoada para o surgimento da modelagem dimensional. A capacidade de poder observar um banco de dados no formato de um cubo, contendo duas, três, ou mais dimensões, permite aos usuários analisar seu negócio através de uma forma concreta e compreensível, pois os conjuntos de dados abstratos se tornam reais.

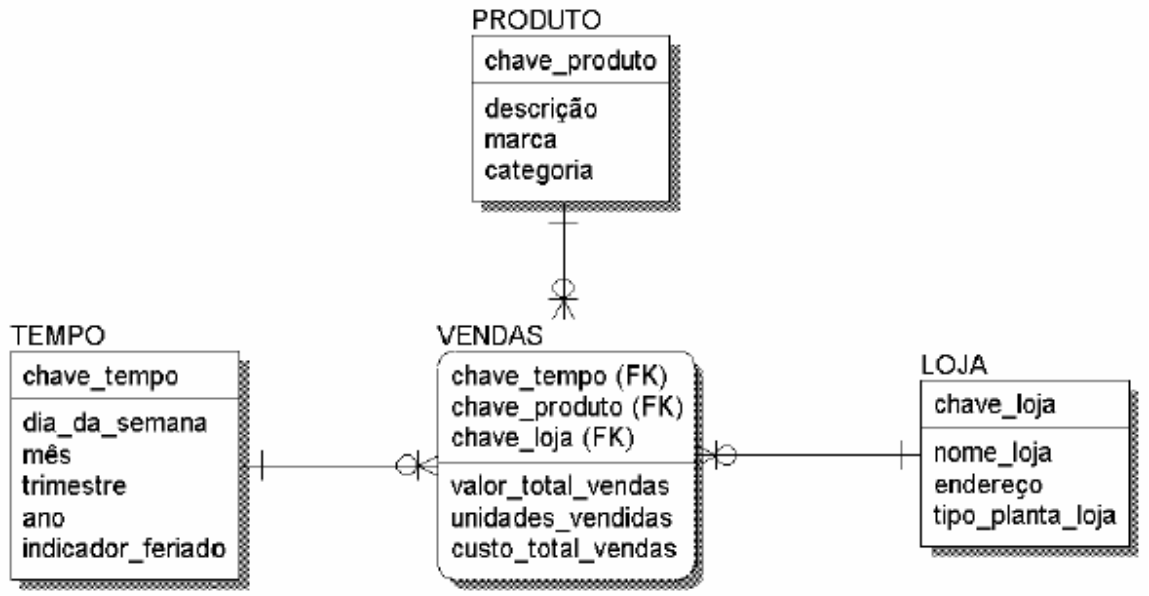
Sobre o cubo (ou hipercubo) gerado, são definidas as seguintes operações:

- **Slice and Dice:** ou fatiamento do cubo, é a restrição das coordenadas nas dimensões de acordo com critérios definido em cima de atributos das dimensões. Por exemplo: visualizar somente os dados dos mercados de pequeno porte.
- **Agregação:** permite reduzir a dimensionalidade de um cubo ou de uma fatia de cubo. Ex: ao visualizar a receita total de cada mercado por mês estamos agregando os dados de receita na dimensão produto, eliminando-a do cubo.
- **Drill-up/Drill-down:** é a navegação entre níveis de agregação, de acordo com hierarquias existentes nas dimensões. Ex: o usuário pode começar visualizando dados totais de receita para os mercados e fazer um drill-up para visualizar os dados totais por bairro, agregando os mercados próximos. A modelagem dimensional proporciona um ganho de tempo na consulta, melhor organização do sistema e a sua utilização se dá de forma intuitiva para o usuário.

Um modelo dimensional é definido em termos de suas **dimensões** e suas medidas (ou **fatos**):

- **Dimensões:** Representam as possíveis formas de visualizar os dados. São as entradas para as consultas (tempo, região, cliente, etc). São os atributos e hierarquias das dimensões que permitem realizar as operações descritas anteriormente, por isso a sua descoberta é um fator crítico para o sucesso do DW.
- **Fatos:** É a tabela central que interliga as dimensões e tem os indicadores de análise ou métricas (quantidade, valores,etc). A tabela fato deverá possuir, no mínimo, as três primeiras dimensões. Os fatos são as medições do negócio, e geralmente são dados numéricos e aditivos, podendo ser agregados por somas, médias, etc. Exemplos de fatos são: valor total da venda, quantidade vendida, etc.

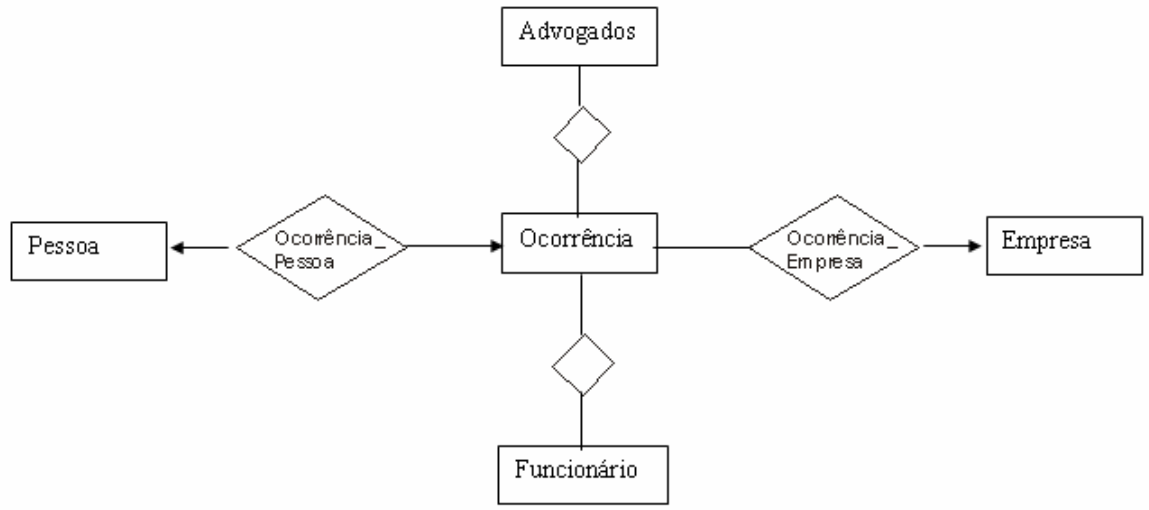
Não existe uma notação gráfica amplamente adotada para modelos dimensionais. Uma das representações lógicas mais comumente utilizada é o esquema estrela (*star schema*), que utiliza a abordagem relacional e os seus componentes, com algumas restrições (entidades, relacionamentos, chaves, cardinalidade, etc:



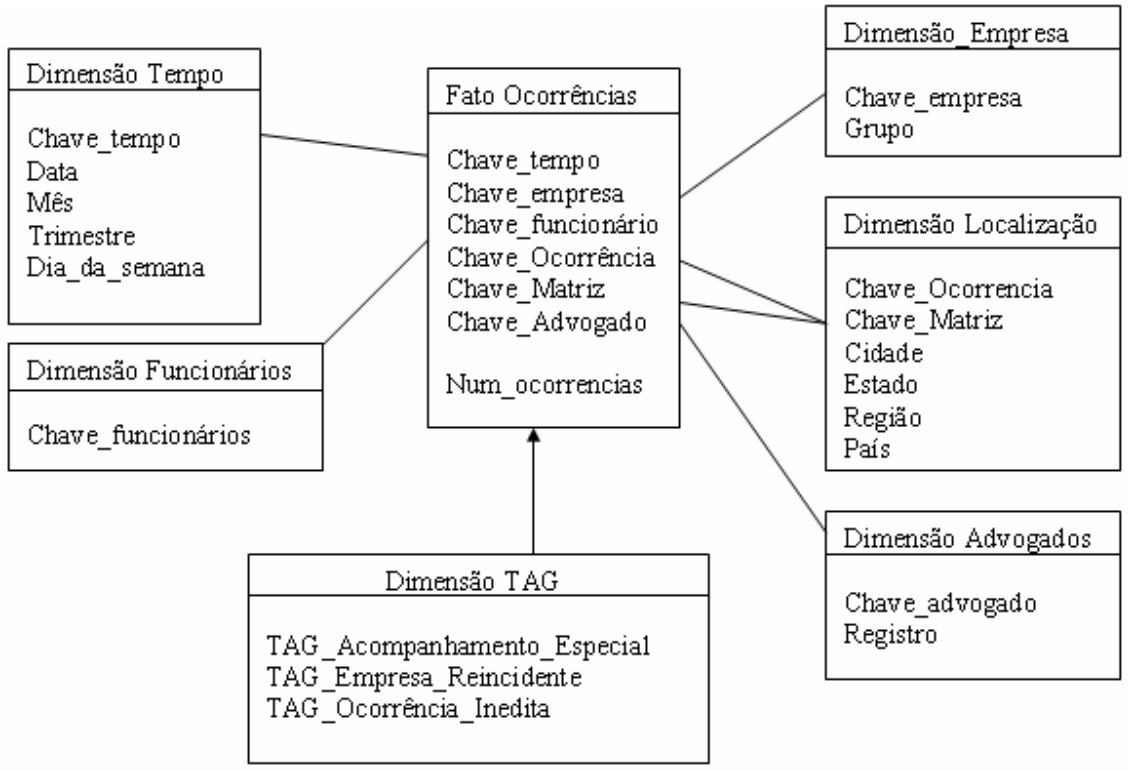
Existe uma variação deste sistema, o floco-de-neve (*snowflake schema*), que faz a normalização das tabelas de dimensão, evitando a redundância. Mas em geral normalizar uma dimensão não traz muito benefício para o modelo, piorando a compreensibilidade e diminuindo a performance, em troca da economia de espaço.

Desenho de modelos dimensionais a partir de modelos transacionais normalizados

Diagrama Entidade Relacionamento referente ao modelo transacional de um estudo de caso de determinado serviço de defesa do consumidor:



Foi identificado o fato ocorrência. O grão do fato será o número de ocorrências registradas de uma empresa, em um dia, acompanhada por um advogado e por um funcionário do serviço de defesa do consumidor:



EIS - Enterprise Information System

Um EIS é geralmente qualquer tipo de sistema de computação de "classe *enterprise*", ou seja, que ofereça alta qualidade de serviço (QoS) e lide com uma grande quantidade de dados – suportando grandes organizações. Tipicamente:

- Operado por Administradores de Sistemas profissionais
- Implantado em servidores dedicados
- Oferece conectividade de rede
- Fornece serviços que suporta as operações da organização

ECM - Enterprise Content Management

ECM é uma tecnologia de software que permite as organizações criarem, capturarem, gerenciarem, armazenarem, publicar, distribuir, buscar e personalizar conteúdo digital, como imagens, textos, relatórios, vídeo, código, entre outros. Os sistemas ECM focam principalmente no ciclo de vida das informações (criação, armazenamento, disseminação, destruição).

Os sistemas ECM podem ser vistos de três formas: como serviços independentes, como um *middleware* integrado (e dessa forma fazendo bastante uso das tecnologias de EAI e SOA) ou como um repositório uniforme para todas as informações.

Os componentes de um sistema ECM são:

- **Captura:** são funcionalidades e componentes para gerar, capturar, preparar e processar informação eletrônica ou analógica. Alguns exemplos são as ferramentas OCR (*Optical Character Recognition*), HCR (*Handprint Character Recognition*), processamento de formulários, índices e classificação automática.
- **Gerenciamento:** lidam com o processamento e uso da informação, envolvendo banco de dados para consultas e administração, e sistemas de autorização de acesso. Ferramentas de controle de versões (*Chekin-Chekout*), busca e navegação, colaboração e *workflow* estão incluídas nessa categoria.

- **Armazenamento:** dividido em duas categorias, *Store* (para armazenamento temporário) e *Preserve* (armazenamento de longo prazo), sendo componentes da infra-estrutura responsáveis por guardar a informação, como banco de dados, repositórios, *data warehouses*, serviços de biblioteca e os próprios dispositivos de armazenamento (disco, etc).
- **Entrega (*Deliver*):** é o componente que apresenta a informação para os demais, contendo funções para a introdução de novas informações no sistema, personalização, publicação e distribuição, envolvendo o formato (XML, PDF, TIF), as tecnologias de segurança, portais para Internet, extranet e intranet, e outras mídias (e-mail, papel, TV, etc).

Data Mining

Knowledge Discovery in Databases (KDD) é o processo de extração de padrões (conhecimentos) embutidos nos dados. Além disso, os padrões extraídos devem ser válidos, novos (previamente desconhecidos), potencialmente úteis e compreensíveis.

Data Mining é um conjunto de técnicas e ferramentas usadas para identificar padrões embutidos em grandes massas de dados.

Data Mining também conhecido como "arqueologia" da informação, é o processo de extrair inteligentemente tendências e informações escondidas em banco de dados ou outros repositórios de informação. Data mining utiliza-se nas áreas de engenharia, ciência, medicina, negócios, educação, etc. É um campo interdisciplinar que liga áreas como sistema de banco de dados, estatística, aprendizado computacional, visualização de dados, recuperação de informação e computação de alto desempenho. Duas técnicas de Data Mining são: Descoberta de Regras de Associação e *Clustering*.

Qualquer sistema de *Data Warehouse* (DW) só funciona e pode ser utilizado plenamente, com boas ferramentas de exploração. Com o surgimento do DW, a tecnologia de Data Mining (mineração de dados) também ganhou a atenção do mercado.

Como o DW, possui bases de dados bem organizadas e consolidadas, as ferramentas de Data Mining ganharam grande importância e utilidade. Essa técnica, orientada a mineração de dados, oferece uma poderosa alternativa para as empresas descobrirem novas oportunidades de negócio e acima de tudo, traçarem novas estratégias para o futuro.

O propósito da análise de dados é descobrir previamente características dos dados, sejam relacionamentos, dependências ou tendências desconhecidas. Tais descobertas tornam-se parte da estrutura informacional em que decisões são formadas. Uma típica ferramenta de análise de dados ajuda os usuários finais na definição do problema, na seleção de dados e a iniciar uma apropriada análise para geração da informação, que ajudará a resolver problemas descobertos por eles. Em outras palavras, o usuário final reage a um estímulo externo, a descoberta do problema por ele mesmo. Se o usuário falhar na detecção do problema, nenhuma ação é tomada.

A premissa do Data Mining é uma argumentação ativa, isto é, em vez do usuário definir o problema, selecionar os dados e as ferramentas para analisar tais dados, as ferramentas do Data Mining pesquisam automaticamente os mesmos a procura de anomalias e possíveis relacionamentos, identificando assim problemas que não tinham sido identificados pelo usuário. Em outras palavras, as ferramentas de Data Mining analisam os dados, descobrem problemas ou oportunidades escondidas nos relacionamentos dos dados, e então diagnosticam o comportamento dos negócios, requerendo a mínima intervenção do usuário, assim ele se dedicará somente a ir em busca do conhecimento e produzir mais vantagens competitivas.

As ferramentas de Data Mining, baseadas em algoritmos que forma a construção de blocos de inteligência artificial, redes neurais, regras de indução, e lógica de predicados, somente facilitam e auxiliam o trabalho dos analistas de negócio das empresas, ajudando as mesmas a conseguirem serem mais competitivas e maximizarem seus lucros.

Análise e Projeto de Sistemas

Análise e Projeto Estruturado de Sistemas

É um assunto abordado por DeMarco, Yourdon e Constantine.

A análise estruturada é a metodologia *front-end* que permite usuários ou analistas de sistemas converter um problema do mundo real em um diagrama pictórico ou outra representação lógica que pode ser utilizada subseqüentemente pelos desenvolvedores e programadores para projetar um sistema de informação.

O projeto estruturado se preocupa com o projeto físico baseado na análise estruturada, ou seja, a transformação das informações lógicas obtidas no passo anterior para um sistema de informação físico.

A análise e projeto estruturado pode ser a metodologia de análise e projeto mais conhecida. Uma de suas características é a análise *top-down* (hierárquica), que tende a gerar sistemas bem organizados. O seu enfoque passo a passo simplifica o gerenciamento do projeto, do risco e de recursos.

Manter a grande quantidade de dados gerados por esta metodologia pode consumir tempo. Além disso, o seu passo a passo é inflexível, e comparado com outras metodologias não é muito amigável.

Ferramentas relevantes incluem: Dicionários de Dados, DFD para a representação das funções do sistema, o modelo E-R para os relacionamentos entre os dados e o DTE para o comportamento dependente do tempo.

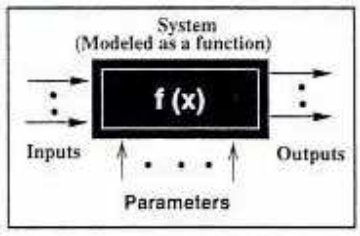
Modelagem de dados

Modelagem de dados é olhar os dados do sistema das três maneiras possíveis (dados, funções e comportamento) em métodos de análise ou projeto estruturados. O diagrama de modelagem de dados mais utilizado é o Diagrama de Entidade Relacionamento (MER). Eles são construídos de forma que as suas estruturas de dados sejam normalizadas de uma vez. O modelo é criado gradualmente: começando com as entidades maiores, adicionando os relacionamentos, depois as chaves e os atributos, para depois adicionar novas entidades.

Em modelagem de dados, os atributos de objetos de dados podem ser usados para nomear uma instância do objeto de dados, descrever a instância ou fazer referência a outra instância de outra tabela.

Modelagem Funcional

Na modelagem funcional, o sistema em consideração é modelado como uma caixa preta, com um conjunto de entradas e saídas. O analista da modelagem está preocupado apenas com o seu funcionamento, e não com os seus detalhes mais internos. A frase chave aqui é "Dada esta entrada, qual é a saída do sistema?"



A principal vantagem desta metodologia é a sua simplicidade e a sua habilidade de combinar modelos menores para obter um modelo de sistemas maiores.

A modelagem funcional não é apropriada quando os objetos são fortemente conectados (de uma maneira não direcional), como redes elétricas analógicas. Redes lógicas (digitais) não são um problema, pois a direcionalidade é especificada no projeto.

A modelagem funcional pode ser classificada em duas maneiras: *function-based approach* e *variable-based approach*.

No modelo baseado em funções existem funções de transferência, que são os blocos de construção do sistema, que fazem a conexão entre cada uma de suas partes, indicando a direção do fluxo de dados por setas. No modelo baseado em variáveis, o foco é dado às variáveis afetadas pelas funções. A modelagem funcional é suportada pelos diagramas de componente e de execução.

O Dicionário de Dados

O dicionário de dados é uma listagem organizada de todos os elementos de dados que são pertinentes ao sistema, com definições precisas e rigorosas, de forma que tanto o usuário como o analista de sistemas tenham uma compreensão comum das entradas, das saídas, dos componentes dos depósitos de dados. [PRESSMAN]

O DFD representa apenas o fluxo de informação do sistema, e o dicionário de dados completa a especificação do sistema. É uma ferramenta importante durante a análise detalhada do DFD, e também na fase de projeto que segue a análise. É uma tarefa tediosa, mas necessária para a precisão e organização sistemática dos termos.

=	é composto de
+	e
( )	opcional (pode estar presente ou ausente)
{ }	iteração
[ ]	seleciona alguma das várias alternativas
**	comentário
@	identificador (campo chave)
|	separa alternativas na construção [ ]

Um dicionário de dados (papel ou automático), deve ser mantido em cada estado da criação do DFD. É importante descrever no dicionário de dados: o significado dos dados, a composição de pacotes de dados, especificação dos valores e unidades relevantes, e o relacionamento entre esses dados.

Para definir um dicionário de dados completo, o significado, a composição e os valores legais de um elemento de dado deve ser descrito por comentários:

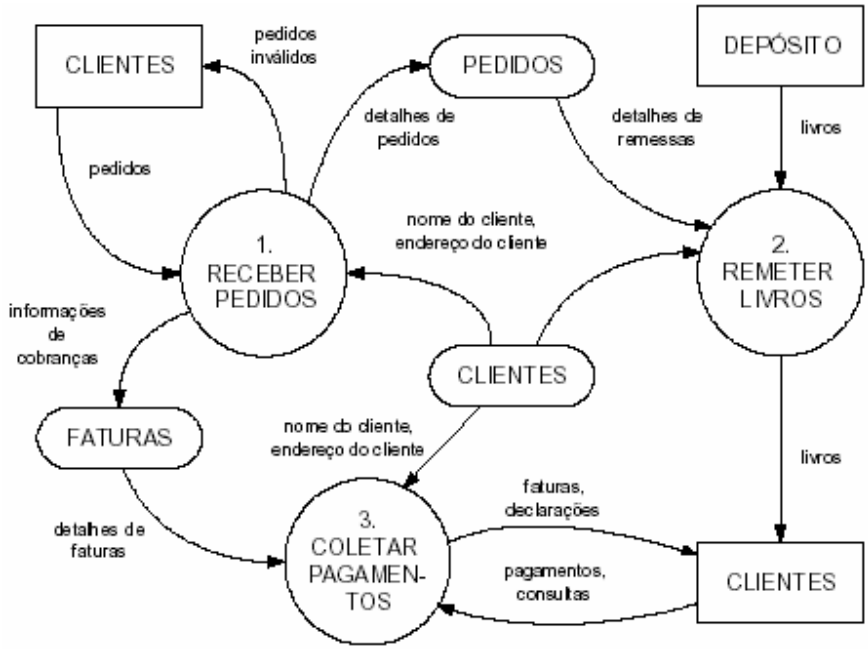
```
Peso_atual = *Peso ao entrar no Hospital*
*unidade: quilos; faixa de valores: 1-400*
```

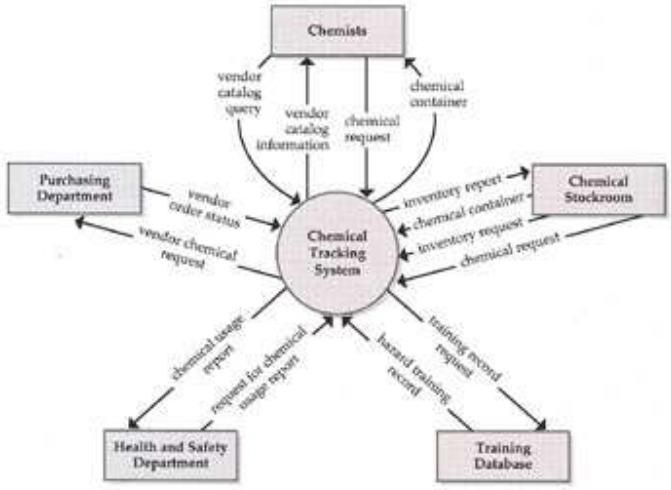
Exemplo de utilização:

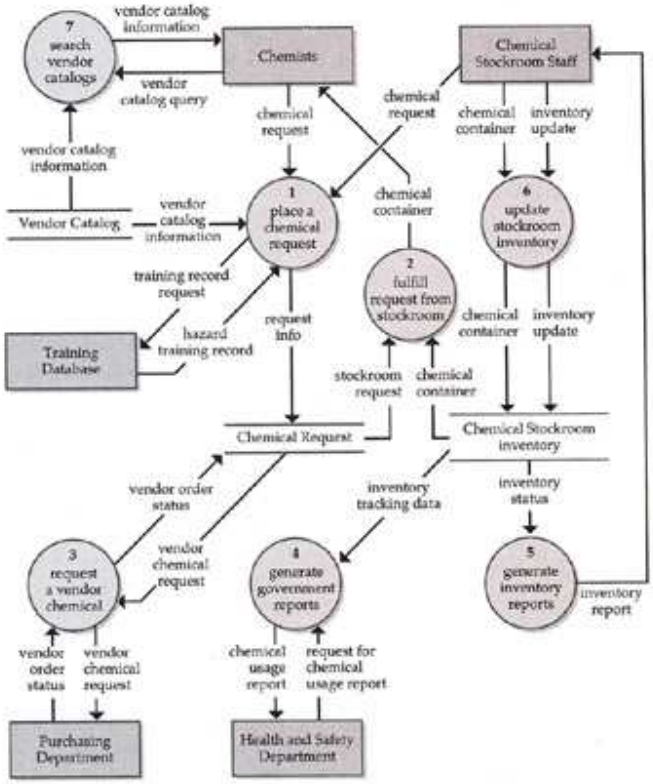
```
name = courtesy-title + first-name + (middle-name) + last-name
courtesy-title = [Mr. | Miss | Mrs. | Ms. | Dr. | Prof. ]
first-name = {legal-character}
middle-name = {legal-character}
last-name = {legal-character}
legal-character = [A-Z|a-z|0-9|'|-| | ]
```

DFD e Modelagem de Fluxo de Dados

Um DFD ou Diagrama de Fluxo de Dados (ou Modelo Funcional ou de Processos) é uma técnica gráfica que mostra o fluxo da informação dos dados de maneira a mostrar as transformações de suas entradas e saídas. O modelo é simples e intuitivo, encapsulando os detalhes e mostrando a estrutura como um todo. Segue um exemplo de DFD (depósitos através de elipses):







É necessário que as entradas e saídas de um fluxo de dados de um DFD multi-nível seja consistente entre todos os níveis. Esta regra só pode ser relaxada quando uma entrada ou saída de um processo for a combinação de dois ou mais fluxos de dados relacionados.

Em resumo, o DFD é utilizado para analisar e projetar as entradas, saídas, processos, armazenamentos e controles de uma informação do sistema. Existem DFDs lógicos (fase de análise) e físicos (fase de projeto), e o último possui de informações da implementação do sistema.

A Figura 1 mostra o **Diagrama de Contexto**, onde todo o sistema é representado por um único círculo (um único Processo). Pode ser utilizado para construir o DFD nível 0 (Figura 2), com mais detalhes e subdivisões. Note a **consistência** de entradas e saídas!

Os fluxos de dados são representados por uma seta (que entra ou sai de um processo).

Os depósitos são representados por duas linhas paralelas ou elipses, e são os dados armazenados (repouso), **internos** ao sistema. Os retângulos são os terminadores, que são as entidades externas com as quais o sistema se comunica, pode ser uma pessoa ou sistema. Ao elaborar um DFD, deve-se evitar processos (círculos) com entradas mas sem saídas, e também dar nomes significativos aos seus componentes. Não ligue setas entre processos!

Outros níveis podem ser criados para especificar mais o sistema (subdividir em detalhes). Lembre de **Numerar** os processos. Também evite criar DFDs muito complexos.

Métodos Funcionais x Métodos OO

Os métodos a serem aplicados, no âmbito das metodologias de desenvolvimento, correspondem a conjuntos de atividades que organizam a execução de determinadas fases do ciclo de vida do sistema.

Métodos Funcionais

Os métodos funcionais baseiam-se fundamentalmente na modelação do fluxo de dados, recomendando, o processo de especificação, que se inicie o desenvolvimento do sistema pela análise dos dados. Esta análise inicia-se com a caracterização do ambiente do sistema, bem como das suas entradas e saídas, o que leva à definição de um diagrama de contexto do sistema.

As atividades internas ao sistema são decompostas por refinamentos sucessivos de DFDs até que cada atividade acaba por ser especificada num formato textual.

Uma vez que os DFDs não se adequam à especificação do comportamento temporal das atividades, podem ser adicionados STDs (Diagramas de Transição de Estados) aos DFDs para possibilitar a especificação dos estados de todas as atividades.

No que diz respeito à concepção, os métodos funcionais suportam a obtenção da arquitetura conceitual do sistema e a implementação é baseada na transformação dos modelos em programas, codificada à luz do paradigma da programação estruturada

A abordagem dos métodos funcionais tem, no entanto, duas grandes desvantagens:

- as decisões mais críticas relacionadas com a estrutura global do sistema têm que ser efetuadas muito cedo, no processo de desenvolvimento, numa altura em que o problema ainda não está bem percebido
- uma vez que a decomposição é efetuada principalmente no domínio funcional, as estruturas de dados mais importantes tendem a ser globais a todo o sistema, o que coloca problemas significativos à fase de manutenção, quando as representações dos dados têm que ser alteradas (devido à novos requisitos)

Métodos OO (orientados por objetos)

Os métodos OO exploram uma abordagem (completamente diferente dos métodos funcionais) inerentemente baseada no encapsulamento da informação e na abstração dos tipos de dados.

A principal característica dos métodos OO é que a estruturação conceitual das representações, podendo ser realizada segundo uma abordagem *bottom-up*, deve modelar a estrutura do mundo real, no qual se baseia o problema em causa.

Cada objeto é uma abstração de uma entidade do mundo real, com um estado interno e com uma interface (a única parte do objeto visível externamente) que recebe os estímulos do exterior e que responde segundo a dinâmica interna do objeto.

Desta forma, uma rede de objetos cooperantes conceitualiza um modelo natural do problema, estando todas as características dos objetos reais encapsuladas nos seus modelos computacionais.

Assim, qualquer alteração no ambiente só tem conseqüências locais no interior de objetos particulares, não existindo a propagação para fora das fronteiras dos objetos em causa. Algumas vantagens são:

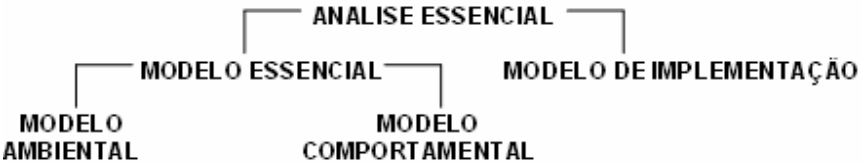
- a fase de concepção permite obter modelos arquiteturais menos dependentes da tecnologia de implementação, sendo esta baseada no paradigma da programação orientada por objetos
- os vários métodos orientados a objetos desenvolvidos por Rumbaugh, Jacobson e Booch geraram uma notação única, a UML, que pode ser utilizada por qualquer método OO.

A orientação a objetos é uma técnica para modelação de sistemas que é relativamente simples de se pensar, pois as pessoas lidam no dia-a-dia com objetos que interagem entre si.

Análise Essencial

A análise essencial começa a especificação de um sistema pela identificação dos eventos que o afetam. Divide-se em:

- **Modelo Essencial:** apresenta o sistema num grau de abstração independente de restrições tecnológicas.
    - *Modelo Ambiental:* define a fronteira entre o sistema e o resto do mundo. É representado pelos diagramas: Declaração de Objetivos, Diagrama de Contexto, Lista de Eventos (evento orientado a fluxo, temporal e temporal relativo) e Dicionário de Dados Preliminar (opcional).
    - *Modelo Comportamental:* define o comportamento das partes internas do sistema necessário para interagir com o ambiente. É representado pelos diagramas: Diagrama de Fluxo de Dados, Mini-Especificação, Diagrama de Transição de Estado e Diagrama Entidade Relacionamento.
- **Modelo de Implementação:** apresenta o sistema num grau de abstração completamente dependente de restrições tecnológicas. É derivado do modelo essencial. Diz respeito a implementação do sistema. Envolve a construção do Modelo Lógico de Dados, as características de processamento das funções / processos e a interface homem-máquina.



A análise essencial também utiliza os métodos **Modelagem de Dados** e **Modelagem Funcional**.

As atividades essenciais são todas as tarefas que o sistema teria que executar se fosse implementado com tecnologia perfeita:

- **Atividades fundamentais:** executam tarefas que são parte dos objetivos do sistema.
- **Atividades custodiais:** mantém a memória essencial.
- **Atividades Essenciais compostas:** ambas.

Análise Essencial x Análise Estruturada

A Análise Essencial começa pelo modelo essencial, o que equivale, na Análise Estruturada, começar diretamente pelo modelo lógico proposto. A Análise Estruturada aborda duas perspectivas do sistema - função e dados - ao passo que a Análise Essencial aborda três perspectivas - **função**, **dados** e **controle**.

Além disso, na Análise Estruturada o particionamento é feito através da abordagem *top-down*, enquanto na Análise Essencial, o particionamento é por eventos.

Análise Estruturada: modelo físico atual => modelo lógico atual => modelo lógico proposto => modelo físico proposto

Análise Essencial: modelo essencial => modelo de implementação

Análise e projeto orientado a objetos

A identificação de objetos inicia-se ao examinar a declaração do problema ou ao executar uma "análise gramatical" da narrativa de processamento do sistema a ser construído. Os objetos podem ser: **Entidades Externas** (outros sistemas, dispositivos, pessoas), **Coisas** (relatórios, displays, cartas, cartazes), **Ocorrências** ou **eventos** (uma transferência de propriedade ou a conclusão de uma série de movimentos de robô), **Papéis** (gerente, engenheiro, vendedor), **Unidades Organizacionais** (divisão, grupo, equipe), **Lugares** (piso de fábrica ou área de descarga) ou **Estruturas** (sensores, veículos de quatro rodas ou computadores).

Critérios de seleção que se usam ao examinar cada objeto em potencial para inclusão no modelo de análise:

- **Informação Repetida:** o objeto em potencial será útil durante a análise somente se a informação sobre ele precisar ser lembrada de forma que o sistema possa funcionar.
- **Serviços Necessários:** o objeto em potencial deve ter um conjunto de operações identificáveis que podem mudar o valor de seus atributos de alguma forma.
- **Múltiplos Atributos:** um objeto com um único atributo pode, de fato, ser útil durante a fase de projeto, mas provavelmente ele será mais bem representado como um atributo de um outro objeto.
- **Atributos Comuns:** um conjunto de atributos pode ser definido para o objeto em potencial, e esses atributos aplicam-se a todas as ocorrências do objeto.
- **Operações Comuns:** um conjunto de operações pode ser definido para o objeto em potencial, e essas operações aplicam-se a todas as ocorrências do objeto.
- **Requisitos Essenciais:** entidades externas que aparecem no espaço problema e produzem ou consomem informações que são essenciais à operação de qualquer solução para o sistema quase sempre serão definidas como objetos no modelo de requisitos.

Modelagem Dinâmica

A modelagem dinâmica do sistema se resume às associações de comunicação (dinâmicas) incluídas no modelo de objetos. Em OMT, a modelagem das classes isoladamente é composta por um Diagrama de Transição de Estados para a classe, e um fluxograma para cada método. Também utiliza os Diagramas de Eventos, como ferramenta de identificação do comportamento dinâmico, e o Diagrama de Fluxo de Eventos. O DFD também descreve o comportamento do sistema.

A modelagem dinâmica se refere a dois aspectos: descrição comportamental, que se preocupa com o fluxo de controle do sistema (seqüências de procedimentos), e descrição funcional, que se preocupa em descrever os fluxos de dados e suas transformações. Estão baseados nos conceitos de evento e estado. Em Análise Orientada a Objetos, eventos representam estímulos externos, e os estados representam os valores dos objetos.

Engloba os sistemas que reproduzem o comportamento de elementos do mundo real, como simuladores e jogos, sistemas de tempo real (controle de processos, comunicações).

UML

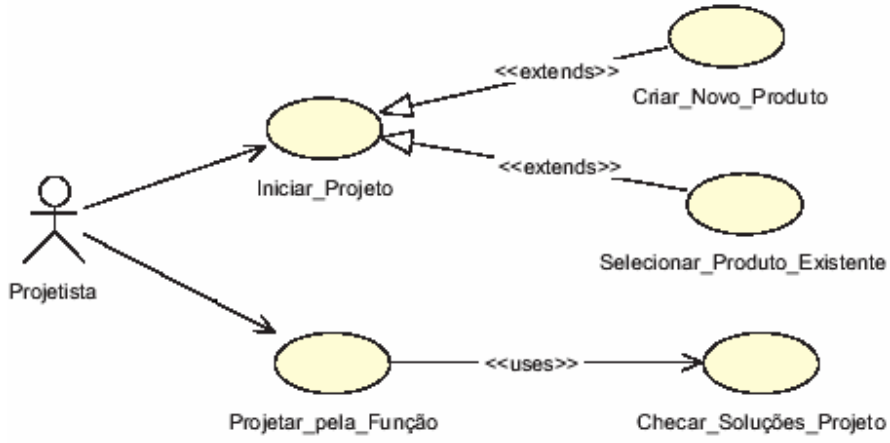
UML significa Linguagem de Modelagem Unificada (*Unified Modeling Language*), e é um sistema de notação orientada a objetos criada por Grady Booch, James Rumbaugh e Ivar Jacobson, em conjunto com a Rational Software Corporation, através da fusão de diversas tecnologias de modelagens de sistemas.

Atualmente a UML é aceita pela OMG (Object Management Group) como o padrão para modelagem de programas orientados a objetos, e sua especificação está disponível no *site* da organização (versão UML 1.5).

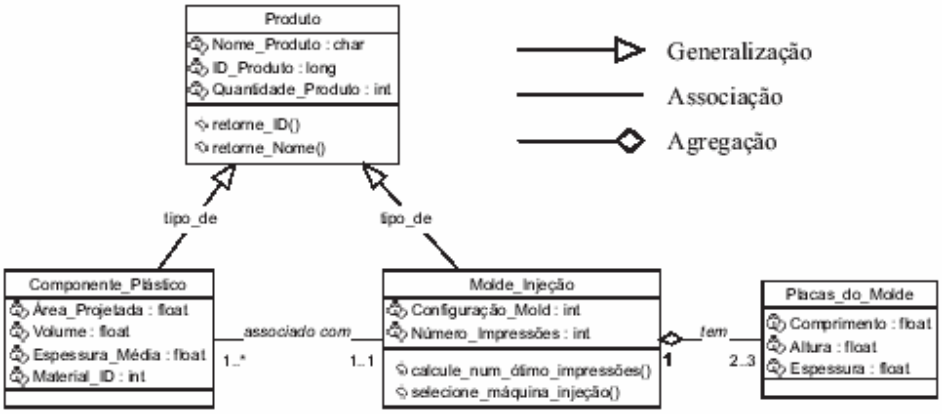
Diagrama de casos de uso

Diagramas de caso de uso são utilizados para modelar a funcionalidade do sistema durante a sua operação, através de atores e casos de uso. É importante lembrar que este diagrama não é adequado para representar o sistema de forma detalhada, pois não pode descrever os seus mecanismos internos, sendo mais utilizados para facilitar a interação entre os usuários do sistema e os projetistas.

- **Atores (actors):** são representações de qualquer entidade que interage com o sistema. Não precisa ser necessariamente uma pessoa, podendo ser também uma máquina, um outro sistema, etc. Atores **não** fazem parte do sistema, apenas representam os papéis que o usuário desempenha. Este papel pode ser ativo ou passivo (ator envia ou recebe informação do sistema).
- **Casos de uso (use cases):** são uma seqüência de ações ou eventos que influenciam o sistema. Cada caso de uso descreve uma característica, e o diagrama com todos os casos de uso deve representar todas as situações possíveis de utilização do sistema. Normalmente os diagramas são acompanhados por uma descrição de cada caso de uso – narrativas de texto do caso de uso – em que devem ser ressaltados vários atributos (como nome, atores relacionados, objetivo, fluxo de eventos, pré e pós condições, etc).



O diagrama de classes representa a estrutura interna (atributos e métodos) e as relações entre um conjunto de classes. Tais relações definirão principalmente a forma em que os objetos serão implementados. Por exemplo, Molde_Injeção e Componente_Plástico são tipos de Produto e herdam os atributos e métodos de tal classe.



Existem três perspectivas quando projeta-se diagramas de classes:

- **Conceitual:** A perspectiva conceitual, projeta um diagrama que representa os conceitos no domínio que está sendo estudado. Estes conceitos serão naturalmente relacionados às classes que vão executá-los, mas freqüentemente não existe um mapeamento direto. Um modelo conceitual deve ser projetado com pouca ou nenhuma preocupação com o software que poderá implementá-lo, portanto ele pode ser considerado independente de linguagem. (Cook e Daniels chamam isso de perspectiva essencial).
- **Especificação:** examinamos o software, estamos analisando as suas interfaces, não a sua implementação. O desenvolvimento orientado a objeto põe muita ênfase na diferença entre interface e implementação, mas isto é freqüentemente negligenciado na prática porque a noção de classe em uma linguagem OO combina interface com implementação, o que é um erro, pois a chave para uma programação OO eficaz é programar para uma interface de classe em vez de fazê-lo para sua implementação. Ouve-se com freqüência a palavra “tipo” para falar sobre uma interface de uma classe; um tipo pode ter muitas classes que o implementam e uma classe pode implementar muitos tipos.
- **Implementação:** Nesta visão, realmente, temos classes e estamos pondo implementação às claras. Esta é a perspectiva usada com mais freqüência, mas, de várias formas, a perspectiva de especificação é a melhor para ser utilizada.

Como a modelagem de classes é utilizada tanto na análise quanto no projeto das aplicações orientadas a objeto, o processo de modelagem compreende passos que envolvem características tanto de análise quanto de projeto. Na maioria das vezes, os passos que envolvem “identificar” estão relacionados com a análise, e aqueles que requerem a ação “definir” dizem respeito ao projeto. Os passos a serem observados durante a modelagem de classes são os seguintes.

1. Identificar as classes;
2. Identificar os atributos das classes (sem levar em consideração suas visibilidades ou tipos);
3. Analisar os atributos das classes, identificando que alguns deles são na realidade relacionamentos;
4. Identificar os métodos;
5. Identificar os relacionamentos entre os objetos;
6. Definir a herança;
7. Definir as colaborações;
8. Definir a agregação.

Agregação x composição

A associação tem duas formas particulares: a **agregação** e a **composição**. As duas são muito parecidas, elas relacionam um objeto composto com suas partes. Por exemplo, a associação entre uma universidade e seus departamentos é uma agregação, a associação entre um carro e suas partes é uma composição. A diferença entre os dois é que a composição é mais ``física": uma parte não pode pertencer a dois objetos compostos ao mesmo tempo (um motor não pode pertencer a dois carros ao mesmo tempo), e uma parte não pode existir sem o objeto composto (mas o composto pode ``sobreviver" às suas partes).

A diferença entre os dois não é sempre clara. Por exemplo, a associação entre uma casa e os suas paredes é uma composição, mas entre uma peça da casa e um parede, é uma agregação (a parede pertence às duas peças e não desaparece com a destruição de uma das duas peças). Na dúvida, é melhor usar a agregação que é menos restrita, ou até uma associação simples.

Agregação e composição aceitam multiplicidade, só do lado das partes para a composição (um carro é composto de quatro rodas) e dos dois lados com a agregação (uma palavra pode pertencer a várias frases e uma frase possui várias palavras).

Geralmente, a composição implica uma forma de propagação de algumas propriedades. Por exemplo, quando o objeto composto morrer, as partes morrem também (definição da composição), ou quando um carro se mover, as partes se movem também. Todas as propriedades não são propagadas, por exemplo, um carro pode ser vermelho e o motor não. O tempo de vida do membro depende do tempo de vida do objeto composto.

Agregação e composição são representadas com associações acrescentadas de um losango do lado do objeto composto: losango vazio para a agregação e losango preto para a composição. A composição pode também ser representada com uma classe com as partes dentro do retângulo.

Diagrama de estados

Representam o comportamento interno de uma classe durante sua vida, mostrando como específicos eventos (métodos) podem mudar as fases da vida de mesma. Por exemplo, após uma avaliação, uma Solução_de_Projeto poderá tornar-se Aceita ou Rejeitada.

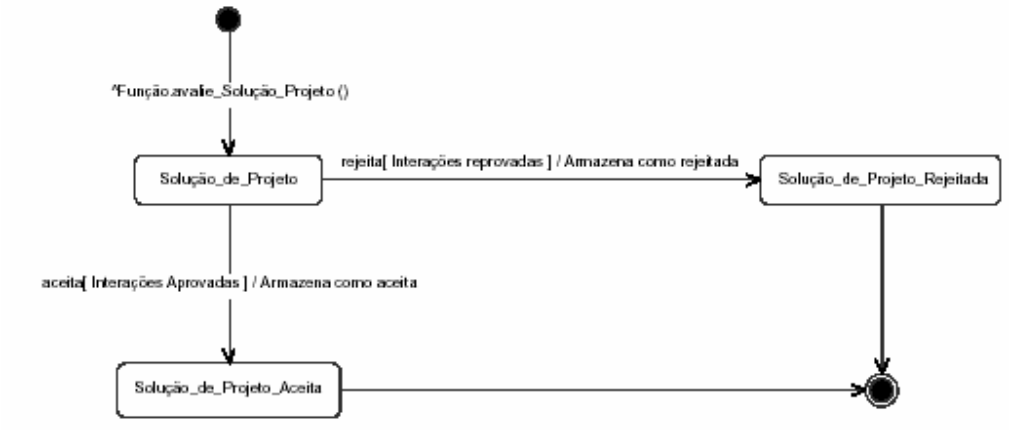


Diagrama de seqüência

Capturam e representam a colaboração necessária entre classes, ou Categorias, através de seus métodos. Basicamente, os aspectos comportamentais dos objetos são focalizados, mostrando quais métodos são necessários para satisfazer um "Use Case" específico. O "Use Case" avalie_Soluções_Projeto mostra como as categorias Funções e Soluções_de_Projeto colaboram para atender uma funcionalidade específica do sistema.

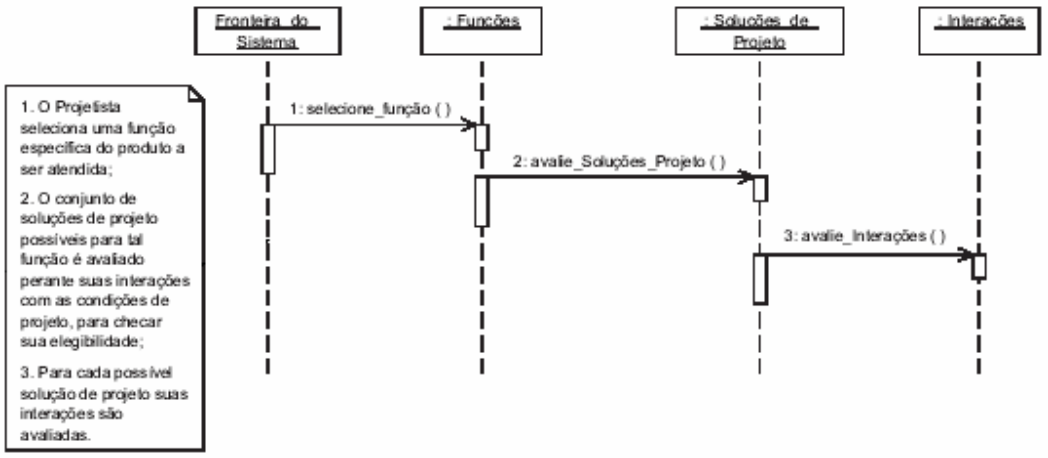


Diagrama de atividades

Representa o conjunto de passos a serem executados por um método, mostrando como uma operação pode(ria) ser implementada. São compostos por atividades (retângulos arredondados), transições (setas), barras de sincronização, decisões (losango) e marcadores de início e fim.

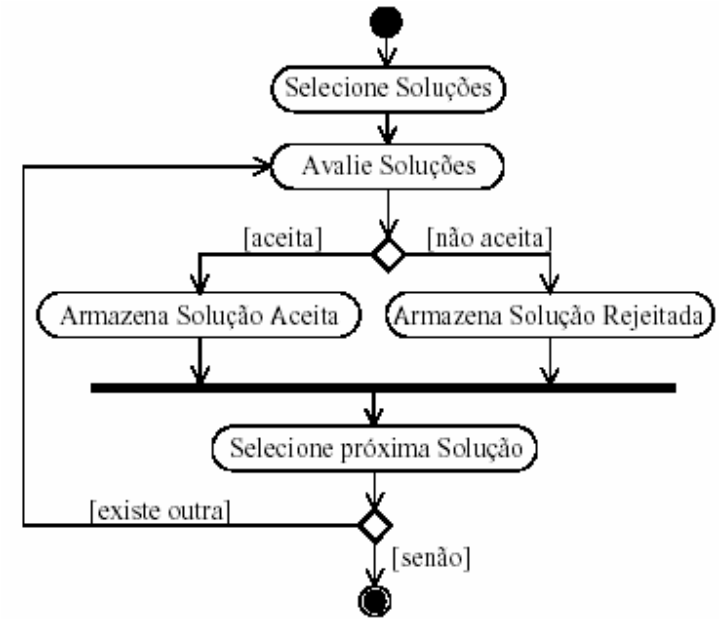


Diagrama de colaboração/comunicação

O diagrama de colaboração, que a partir da UML 2.0 passou a se chamar diagrama de comunicação, possui uma função similar ao diagrama de seqüência. Mas ao contrário deste, o seu foco não é no tempo, e sim na organização dos objetos. Por isso, esse diagrama mostra explicitamente as conexões entre os objetos (o que o de seqüência não faz), enquanto ele deve acrescentar números de seqüência às mensagens para indicar a ordem de chamada. Outra diferença é que esse diagrama NÃO mostra explicitamente o retorno das chamadas.

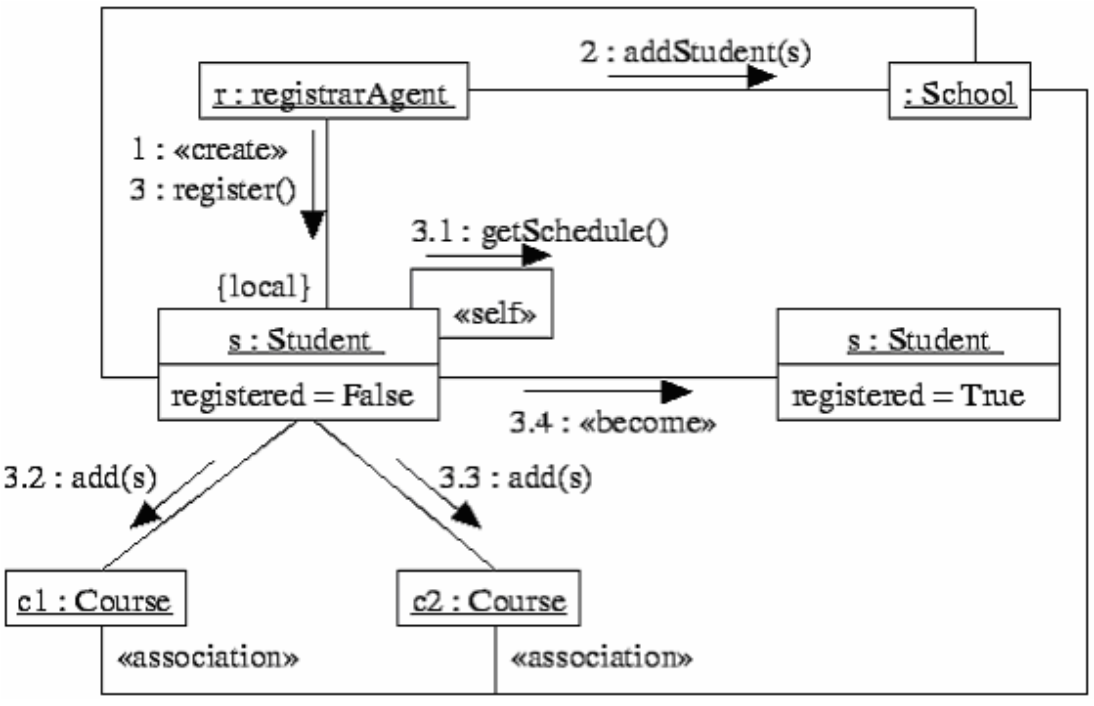


Diagrama de implementação

O diagrama de implantação (ou distribuição) é usado para sistema distribuídos. Ele permite apresentar a topologia de uma rede de "máquinas" e qual processo (um componente executável) cada "máquina" vai rodar.

As "maquinas" são chamadas de nós. Um nó apresenta uma fonte computacional, sendo normalmente um processador com alguma memória. Um nó é representado por um cubo. Alguns de seus objetivos é modelar um sistema embarcado, sistemas distribuídos ou sistemas cliente / servidor.

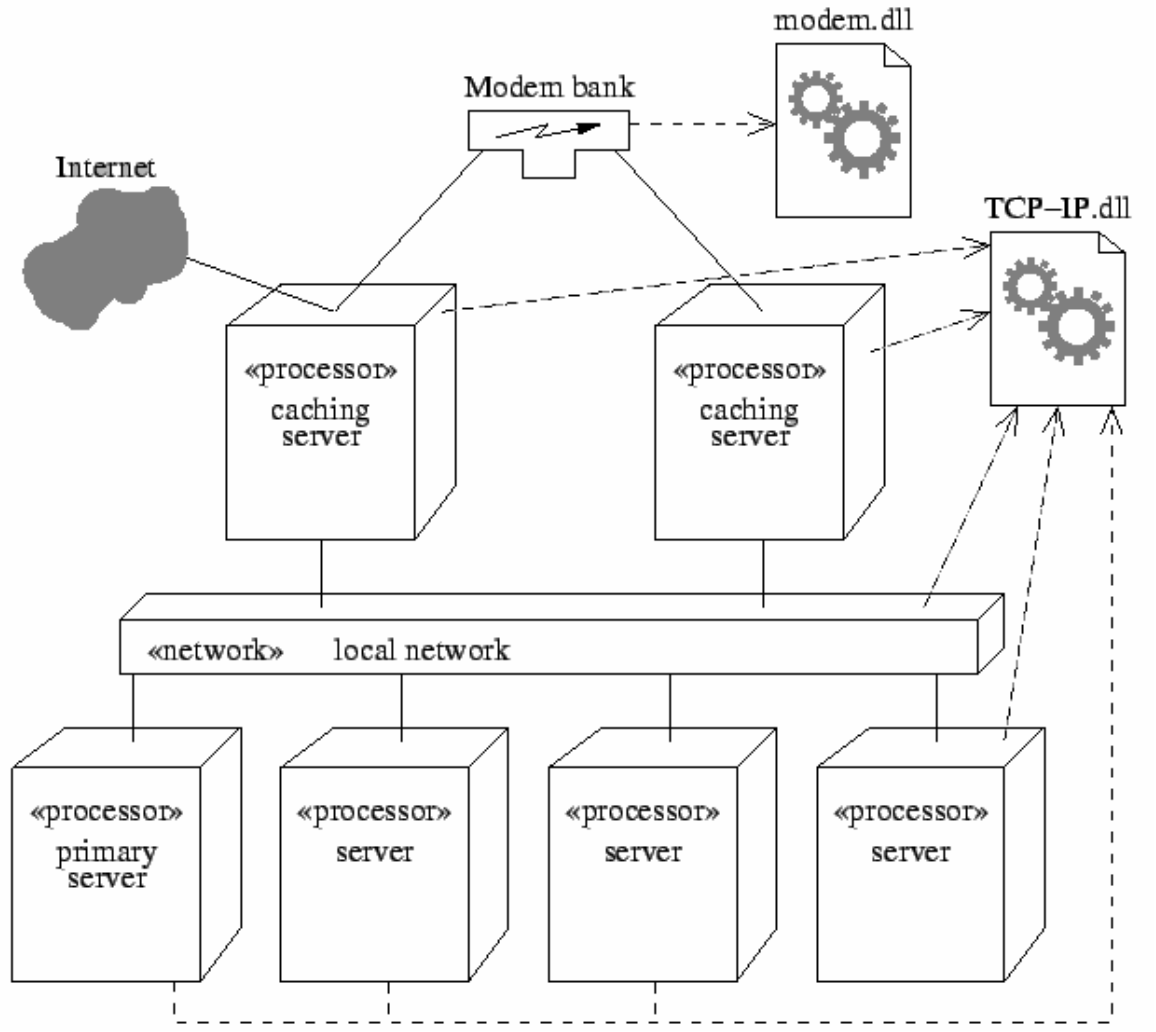
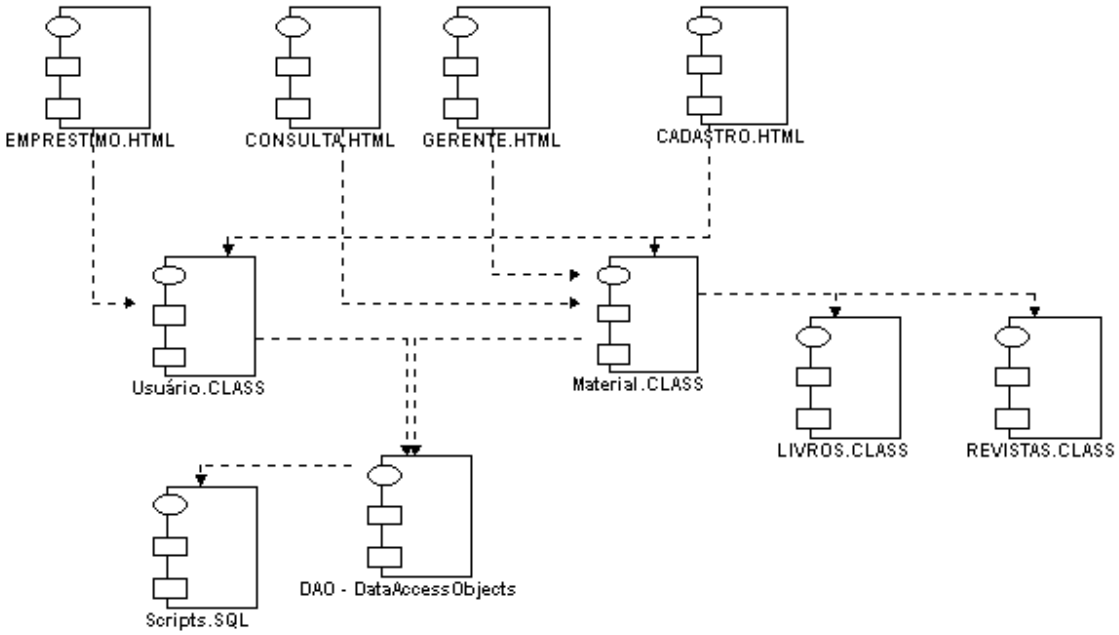


Diagrama de componentes

O diagrama de componentes mostra a organização entre arquivos de código fonte, bibliotecas, tabelas de banco de dados, etc. A relação a mais usada é a de dependência, mostrando como um arquivo de código fonte depende de um outro que ele incluí, o como um executável depende de uma biblioteca por exemplo.

Um componente é um **parte física** do sistema. Muitas vezes um componente mostra um arquivo específico do sistema. A UML reconhece: executáveis (na figura), bibliotecas (ver diagrama de implementação), tabelas (uma tabela de um banco), documentos (textos livres, ajuda) e arquivos (scripts).



Utilização dos diagramas

Os **Diagramas de Casos de Uso** são utilizados para modelar o contexto (os requisitos) do sistema.

Os **Diagramas de Classes** modelam as classes, interfaces e seus relacionamentos. Modelam os aspectos estáticos do sistema. Já os **Diagramas de Objetos** mostram o conjunto de objetos e seus relacionamentos, existentes em um determinado momento

Os **Diagramas de Interação** mostram as interações existentes entre um conjunto de objetos e seus relacionamentos, incluindo as mensagens trocadas. Podem ser de dois tipos: **Seqüência**: enfatiza a ordenação das mensagens ao longo do tempo e **Colaboração**: enfatiza a organização dos objetos que participam de uma interação. Ambos descrevem o comportamento de VÁRIOS objetos em um ÚNICO caso de uso.

Os **Diagramas de Estado** descrevem um comportamento de UM objeto, através de VÁRIOS de casos de uso. Seu comportamento é caracterizado pela sua resposta a eventos externos ao seu contexto.

Os **Diagramas de Atividades** apresentam uma seqüência geral de ações para VÁRIOS objetos e casos de uso. Mostra o fluxo de execução das atividades, os aspectos dinâmicos do sistema

O **Diagrama de Componentes** modela os aspectos físicos do sistema, mostra a organização e dependências entre um conjunto de componentes.

O **Diagrama de Implantação** mostra a configuração do sistema (unidades de processamento e os componentes que rodam nele) em tempo de execução.

Na versão UML 2.0 existem também os **Diagramas de Visão Geral** e o **Diagrama Temporal**. O primeiro é uma variação do diagrama de atividades que mostra de uma forma geral o fluxo de controle dentro de um sistema ou processo de negócios. Cada nó ou atividade dentro do diagrama pode representar outro diagrama de interação. Já o Diagrama Temporal mostra a mudança de estado de um objeto numa passagem de tempo, o que é muito útil para sistemas de tempo real.

A **arquitetura de sistemas em camadas**, aplicada aos sistemas desenvolvidos sobre a orientação a objetos, determina a distribuição das classes por três camadas.

- A primeira camada identificada como **camada de apresentação** abrange as classes que são responsáveis pela interação dos usuários com o sistema.
- A segunda camada identificada como **camada de negócios** controla as regras de negócio da aplicação. As classes de negócio armazenam os requisitos que determinam como calcular, como processar, o que persistir e o que recuperar.
- A terceira camada identificada como **camada de persistência** controla a persistência dos dados. Os dados vivem temporariamente nas instâncias da segunda camada, mas devem ser mantidos ao término da aplicação. Para isso,é necessário armazená-lo em um meio físico permanente. Assim, esta camada trata como os dados são persistidos em simples arquivos ou em bancos de dados.

Padrões de Projetos (*Design Patterns*)

Patterns são soluções genéricas e reutilizáveis, aplicáveis em classes de problemas bem conhecidos. Padrões são definidos como a solução de problemas recorrentes. Qualquer linguagem orientada a objetos pode aplicar um padrão de desenvolvimento a uma aplicação, permitindo a comunicação do seu problema, estratégias e boas práticas de programação, além da solução para o problema proposto.

Estas soluções foram experimentadas por vários desenvolvedores experientes, e por isso podem ser reusadas de forma confiável. Para identificar um padrão, é necessário detectar o problema que ocorre com um projeto, dadas as suas características. Os padrões costumam incluir implementações de alto nível, recomendadas para compor uma determinada solução.

Estrutura do *Pattern*: Nome, Problema, Solução e Conseqüências. Organizados pelo seu escopo e propósito:

Abstract Factory: Provê uma interface para criação de famílias de objetos relacionados ou interdependentes. Remove a dependência entre o cliente, que usa os objetos, e a classe dos objetos produzidos.

Adapter: Converte a interface de uma classe em outra, esperada pelo cliente. O Adapter permite que classes que antes não poderiam trabalhar juntas, por incompatibilidade de interfaces, possam agora fazê-lo.

Bridge: Separa uma abstração de sua implementação, de modo que ambas possam variar independentemente.

Builder: Provê uma interface genérica para a construção incremental de agregações. Um Builder esconde os detalhes de como os componentes são criados, representados e compostos.

Chain of Responsibility: Encadeia os objetos receptores e transporta a mensagem através da corrente até que um dos objetos a responda. Assim, separa (provê *loose coupling*) objetos transmissores dos receptores, dando a chance de mais de um objeto poder tratar a mensagem.

Command: Encapsula uma mensagem como um objeto, de modo que se possa parametrizar clientes com diferentes mensagens. Separa, então, o criador da mensagem do executor da mesma.

Composite: Compõe objetos em árvores de agregação (relacionamento parte-todo). O Composite permite que objetos agregados sejam tratados como um único objeto.

Decorator: Anexa responsabilidades adicionais a um objeto dinamicamente. Provê uma alternativa flexível para extensão de funcionalidade, sem ter que usar Herança.

Facade: Provê uma interface unificada para um conjunto de interfaces em um subsistema. O Facade define uma interface alto nível para facilitar o uso deste subsistema.

Factory Method: Define uma interface para criação de um objeto, permitindo que as suas subclasses decidam qual classe instanciar. O Factory Method deixa a responsabilidade de instanciação para as subclasses.

Flyweight: Usa o compartilhamento para dar suporte eficiente a um grande número de objetos com alto nível de granularidade.

Interpreter: Usado para definição de linguagem. Define representações para gramáticas e abstrações para análise sintática.

Iterator: Provê um modo de acesso a elementos de um agregado de objetos, seqüencialmente, sem exposição de estruturas internas.

Mediator: Desacopla e gerencia as colaborações entre um grupo de objetos. Define um objeto que encapsula as interações dentre desse grupo.

Memento: Captura e externaliza o estado interno de um objeto (captura um "snapshot"). O Memento não viola o encapsulamento.

Observer: Provê sincronização, coordenação e consistência entre objetos relacionados.

Prototype: Especifica os tipos de objetos a serem criados num sistema, usando uma instância protótipo. Cria novos objetos copiando este protótipo.

Proxy: Provê Design para um controlador de acesso a um objeto.

Singleton: Assegura que uma classe tenha apenas uma instância e provê um ponto global de acesso a ela.

State: Deixa um objeto mudar seu comportamento quando seu estado interno muda, mudando, efetivamente, a classe do objeto.

Strategy: Define uma família de algoritmos, encapsula cada um deles, e torna a escolha de qual usar flexível. O Strategy desacopla os algoritmos dos clientes que os usa.

Template Method: Define o esqueleto de um algoritmo em uma operação. O Template Method permite que subclasses componham o algoritmo e tenham a possibilidade de redefinir certos passos a serem tomados no processo, sem contudo mudá-lo.

Visitor: Representa uma operação a ser realizada sobre elementos da estrutura de um objeto. O Visitor permite que se crie um nova operação sem que se mude a classe dos elementos sobre as quais ela opera.

Uso/conceitos de ferramentas de suporte à análise e projetos orientados a objetos

De um ambiente de desenvolvimento de software (ADS) espera-se mais do apoio à implementação, como ocorre com um ambiente de desenvolvimento de programas. Espera-se apoio aos demais ciclos de vida do software, a partir de editores dirigidos à sintaxe (inclusive gráficos), mecanismos de validação, entre outros.

Este tipo de ambiente de desenvolvimento de software recebe a denominação genérica CASE (*Computer-Aided Software Engineering*), pois apóia a evolução do software em cada etapa do seu ciclo de vida.

Em linhas gerais, ferramentas CASE apóiam a execução de atividades do desenvolvimento do software de forma automatizada. Em alguns casos, implementam um ambiente relativamente refinado no qual várias atividades de especificação ou codificação são apoiadas por recursos computacionais.

Dependendo da atividade suportada podem ser classificados em Lower CASE, provendo suporte à codificação, teste, depuração e manutenção do código ou Upper CASE, suportando tarefas de análise e projeto de sistemas.

Algumas ferramentas CASE típicas são geradores de código, editores UML, ferramentas de Refatoração (*refactoring*) e ferramentas de gerenciamento de configuração, como *softwares* de controle de versões.

Por exemplo, ao se utilizar ferramentas CASE para um produto de software de banco de dados, deve-se:

- Modelar processos de negócio e do mundo real, e o fluxo de dados
- Desenvolver Modelos de Dados na forma de diagramas E-R
- Desenvolver a descrição dos processos e funções
- Produzir *scripts* SQL de criação de bancos e *Stored Procedures*

Algumas ferramentas CASE conhecidas são:

- ArgoUML
- Eclipse (com plugins)
- Oracle Designer
- Rational Rose da IBM
- Visual Paradigm for UML

Engenharia de Software

Princípios de Engenharia de Software

A Engenharia de Software (ES) surgiu em meados dos anos 70 numa tentativa de contornar a crise do software e dar um tratamento mais sistemático e controlado ao desenvolvimento de sistemas de software complexos.

A engenharia de software envolve o uso de modelos abstratos e precisos que permitem ao engenheiro especificar, projetar, implementar e manter sistemas de software, avaliando e garantido suas qualidades. Além disto, a engenharia de software deve oferecer mecanismos para se planejar e gerenciar o processo de desenvolvimento

A tarefa fundamental na execução de um processo de planejamento é definir e agrupar os pacotes de trabalho, com as respectivas entregas de cada pacote. Um *Baseline* (ou linha básica) é o marco de referência no desenvolvimento, que se caracteriza pela entrega e aprovação de um ou mais itens de configuração.

Engenharia de sistemas

A engenharia de sistemas preocupa-se com os aspectos do desenvolvimento de sistemas baseados em computador (hardware, software, engenharia de processos). A engenharia de software faz parte deste processo.

Os engenheiros de sistema estão envolvidos na especificação, projeto arquitetural, integração e desenvolvimento de sistemas de informação.

Ciclo de Vida

O desenvolvimento de um produto de software foi modelado de acordo com a engenharia convencional. Para PRESSMAN, o ciclo de vida do software é dividido em seis partes, descritas a seguir:

- **Avaliação:** É a fase correspondente à engenharia de sistemas, iniciando com os requisitos.
- **Análise:** Processo de coleta dos requisitos intensificado e concentrado no software.
- **Projeto:** Processo de múltiplos passos, concentra-se na estrutura de dados, arquitetura de software, detalhes procedimentais e caracterização da interface. A qualidade pode ser avaliada nesta etapa.
- **Implementação:** Etapa de codificação que transforma o projeto em uma forma computacional.
- **Teste:** Concentra-se em testar os aspectos lógicos do sistema, e também nos aspectos funcionais, descobrindo erros e que os resultados reais concordem com os resultados exigidos pelo sistema.
- **Manutenção:** Responsável por adaptar e evoluir o software de acordo com exigências futuras.

O ciclo de qualidade PDCA (*Plan, Do, Check, Act*) pode ser relacionado com o ciclo do desenvolvimento, sendo a etapa *Plan* equivalente às três primeiras (Avaliação, Análise e Projeto), e as demais em seguida.

Processos de Software

Um processo de *software* é um conjunto de práticas, estruturas organizacionais, tecnologias, procedimentos e artefatos que são necessários para conceber, desenvolver, implantar e manter um produto de *software*.

Enxergar o desenvolvimento de um *software* como um processo ajuda a identificar as diferentes dimensões da tarefa (tecnologia, organização, marketing, economia), evitar problemas e estabelecer práticas efetivas.

As atividades genéricas em todos os processos de software são:

- **Especificação:** O que o sistema deve fazer e as restrições aplicadas ao seu desenvolvimento.
- **Desenvolvimento:** A produção do sistema de software.
- **Validação:** Verifica se o sistema atende às especificações do cliente.
- **Evolução:** Modificar o software em resposta a necessidades de alteração.

Análise

Na fase de análise, também denominada análise de requisitos, são estabelecidos os requisitos do sistema a ser desenvolvido. Nesta fase, o engenheiro de software necessita compreender o domínio do problema no qual está trabalhando. Os requisitos e a modelagem conceitual do sistema são documentados e revisados.

A análise de requisitos começa ao se reconhecer que um problema necessita da tecnologia de informação como solução ou ao surgir uma idéia de um novo negócio ou de um novo sistema na empresa. Esta fase é concluída ao se ter uma descrição completa do comportamento do software a ser construído, documentado na Especificação de Requisitos. Aqui são feitas: a análise do problema, descrição do produto e avaliação da especificação.

Projeto (design) e Codificação

O projeto do software é, na realidade, um processo de múltiplos passos que reforça quatro atributos importantes: estrutura de dados, arquitetura do software, detalhe procedural e projeto da interface. A fase de projeto tem por objetivo traduzir os requisitos definidos em uma representação de software com detalhes suficientes para que possa ser implementado. Como os requisitos, o projeto é documentado (Especificação de Projeto), sendo este realizado com base na Especificação de Requisitos. Pode-se verificar que quanto mais refinado (detalhado) for a fase de projeto, mais clara será a fase de construção.

Na fase de construção, os programas são codificados, as bases de dados criadas e os módulos do software integrados.

Verificação x Validação

Uma vez que o código foi gerado, segue-se a fase de avaliação da qualidade do software. Esta avaliação deve ser realizada através de certificação utilizando inspeção, *walkthrough*, prova formal ou testes.

As **inspeções formais** são uma técnica de revisão sistemática do software ou de alguns de seus componentes, executada, sistematicamente, ao final de cada fase do projeto, com o objetivo único de encontrar erros. Todo o material gerado é lido, os erros anotados e uma estatística dos erros encontrados é mantida, para fins de posterior estudo da eficácia do procedimento. É uma técnica preventiva, barata, que depende da experiência do inspetor e é pouco eficaz para fatores operacionais.

Verificação: “Estamos construindo certo o produto?”, ou seja, o software deve estar de acordo com a sua especificação.

Validação: “Estamos construindo o produto certo?”, ou seja, o software deve atender às necessidades dos usuários.

Testes

Os testes concentram-se em validar os procedimentos lógicos internos do programa, garantindo que todos os comandos foram testados e que o comportamento funcional externo do sistema produz os resultados esperados para determinadas entradas de dados. **Pressman** afirma que a atividade de teste de software é um elemento crítico da garantia de qualidade de software e representa a última revisão de especificação, projeto e codificação. A intenção da realização de testes é achar erros, e é necessário que estes sejam planejados com seus objetivos claramente definidos. Na análise dos resultados deve-se procurar identificar a falha que causou o erro evidenciado pelo teste. Testar exaustivamente um software nem sempre é possível, por isso os testes não podem assegurar a correção total de um sistema. Entretanto, a utilização de testes combinados com algumas técnicas de controle da qualidade contribui para se obter um nível aceitável de confiabilidade.

Algumas dificuldades que podem ser encontradas durante o ciclo dos testes são:

- Detecção de falhas se dá através da ocorrência de defeitos
- É necessária a presença de uma especificação completa e não ambígua
- Falhas nos requisitos podem não ser detectadas
- Certas tarefas de testes não podem ser automatizadas, dificultando os testes exaustivos
- Veredictos de testes dependem das saídas esperadas, mas produzi-las pode ser difícil ou impossível

Note que os testes são apenas uma das maneiras de encontrar erros no sistema. Inspeções e revisões técnicas também devem ser executadas continuamente após a codificação com o objetivo de monitorar a qualidade.

- A atividade de teste não prova a ausência de erros, apenas a existência dos mesmos;

- Bons casos de teste são aqueles que encontram falhas no sistema até então não descobertas;
- Bons casos de teste são projetados levando em conta os requisitos do projeto;
- Um critério utilizado para determinação do esforço a ser gasto na atividade de teste de software é verificar qual o grau de severidade das conseqüências advindas do seu mau funcionamento;
- A probabilidade de encontrar um erro numa determinada parte do sistema é proporcional ao número de erros já encontrados nesta parte;

Garantia da qualidade

As atividades de garantia de qualidade estão focadas na utilização de processos, para gerenciar, produzir e entregar a solução, podendo ser realizadas pelo Sistema de Informação Gerencial, pelo patrocinador do projeto ou por uma terceira pessoa que revise estas atividades.

A Garantia da Qualidade não se refere diretamente as *deliverables* específicas. Esta refere-se ao processo usado na criação das *deliverables*. Em geral a atividade de garantia da qualidade esta focada no processo de gerenciamento e de entrega da solução, e poderá ser realizada por um gestor, por um cliente ou por uma terceira parte revisora. Por exemplo, um revisor independente de projeto talvez não esteja habilitado em dizer se o conteúdo de uma *deliverable* em particular é aceitável ou não. Entretanto, eles deverão estar habilitados a dizer, baseado no processo utilizado para criação da *deliverable*, se está parece ser aceitável ou não.

O **Controle de Qualidade** é uma atividade separada da garantia de qualidade, mas que também faz parte do gerenciamento da qualidade. As atividades de controle de qualidade são realizadas continuamente no andamento do projeto para verificar se o gerenciamento do projeto e as *deliverables* são de alta qualidade.

Manutenção

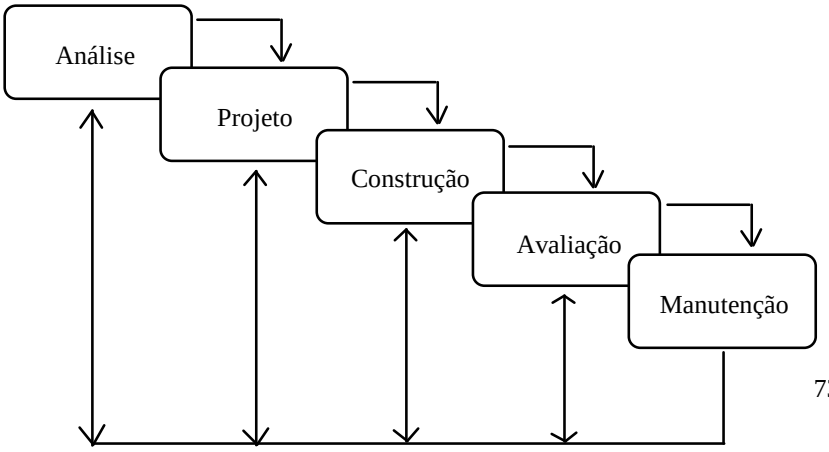
Além da fase de Desenvolvimento e da Validação, o terceiro processo de desenvolvimento de software envolve a manutenção do sistema, onde o software sofre correções, adaptações e ampliações para corrigir erros encontrados após a entrega do produto, atender os novos requisitos do usuário e mudanças na tecnologia.

O gerenciamento de risco nesta fase é fundamental, dado que 70% dos custos de um sistema de software correspondem à manutenção de um sistema. Por isso é importante controlar a manutenibilidade durante o desenvolvimento, através de métricas e modelos. Os tipos de ações tomadas durante a fase de manutenção são:

- **Preventiva:** previne falhas e defeitos que não geraram falhas, é implementada para eliminar as causas de uma possível não-conformidade, defeito ou outra situação indesejável, a fim de prevenir sua ocorrência.
- **Corretiva:** problemas decorrentes de defeitos, é implementada para eliminar as causas de uma não-conformidade, de um defeito ou de outra situação indesejável existente, a fim de prevenir sua repetição.
- **Adaptativa:** após mudança no ambiente, software ou hardware
- **Perfectiva:** melhorar algum aspecto do sistema (ex: refactoring)

Modelos de ciclo de vida

Modelos de ciclo de vida descrevem as etapas do processo de desenvolvimento de sistemas e as atividades a serem realizadas em cada etapa. A definição dessas etapas e atividades possibilita prover marcos e pontos de controle para avaliação da qualidade e gerência do projeto.



Modelo Clássico ou Cascata

A primeira proposta deu origem ao *modelo tradicional* ou cascata. Nesse modelo as fases são executadas sistematicamente em seqüência.

O sistema é entregue ao usuário após se chegar a um resultado satisfatório na fase de avaliação, com um certo grau de confiança. É claro que o sistema ainda passará por alterações, devido principalmente a erros observados pelo usuário, ou por mudanças necessárias para melhor adaptação ao ambiente do usuário, ou por problemas de desempenho.

Existem diversas variações deste modelo, e uma das suas características é esgotar cada uma das fases antes de passar para a fase seguinte, o que evita desperdícios de esforços, facilita o planejamento (e cronogramas). As desvantagens: a dificuldade em acomodar alterações após o seu início, e a demora para começar a construção.

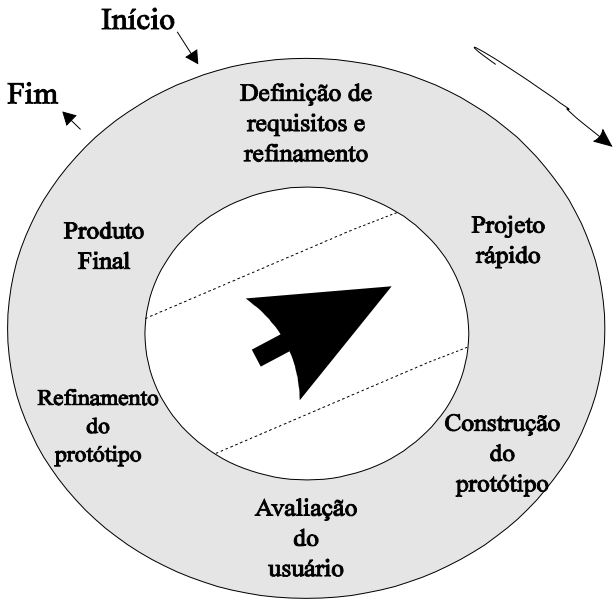
Modelo orientado a reuso

Reutilização de software é o processo de desenvolvimento de software cujo objetivo é utilizar a experiência adquirida e os produtos obtidos anteriormente, para o desenvolvimento de novos produtos, podendo ser aplicada nas fases de Análise, Projeto, Construção ou Avaliação. Assim sendo, a reutilização utiliza: código já testado anteriormente, projetos validados, especificações de requisitos, e / ou planos para testes.

A expressão "reutilização de software" indica uma atividade particular no desenvolvimento do software. O modelo é bem parecido com o tradicional, mas conta com um **repositório**, que guarda as informações adquiridas em fases anteriores.

Modelo Evolucionário ou Prototipagem

O modelo de ciclo de vida evolucionário surgiu propondo um desenvolvimento que expandisse o sistema gradativamente, permitindo que se obtivessem modelos do comportamento do software antecipadamente, os denominados protótipos. A prototipagem é uma forma de desenvolvimento incremental e contém quatro tipos diferentes: ilustrativo (telas), simulado (acesso ao banco de dados é simulado), funcional (subconjunto limitado) e evolucionário (aumenta gradualmente). Os três primeiros tipos são construídos para se atingir objetivos propostos a priori, sendo considerados descartáveis. Ao final o protótipo se tornará um produto operacional.



Tal como o processo de desenvolvimento convencional de software (modelo cascata), a prototipagem se inicia com a etapa de definição dos requisitos. Todos os objetivos a serem atingidos pelo software são definidos, sendo identificadas as áreas que merecem uma definição mais refinada.

A partir dos requisitos levantados na fase de definição, um projeto inicial é construído e apresentado ao usuário, refletindo os aspectos visíveis do software. Este projeto rápido leva à construção de um protótipo.

Um benefício é que o desenvolvedor do software pode compreender melhor o domínio sobre o qual está trabalhando, a partir das questões levantadas pelo usuário. Além disso partes do protótipo podem ser aproveitadas. Porém o custo normalmente é mais alto, e a construção do protótipo acaba atrasando o início da implementação final.

Técnicas de quarta geração

Não são um modelo especificamente dito, mas um termo amplo, cujo significado abrange ferramentas que possibilitam o desenvolvedor de software a especificar determinadas características do software num nível elevado, e o código executável seja gerado a partir dessas especificações automaticamente.

Algumas características: Linguagens não procedimentais de acesso a Bancos de Dados, geração de relatórios, definição e manipulação de telas, capacidade gráfica de alto nível, geração de código, etc.

Defensores reivindicam uma drástica redução no tempo de desenvolvimento de software em função de uma maior produtividade. Opositores afirmam que as ferramentas não são tão mais fáceis de usar do que ferramentas mais convencionais e que o

código gerado é ineficiente. O domínio de aplicações para as quais estas técnicas estão mais desenvolvidas são as aplicações comerciais, geralmente ligadas a grandes bancos de dados.

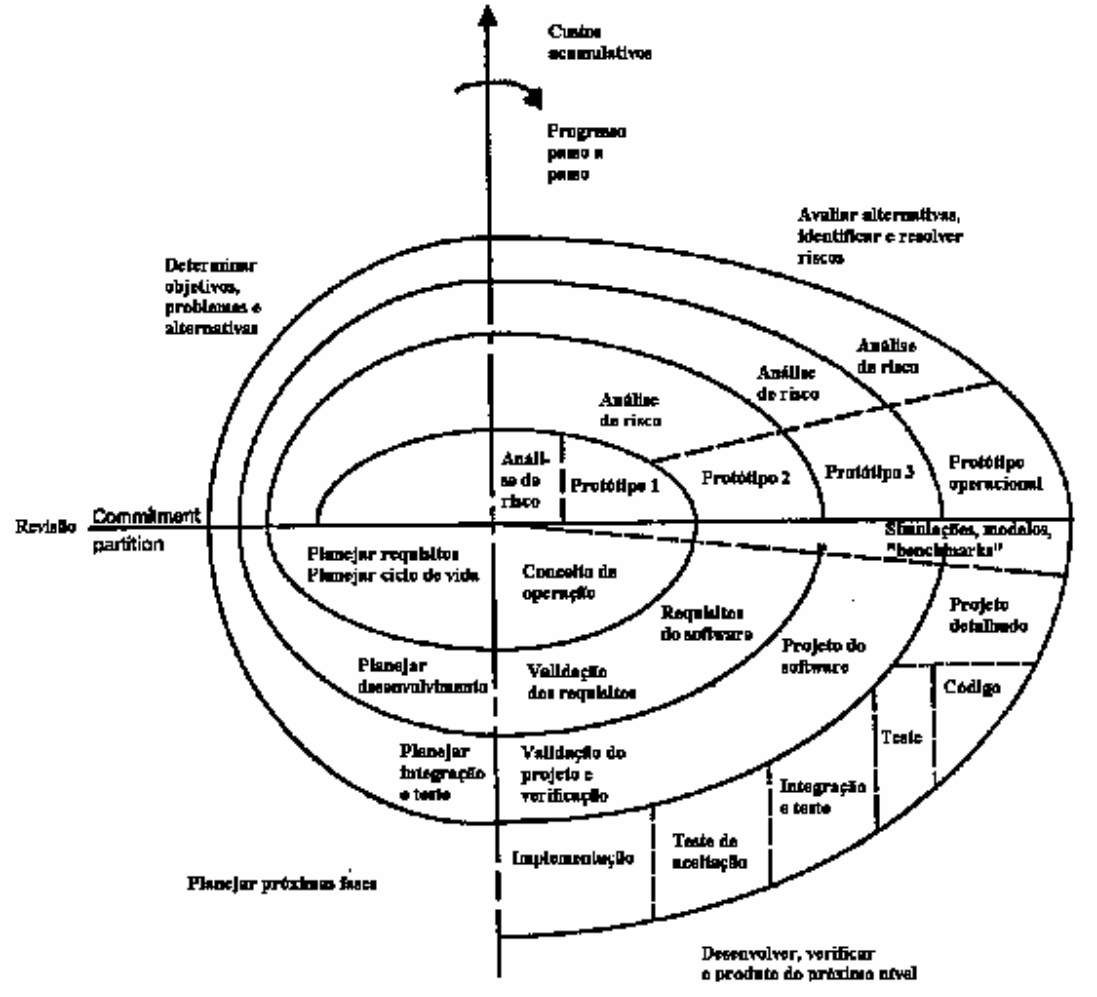
Existe um certo ganho de tempo de desenvolvimento das aplicações, especialmente as pequenas.

Modelo em Espiral

O modelo em espiral proposto por Boehm fornece uma estrutura de trabalho para a produção de software baseada em processo e níveis de risco permitindo a análise dos riscos em várias etapas do desenvolvimento. Esse modelo pode ser considerado um meta-modelo pois pode abranger diferentes outros modelos, desde o modelo cascata até os vários tipos de protótipos. É um modelo cíclico.

A análise dos modelos de ciclo de vida pode ser feita considerando-se a identificação monolítica ou incremental do processo de desenvolvimento. Na estrutura de trabalho monolítica, os detalhes de cada fase são completados em sua totalidade, antes do início da etapa seguinte. Sendo assim, o comportamento do software só pode ser avaliado no final do desenvolvimento. Na estrutura incremental, os detalhes podem ser retardados com o benefício de se produzir antecipadamente um protótipo mostrando o funcionamento do produto.

As vantagens: o foco nas opções de reuso e eliminação precoce de erros (qualidade desde o início), além de integrar a fase de desenvolvimento e manutenção. Alguns dos problemas é a dificuldade de controlar todo o processo, requerendo experiência em avaliação de risco, e precisa de refinamento para uso geral.



RAD - Desenvolvimento rápido de aplicações

A técnica de desenvolvimento rápido de aplicações (RAD) é um bom ponto de partida para a elaboração do projeto de interfaces. Usando de forma embutida a orientação a objetos, essa técnica fundamenta-se nos seguintes conceitos:

- projeto centrado no usuário
- prototipação rápida
- particionamento de processos em clientes e servidores
- modelagem de objetos

O Delphi, por exemplo, conta com várias ferramentas RAD que auxiliam o desenvolvimento, para a redução do tempo de desenvolvimento e manutenções improváveis. Adicionando, uma arquitetura aonde é possível a separação da GUI (*Graphic Unit Interface*), das regras de negócio lógicas e do desenho do banco de dados.

O desenho do banco de dados, a construção das metodologias de negócio e o desenho e a criação das janelas de entrada são efetivados dentro da aplicação. O desenvolvimento, então paralelamente, resultará em uma maior rapidez. O processo de codificação reduz drasticamente ao herdas as janelas de entrada, módulos, etc.

Testes

De acordo com [PRESSMAN], a atividade de teste pode ser conduzida em quatro fases:

- **Teste de Unidade:** concentra-se no esforço de verificação da menor unidade de projeto de software chamado de unidade ou módulo.
- **Teste de Integração:** é uma técnica sistemática para a construção da estrutura de programa, ao mesmo tempo sendo realizados testes para descobrir erros associados a interfaces.
- **Teste de Validação:** o *software* está completamente montado como um pacote, e a validação é definida como bem-sucedida quando ele funciona de uma maneira razoavelmente esperada pelo cliente.
- **Teste de Sistema:** é uma série de diferentes testes, cujo principal propósito é pôr o sistema completamente à prova, verificados quanto à sua integração adequada e funções atribuídas.

Nenhuma das técnicas de teste é completa, ou seja, suficiente para garantir a qualidade da atividade de teste.

Tipos de Testes

Podemos classificar os métodos de teste de acordo com suas características básicas:

- **Testes estáticos ou testes humanos:** não são feitos através da execução real do programa, mas sim através da execução conceitual do mesmo. Métodos classificados como estáticos são o de *walkthrough*, inspeções *e peer rating*. São utilizados principalmente para validar as primeiras etapas do projeto.
- **Testes dinâmicos:** seguem o modelo tradicional de teste de programa, no qual o programa é executado com alguns casos de teste e os resultados são examinados para verificar se o programa operou de acordo com o esperado. Usados na validação do código em módulos e na integração do sistema.
- **Testes funcionais:** é o teste exaustivo, no entanto o domínio de um programa é muito grande e são criadas formas de derivar um conjunto de dados de teste representativo que consiga exercitar completamente a estrutura do sistema. São métodos de testes funcionais ou de **caixa preta**, o particionamento de equivalência, a análise de valor limite, a técnica de grafo de causa-efeito e testes de comparação.
- **Testes estruturais:** diferentemente dos testes funcionais, que se preocupam com a função que o programa desempenha sem se preocupar com como a função foi implementada, o teste estrutural enfoca a implementação e a estrutura da função. Geralmente usado durante a fase de codificação. A intenção é encontrar dados de teste que terão cobertura de todas estruturas presentes na representação formal.
- **Testes de unidade:** concentra-se no esforço de verificação da menor unidade de projeto de software, o módulo. Este teste baseia-se sempre na **caixa branca**, e são verificados: a interface com o módulo (fluxo de informações de dentro para fora), estrutura de dados local, condições de limite, todos os caminhos independentes são executados pelo menos uma vez, e caminhos de tratamento de erro.
- **Testes de integração:** é uma técnica sistemática para a construção da estrutura de programa, que procura descobrir erros relacionados à interface. A partir dos módulos testados no nível de unidade, é construída a estrutura do programa que foi determinada pelo projeto. A integração pode ser incremental (*top-down ou bottom-up*) ou não-incremental.

- **Testes de validação:** pode ser considerado bem sucedido quando o software funciona da maneira esperada pelo cliente. Verifica-se se o produto certo foi construído, seguindo a especificação de requisitos do software. A validação do software, na fase de testes, é realizada por meio de uma série de testes de caixa preta que demonstram a conformidade com os requisitos.
- **Testes alfa e beta:** são os testes de aceitação, feitos pelo usuário, que dificilmente opera o sistema da forma prevista, e visam descobrir erros cumulativos que poderiam deteriorar o sistema no decorrer do tempo. Os testes alfa são conduzidos em ambiente controlado (do desenvolvedor). Já o teste beta é realizado em uma ou mais instalações do cliente pelo usuário final do software.
- **Teste de segurança:** tenta verificar se todos os mecanismos de proteção embutidos em um sistema o protegerão de acessos indevidos. O analista pode tentar adquirir senhas via contatos externos, atacar o sistema com software customizado, desarmar o sistema negando serviço a outros, etc...
- **Testes de estresse:** feito para confrontar o sistema com situações anormais. O teste de estresse executa o sistema de forma que exige recursos em quantidade, freqüência ou volume anormais.
- **Teste de desempenho:** é idealizado para testar o desempenho de *runtime* do software dentro do contexto de um sistema integrado. Algumas vezes são combinados com os de estresse utilizando instrumentação de hardware e de software (para medição dos recursos com ciclo de processador).

Walkthough

Os objetivos do *walkthrough* são descobrir erros de função, lógica ou implementação, e verificar os requisitos, uniformidade e facilidade de administração do software. É uma técnica manual, executada em reuniões formais com um moderador, e inclui os desenvolvedores. É feita uma leitura do produto, e algumas simulações são realizadas. O propósito da técnica de walkthrough é estimular a discussão.

Técnica de prova de corretitude

Em um nível informal, ela reduz o número de passos envolvidos em um walkthrough. Em um nível mais formal, a lógica matemática é aplicada para dar suporte ao problema de provar que um programa está de acordo com sua especificação. A técnica consiste em validar a consistência de uma saída de acordo com o programa e a entrada.

Quando linguagens de programação são definidas formalmente (sintaxe e semântica formal), pode-se provar matematicamente que um programa está correto em relação à especificação formal. Ex.:

```
{x = 5}          Especificação: condição inicial
x: = x + 1
{x = 6}          Especificação: condição final
```

Simulações

É a utilização de um modelo executável para representar o comportamento de um objeto. Esta ferramenta de testes é bastante útil, contudo é mais empregada em sistemas de tempo real onde a interface com o mundo real é crítica e a integração com o hardware do sistema é fundamental.

Testes de caixa preta

Também conhecidos como **Testes Funcionais**, pois os métodos de teste de caixa preta concentram-se nos requisitos funcionais do software. Ele procura descobrir funções incorretas ou ausentes, erros de interface ou estruturas de dados, erros de desempenho e de inicialização e término. Os testes são gerados através do estudo de suas entradas e saídas.

- **Particionamento de Equivalência (ou Domínio):** é um método que divide o domínio de entrada de um programa em classes de dados a partir das quais os casos de teste podem ser derivados:
    - Inicialmente determina-se entre entradas válidas (pertencentes ao domínio) e inválidas para verificar como o sistema comporta-se com as últimas.

- Para o domínio de dados de entradas válidas devem ser identificadas partições (classes) para os quais o sistema tenha comportamento semelhante.
- O particionamento de equivalência procura definir um caso de teste que descubra classes de erros, assim reduzindo o número total de casos de teste que devem ser desenvolvidos.

- **Análise de Valor Limite:** é uma técnica de projeto de casos de teste que serve como complemento para a técnica de particionamento de classes. Ao invés de selecionar qualquer valor de uma classe de equivalência, a análise do valor limite leva à seleção de casos de teste nas "extremidades" da classe. Em vez de se concentrar somente nas condições de entrada, ela deriva os casos de teste também do domínio de saída.
- **Grafos de Causa-Efeito:** é uma técnica de projeto de casos de teste que oferece uma representação concisa das condições lógicas e das ações correspondentes. A técnica segue quatro passos:
  - As causas (condições de entrada) e efeitos (ações) são relacionados para um módulo e um identificador é atribuído a cada um.
  - Em seguida é gerado um grafo de causa-efeito.
  - O grafo é convertido em uma tabela de decisão.
  - As regras da tabela são convertidos em casos de teste.

Testes de caixa branca

O teste de caixa branca ou **Teste Estrutural** visa conhecer o funcionamento interno de um produto e verificar que a sua operação tem um desempenho de acordo com as especificações. Razões para a sua execução são que erros lógicos e pressuposições incorretas ocorrem em casos especiais, e também que erros tipográficos são aleatórios, e ocorrem ao converter código da fase de projeto.

- **Teste do Caminho Básico:** possibilita que o projetista do caso de teste derive uma medida da complexidade lógica de um projeto procedimental e um conjunto básico de caminhos de execução. Os casos de teste derivados para exercitarem o conjunto básico têm a garantia de executar cada instrução do programa pelo menos uma vez durante a atividade de teste. Para determinar esses caminhos, uma técnica bastante utilizada é a *Complexidade Ciclomática*:
  - **Complexidade Ciclomática:** é uma medida definida pelos engenheiros de software que tentam captar a complexidade de cada rotina. A complexidade ciclomática pode ser calculada das seguinte formas:
    - Seja n o número total de predicados lógicos (comparações, expressões booleanas) que aparecem em uma rotina. A complexidade é dada então por $n+2$.
    - [SEI] Seja o programa um grafo de fluxo onde cada nó representa uma decisão e cada arco um caminho possível. A complexidade é dada por: *número de arcos – número de nós + 2*.
- **Teste de condição:** é um método de projeto de casos de teste que põe à prova as condições lógicas contidas num módulo de programa. Este método concentra-se em testar cada condição do programa e propõe-se a descobrir as seguintes classes de erros: erros de operadores booleanos, relacionais ou aritméticos, quanto à sua corretitude, falta ou excesso.
- **Teste de fluxo de dados:** esse método seleciona caminhos de teste de um programa de acordo com as localizações das definições e usos de variáveis no programa. Ele é útil para selecionar caminhos de teste de um programa que contenha instruções de laços e *if* aninhadas.
- **Teste de laços:** Os laços são amplamente utilizados na estrutura dos programas. Raros são os programas que não utilizam a estrutura de *loop* para controle de execução. As condições de teste são: pular o laço inteiramente, realizar n passagens e verificação de laços aninhados.

Engenharia de requisitos

A engenharia de requisitos de software é uma atividade que engloba a descoberta, documentação e a manutenção do conjunto de requisitos de um sistema de software. A gerência de requisitos tem por finalidade estabelecer e manter um acordo com o cliente com relação aos requisitos a serem observados no projeto de software. Em particular, a gerência de requisitos controla a evolução dos requisitos de um sistema, seja por constatação de novas necessidades ou de deficiências nos requisitos registrados, até o momento.

Requisitos

Crosby diz que qualidade é a "conformidade com os requisitos", logo vê-se a importância da Engenharia de requisitos. Define-se por requisito uma condição ou capacitação necessária a um componente do sistema (ou o próprio sistema) precisa atender, para satisfazer às necessidades do usuário, padrão, especificação ou outro.

Os requisitos podem ser classificados como *funcionais* ou *não-funcionais*, conforme explicado abaixo:

- **Funcionais:** dizem respeito à definição das funções que um sistema deve fazer. Descrevem as transformações realizadas nas entradas ou componentes de um sistema, a fim de que se produzam saídas.
- **Não-funcionais:** dizem respeito a restrições, aspectos de desempenho, interfaces com o usuário, confiabilidade, segurança, manutenibilidade, portabilidade, padrões (aspectos sociais e políticos), e outras que o sistema deve possuir. Alguns desses são traduzidos em funções (operacionalizados).

A diferença prática entre eles é que os requisitos funcionais dizem *o que* o sistema deve fazer, e os requisitos não-funcionais fixam restrições sobre *como* os requisitos funcionais devem ser implementados.

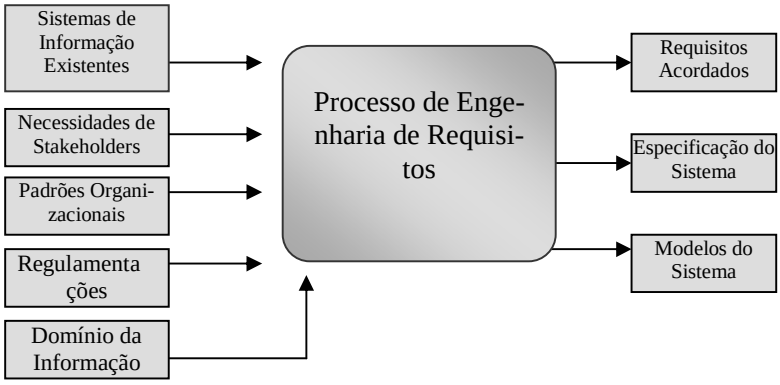
Processo da Engenharia de Requisitos

Os requisitos devem ser especificados de forma completa e consistente, para que os artefatos resultantes das próximas etapas (Projeto, Implementação e Testes) possuam a qualidade desejada. As entradas e saídas deste processo ilustram que os fatores humanos e técnicos devem ser adequadamente tratados. Confira o modelo:

O processo de engenharia de requisitos é composto basicamente pelas seguintes atividades:

- **Elicitação de requisitos:** Inicia-se junto com os usuários, analisando diversos pontos de vista.
- **Análise e negociação de requisitos:** Os requisitos são analisados em ordem de importância.
- **Documentação e Validação de requisitos:** permite o rastreamento e validar sua conformidade.
- **Gerenciamento de requisitos:** objetiva-se controlar a evolução e mudanças dos requisitos.

É importante lembrar que os requisitos podem não refletir as necessidades do cliente em relação ao sistema, pois é comum ocorrer interpretação errônea entre as partes, e requisitos incompletos ou inconsistentes.



Elicitação de requisitos

A elicitação de requisitos envolve as atividades de coleta, comunicação e validação de fatos. Esta é uma atividade de absorção das reais necessidades dos usuários do software, sendo realizada em conjunto com a modelagem do sistema. As técnicas mais utilizadas na elicitação de requisitos são as entrevistas, que podem ser realizadas com um ou mais usuários. Para que estas sejam eficazes elas devem ser planejadas com antecedência, devendo ser identificados os objetivos de cada uma, seus participantes e seu conteúdo.

O uso de perguntas de livre-contexto é uma técnica que facilita as entrevistas. Essas perguntas podem ser divididas em: perguntas iniciais, onde o foco está em definir usuários, objetivos gerais e benefícios; perguntas relacionadas ao problema e percepção do usuário sobre o problema; e meta-perguntas onde o desenvolvedor se certifica do andamento da elicitação de requisitos, se está entrevistando a pessoa certa ou se falta algo.

A técnica FAST (*Facilitated Application Specification Techniques*) é utilizada para que a elicitação de requisitos seja realizada com todos os usuários ou com grupos grandes de usuários, aumentando assim a produtividade da tarefa. O foco desta técnica está na reunião que deve ser conduzida por pessoa neutra (facilitador) com participação de usuários e desenvolvedores.

JAD (*Joint Application Design*) é um exemplo de técnica FAST, e envolve planejamento, preparação e reunião de dinâmica de grupo, estruturada para ser realizada em um período fixo (em torno de 3 a 5 dias), obtendo produtos do ciclo de desenvolvimento do software. Nesta técnica é fundamental o envolvimento gerencial.

Engenharia de Usabilidade

Usabilidade pode ser vista como a medida de qualidade das experiências dos usuários no momento em que interagem com algum produto ou sistema.

A ergonomia de Interface Homem-Computador (IHC) oferece bases teóricas e metodológicas para enfrentar o problema da usabilidade do sistema, pois esta estuda as dificuldades relacionadas entre o homem e a máquina, tendo como objetivo último o alcance de um equilíbrio ótimo entre conforto, segurança e eficiência do utilizador, face aos produtos e ferramentas.

Usabilidade é "a combinação das características: facilidade de aprender, alta velocidade na execução de tarefas, baixa taxa de erros, subjetiva satisfação e retenção do usuário com o tempo (facilidade de lembrar como executar uma tarefa).

Verifica-se que a usabilidade tem assumido um papel importante no *design* de *websites*, visto que os usuários têm que assimilar o projeto visual e navegacional do *site* antes de atingir o seu conteúdo.

Os critérios e recomendações do documento da W3C-WAI de Acessibilidade vale também para a Usabilidade, pois propicia a qualquer usuário um acesso mais rápido às informações na *Web*.

Além disso, a **ISO 9241** objetiva " Promover a saúde e a segurança de usuários de computadores e garantir que eles possam operar estes equipamentos com eficiência e conforto." Nesta norma ISO, utilizam-se as seguintes definições:

- **Usabilidade:** medida na qual um produto pode ser usado por usuários específicos, para alcançar objetivos específicos com eficácia, eficiência e satisfação, em um contexto específico de uso.
- **Eficácia:** acurácia e completude com as quais usuários alcançam objetivos específicos. Ex: Número de tarefas concluídas. Ex: Número de usuários que completaram a tarefa corretamente.
- **Eficiência:** recursos gastos em relação à acurácia e abrangência com as quais usuários atingem objetivos. Ex: Número de toques de tecla para completar a tarefa. Ex: Tempo para completar a tarefa.
- **Satisfação:** ausência do desconforto e presença de atitudes positivas para com o uso de um produto.
- **Usuário:** pessoa que interage com o produto.
- **Objetivo:** resultado pretendido.
- **Tarefa:** conjunto de ações necessárias para alcançar um objetivo.

Análise de requisitos de usabilidade

Partindo da hipótese de um sistema interativo de agendas de contato dos telefones, extensível a outros sistemas, foram identificados os seguintes requisitos e critérios de usabilidade a adotar:

- **Consistência Ação-Efeito:** consiste na disponibilidade do sistema para a entrada de dados ou acionamento de funções, e o status das mesmas, informação de ajuda e suas formas de acesso. Ex. Modelos para entrada de dados (dd/mm/yy); visualização de unidade de medidas na entrada de números; indicação de status e modo; legendas para tipos de informações; existência de pistas para tamanhos dos campos disponíveis; título para as telas; help on-line.
- **Agrupamento e distinção por localização:** relativo à organização visual da informação e à relação de um com o outro, e à localização e forma gráfica para indicar relações. Deve existir uma coerência da presença de uma função em um determinado local com suas classificações e relações por proximidade. Ex: Organização em lista hierárquica; lógica de organização: data, nome, tamanho ou tipo; legendas perto de teclas.
- **Agrupamento e distinção por formato:** consiste em verificar a forma, cor e tamanho, que ajudam a distinguir elementos e que indicam se ele pertence a uma determinada classe ou grupo. Ex: Clara distinção visual de áreas, campos e legenda que tenham diferentes funções.

- **Feedback:** é a resposta do sistema para as ações dos usuários. Rapidez e qualidade da resposta são duas características fundamentais para o feedback. Ex: Toda a entrada de dados deve ser mostrada de forma perceptível, exceto as relativas à segurança; interrupção de processamento da ação do usuário deve ser avisado e retornar ao estado anterior.
- **Leitura de Cor:** significa o emprego correto da cor para permitir um contraste adequado à leitura tanto nas teclas quanto no visor, e o emprego de cores para destacar e alertar determinados eventos no sistema.
- **Capacidade de Leitura:** é a qualidade da leitura de texto na tela. (fonte empregada, tamanho da letra, espaços, tamanho da linha, etc). Ex.: Títulos devem ser centralizados, legendas devem ser em caixa-alta, minimizar hifenização.
- **Facilitação:** critério que trata dos elementos que ajudam a reduzir a carga perceptiva e cognitiva do usuário, aumentando a eficiência do diálogo. Evitar começar números com zero na frente; abreviar códigos com muitos caracteres.
- **Ações Mínimas:** diz respeito à atenção ao número de ações para completar uma tarefa. Ex.: Minimizar o número de passos para seleção no menu; evitar entradas que precisem de pontuação; definir valores padrão para campos;
- **Densidade da Informação:** carga de trabalho provinda de um ponto de vista cognitivo ou perceptual que atende a muitos usuários e não respeita a individualidade daquele presente no momento. Ex.: Prover apenas informação útil para a transação; não encher a tela com informações desnecessárias; usuários não precisarão memorizar dados.
- **Controle do Usuário:** o usuário deve sempre ter o controle do sistema (poder interromper, cancelar e continuar). As possibilidade de controle devem ser mostradas ao usuário.
- **Proteção e Correção de Erro:** são métodos de prevenção ou redução de erros e formas de recuperação quando ocorrerem - prevenção do erro. Ex.: Avisos de possíveis erros, clareza na mensagem de erro, reversibilidade, etc.
- **Consistência:** modo pelo qual o design da interface escolhe (código, formato, procedimentos, etc.) e que estes são mantidos iguais em contextos similares e diferentes em outros contextos. Ex: uso de telas similares.
- **Compatibilidade:** o sistema deve estar de acordo com as características do usuário (memória, percepção, personalização, habilidade, idade, expectativas, etc). Ex.: A estrutura dos dados deve ser natural ao usuário.

Projeto de interfaces

Descreve como o software deverá se comunicar dentro dele mesmo (interfaces internas), com outros sistemas (interfaces externas) e com pessoas que o utilizam (interface com o usuário).

A importância do projeto de interface com o usuário se deve ao fato de que a maioria dos sistemas é desenvolvida para ser utilizada por pessoas. Nesta etapa do projeto, são definidos os formatos de janelas e relatórios, sendo a prototipagem muito utilizada. A IU capta como um usuário comandará o sistema e como o mesmo apresentará as informações a ele.

Nesta fase é necessário delinear as tarefas necessárias para obter a funcionalidade do sistema, estabelecer o perfil dos usuários, construir protótipos e avaliar resultados (dados quantitativos e qualitativos).

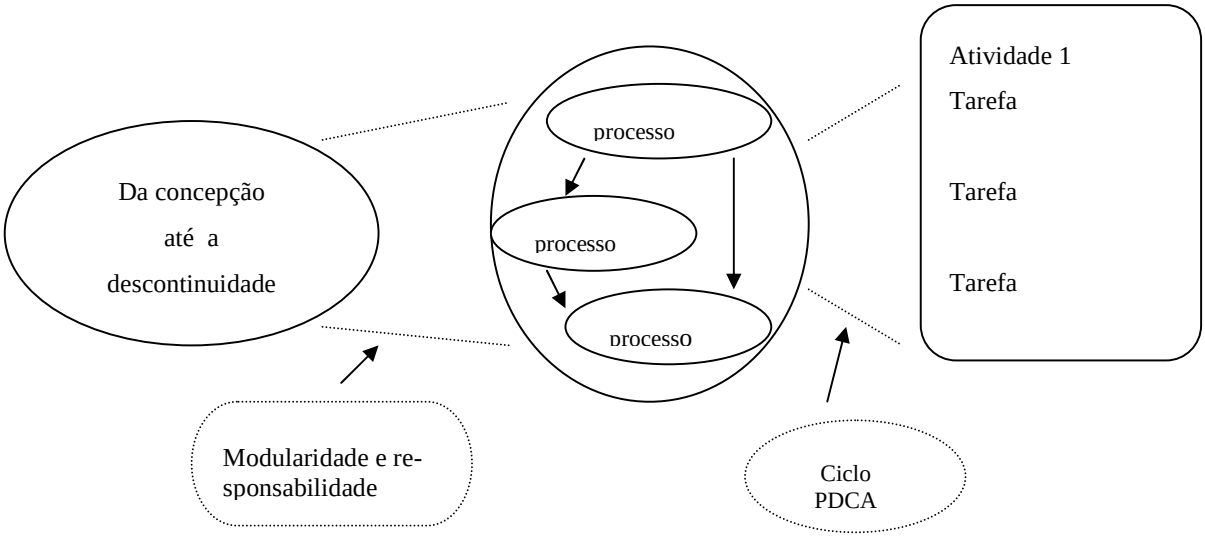
Concepção, Projeto e Implementação de Interfaces

Processos de Software

Sempre existe a necessidade de software cada vez mais complexo, pois o cliente sempre quer mais, melhor e mais rápido. Desta forma não é suficiente apenas a presença de desenvolvedores altamente treinados, é preciso um guia organizacional: um **processo**.

ISO 12207

Esta norma descreve um padrão que aborda todos os processos do ciclo de vida de um software. Ela estabelece os processos e atividades que compõem o processo de desenvolvimento de software, visando auxiliar os envolvidos na produção de software a obter um melhor entendimento das atividades que devem ser executadas



A arquitetura da ISO 12207 está representada na figura, e segue dois princípios básicos:

- **Modularidade:** os processos têm alta coesão e baixo acoplamento, de forma que a alteração de um processo impacte o mínimo na estrutura dos outros processos.
- **Responsabilidade:** cada processo na norma é de responsabilidade de uma 'parte envolvida', que pode ser uma organização ou parte dela (ou mesmo de organizações diferentes).

Os processos que envolvem o ciclo de vida do software são agrupados em três classes a seguir:

- **Fundamentais:** compreende os processos envolvidos na execução do desenvolvimento, operação e manutenção do software durante o ciclo de vida. Eles são: aquisição, fornecimento, desenvolvimento, operação e manutenção. Nestes processos ocorre a atividade de gerenciamento de requisitos.
- **De apoio:** são os processos que auxiliam o sucesso e a qualidade do projeto de software. Compreende os seguintes processos: documentação, gerência de configuração, garantia da qualidade, verificação, validação, revisão conjunta, auditoria e resolução de problemas.
- **Organizacionais:** são os processos empregados por uma organização para estabelecer e implementar uma estrutura constituída pelos processos do ciclo de vida e pelo pessoal envolvido no desenvolvimento de software. São eles: gerência, infra-estrutura, melhoria e treinamento.

Metodologias ágeis: Extreme Programming

Programação extrema (*Extreme Programming*), ou simplesmente XP, é uma metodologia ágil para equipes pequenas e médias e que irão desenvolver software com requisitos vagos e em constante mudança. Para isso, adota a estratégia de constante acompanhamento e realização de vários pequenos ajustes durante o desenvolvimento de software.

A metodologia XP promove seus cinco valores fundamentais. A partir desses valores, possui como princípios básicos: feedback rápido, assumir simplicidade, mudanças incrementais, abraçar mudanças e trabalho de qualidade:

- **Comunicação:** informa os requerimentos do sistema para os desenvolvedores. Em metodologias formais, utiliza-se documentação. Favorece o uso de *designs* simples, metáforas, colaboração de usuários e comunicação verbal.
- **Simplicidade:** incentiva começar com soluções simples e o *refactoring*. Foco na necessidade imediata (não futura).
- **Feedback:** inclui o *feedback* do sistema (testes), do cliente (funcionais) e da equipe (estimativas de tempo, etc).
- **Coragem:** conhecer o sistema, saber refatorá-lo e jogar código fora, capacidade de resolver problemas rapidamente.
- **Respeito:** buscar a qualidade, não gerando erros em outros módulos, ou atrasando o trabalho de outrem.

Dentre as variáveis de controle em projetos (custo, tempo, qualidade e escopo), há um foco explícito em escopo. Para isso, recomenda a priorização de funcionalidades que representem maior valor possível para o negócio. Desta forma, caso seja necessário a diminuição de escopo, as funcionalidades menos valiosas serão adiadas ou canceladas.

A XP desestimula controlar qualidade como variável de projeto pois o pequeno ganho de curto prazo na produtividade ao diminuir qualidade não é compensado por perdas (ou até impedimentos) em médio e longo prazo.

O ciclo de atividades no processo de desenvolvimento de software da metodologia são: **Requisitos (*listening*), Projeto, Codificação** e **Teste**. Para aplicar os valores e princípios durante o desenvolvimento, XP propõe uma série de práticas. Há uma confiança muito grande na sinergia entre elas, os pontos fracos de cada uma são superados pelos pontos fortes de outras.

- **Jogo de Planejamento (*Planning Game*):** neste jogo as atividades são estimar tarefas e identificar prioridades, sendo o cliente essencial neste processo. Com isto cliente sabe o que está acontecendo e o que vai acontecer no projeto.
- **Pequenas versões:** a liberação de pequenas versões funcionais do projeto auxilia muito no processo de aceitação por parte do cliente, que já pode testar uma parte do sistema que está comprando.
- **Metáfora:** busca facilitar a comunicação com o cliente, entendendo a realidade dele. O conceito de rápido para um cliente de um sistema jurídico é diferente para um programador experiente em controlar comunicação em sistemas de tempo real, como controle de tráfego aéreo. É preciso equalizar as palavras do cliente para o significado que ele espera dentro do projeto.
- **Projeto Simples:** simplicidade é um princípio da XP. Por projeto simples significa dizer que caso o cliente tenha pedido que na primeira versão apenas o usuário "teste" possa entrar no sistema com a senha "123" e assim terá acesso a todo o sistema, você vai fazer o código exato para que esta funcionalidade seja atendida, sem se preocupar em sistemas de autenticação e restrições de acesso.
- **Time Coeso:** a equipe de desenvolvimento é formada pelo cliente e o time de desenvolvimento.
- **Testes de Aceitação (*Customer Tests*):** são testes construídos pelo cliente e conjunto de analistas e testadores, para aceitar um determinado requisito do sistema.
- **Ritmo Sustentável:** trabalhar com qualidade, em um ritmo de trabalho saudável (40 horas/semana, 8 horas/dia), sem horas extras. Horas extras são permitidas quando forem trazer produtividade para a execução do projeto.
- **Reuniões em pé (*Stand-up Meeting*):** reuniões em pé para não se perder o foco nos assuntos para deixar as reuniões rápidas, apenas abordando tarefas realizadas e tarefas a realizar pela equipe.
- **Posse Coletiva (*Collective Ownership*):** o código fonte não tem dono e ninguém precisa ter permissão concedida para poder modificar o mesmo. O objetivo com isto é fazer a equipe conhecer todas as partes do sistema.
- **Programação em pares (*Pair Programming*):** é a programação em duplas em um único computador. Geralmente a dupla é criada com alguém sendo iniciante na linguagem e a outra pessoa funcionando como um instrutor. O novato é que fica a frente fazendo a codificação, e o instrutor acompanha ajudando a desenvolver suas habilidades, dessa forma o programa sempre é revisado visto por duas pessoas, evitando e diminuindo assim a possibilidade de erros (*bugs*). E com isto, se busca sempre a evolução da equipe, melhorando a qualidade do código fonte gerado.
- **Padrões de Codificação:** a equipe de desenvolvimento precisa estabelecer regras para codificar e todos devem seguir estas regras. Desta forma parece que todos os códigos fontes foram editados pela mesma pessoa, mesmo a equipe possuindo 10 ou 1000 componentes.
- **Desenvolvimento dirigido por testes:** primeiro crie os testes unitários e depois crie o código para que os testes funcionem. Esta abordagem é complexa no início, pois vai contra o processo de desenvolvimento de muitos anos. Só que os testes unitários são essenciais para que a qualidade do projeto seja mantida.
- **Refatoração:** sempre que puder melhorar uma programação, melhore. O objetivo é sempre que possível refinar o código construído, para que ele fique melhor, mais eficiente e mais simples.
- **Integração Contínua:** sempre que realizar uma nova funcionalidade, nunca esperar uma semana para integrar na versão atual do sistema. Isto só aumenta a possibilidade de conflitos e a possibilidade de erros no código fonte. Integrar de forma contínua permite saber o status real da programação.

Metodologias ágeis: FDD

FDD (*Feature Driven Development*) em um projeto para um grande banco em Singapura, a partir de técnicas de gerenciamento de projetos e de modelagem orientada a objetos.

Utilizando as técnicas, estratégias e padrões, nascia uma metodologia que equilibra as vantagens das metodologias rigorosas (planejamento e modelagem, por ex.) e das metodologias ágeis (ciclos curtos, orientação ao cliente, ênfase em programação).

Desde então, a metodologia tem sido testada e os resultados têm comprovado sua eficácia e eficiência, servindo inclusive para realizar pequenas melhorias ao plano básico. A melhoria do próprio processo é uma de suas premissas naturais.

Por suas características inerentes pode ser utilizada para equipes de tamanhos variados, tendo sido aplicada em projetos com até 250 integrantes. Sua estrutura básica compreende 5 processos, sendo que a descrição de cada um é bem curta. É de fácil assimilação e prática, fornecendo resultados rápidos e satisfatórios mesmo com equipes inexperientes.

- Desenvolver um Modelo Geral (*Develop an Overall Model*)
- Construir uma Lista de Funcionalidades (*Build a Features List*)
- Planejar por Funcionalidade (*Plan By Feature*)
- Projetar por Funcionalidade (*Design By Feature*)
- Codificar por Funcionalidade (*Build By Feature*)

Adotando um ciclo iterativo de desenvolvimento, admite mudanças mesmo em meio à construção do produto, embora o processo de modelagem inicial já forneça um fundamento básico que pode minimizar a necessidade e/ou o impacto de mudanças.

Uma das maiores atrações da FDD são os relatórios de acompanhamento. Com sua origem em experiências de gerenciamento de projetos, certamente essa ênfase em "ver" o que acontece durante o desenvolvimento é uma característica típica. E é justamente essa visibilidade que permite tomar decisões relativas aos riscos, qualidade, mudanças nos requisitos, utilidade para o usuário do produto e várias outras variáveis do projeto, em tempo hábil para não prejudicá-lo.

Com esses relatórios é possível coletar métricas para geração de históricos, que permitirão melhorar as estimativas em próximos projetos, aumentando a precisão e segurança das decisões sobre investimentos, mudanças de prioridades, atendimento de prazos e custos, etc.

Os desenvolvedores aceitam prontamente e incorporam a metodologia de forma rápida, pois:

- não causa interrupções burocráticas para a coleta dos dados sobre o progresso das atividades;
- ajuda-os a se organizarem;
- possibilita a entrega de produtos úteis e com qualidade, repetidamente;
- fortalece o sentimento de equipe, enquanto valoriza o talento individual;
- facilita a difusão do conhecimento do negócio pela organização, diminuindo as barreiras na comunicação.

As metodologias XP e FDD possuem muitas semelhanças e diferenças. Um bom exemplo é que XP utiliza *programação em pares* para remover defeitos (erros) e aumentar a qualidade, enquanto FDD utiliza inspeções.

MDA – Model Driven Architecture

A MDA é uma metodologia de desenvolvimento de *software* proposta pela OMG, e tem por objetivo separar o que o sistema necessita fazer de como o sistema pode fazer, possibilitando que projetos sejam desenvolvidos independentemente de plataforma e que, a partir deste projeto, selecione-se a plataforma específica e se gere um projeto especificamente para ela.

Além disso, a MDA busca prover modelos independentes de plataforma para diversos domínios de aplicações, compreendendo banco de dados, sistemas embarcados, sistemas empresariais, etc. O desenvolvimento em três modelos ou visões:

- **Modelo independente de computação (CIM - *Computation Independent Model*):** é a representação de mais alto nível do sistema, representando as possibilidades e funcionalidades deste que não dependem de computação.
- **Modelo independente de plataforma (PIM - *Plataform Independent Model*):** especifica as funcionalidades do sistema, explicitando a lógica para funcionamento do sistema. O PIM é o projeto do sistema e pode ser transformado em um modelo específico de uma plataforma, o PSM.
- **Modelo específico de plataforma (PSM - *Plataform Specific Model*):** contém as funcionalidades do sistema aplicadas na plataforma.

Os objetivos da MDA são: portabilidade, interoperabilidade e reutilização de elementos de projeto e de código através da separação dos conceitos. Para minimizar custos de desenvolvimento de software, utilizando os padrões adotados pela OMG, em especial os padrões UML, CORBA e OCL, a MDA busca separar as funcionalidades de um sistema da forma como ele deve ser implementado, concentrando-se em um desenvolvimento de software que satisfaça os seus objetivos.

A MDA objetiva prover facilidades para o desenvolvimento de software, como:

- Especificação de um sistema independentemente da plataforma que o suporta;
- Especificação de plataformas;

- A escolha de uma plataforma particular para implementar o sistema;
- Transformação da especificação de um sistema independente de plataforma em uma plataforma particular.

A MDA é uma filosofia de desenvolvimento de software composta por um conjunto de modelos de sistema, utilizados para suprir os seus objetivos. Um modelo de sistema entende-se por uma descrição ou especificação deste sistema, sempre apresentado como uma combinação de desenhos e textos, estes últimos que podem ser expressos em linguagem natural ou linguagem de modelagem, como UML. Logo a MDA fornece modelos para especificação das várias partes constituintes de um sistema. Estes modelos são expressos em diagramas já conhecidos e utilizados pelos projetistas e programadores de software, os diagramas da notação UML. Além disso, a MDA provê aos diagramas uma força maior do que os possuem os diagramas UML convencionais, de forma que os modelos podem ser usados para direcionar o curso do projeto, construção, emprego, operação, manutenção e modificação.

Esta filosofia de desenvolvimento de software busca sustentar na Engenharia de Software tradicional a capacidade de abstração do conceito de implementação de um sistema das funcionalidades deste sistema. Para isso, propõem modelar todas as características do sistema que são independentes de plataforma e, posteriormente, gerar uma implementação do sistema para uma arquitetura específica. Ela tem por objetivo separar o que o sistema necessita fazer (requisitos funcionais) de como o sistema pode fazer (requisitos não funcionais), permitindo uma transição de um modelo independente de arquitetura (PIM) para uma implementação dependente de arquitetura (PSM), sem que sejam necessárias muitas alterações no projeto do sistema neste processo. Alguns benefícios com a adoção da MDA são: rapidez (menos tempo para que um software seja entregue), minimização no impacto ocasionado pelas mudanças de requisitos na escala do desenvolvimento, maior reuso de componentes de software e consistência de implementação (poucos erros para corrigir e pouca propagação de erros), flexibilidade arquitetural e independência de plataforma.

Este é o fluxo utilizado pela OMG para representar a MDA:

- Um modelo que especifica o sistema dependente de computação, mas independente de plataforma;
- Um modelo que especifica possibilidades que são independentes de computação;
- Um mapeamento de um modelo independente de plataforma para uma plataforma específica e;
- Como resultado da transformação deste último modelo gerado, um modelo para a plataforma específica.

A MDA preocupa-se também com geração automática de código. Isto quer dizer que, ao selecionar uma arquitetura, como por exemplo J2EE ou .NET, espera-se que o modelo independente de plataforma PIM, estando convertido em um modelo específico de plataforma PSM, possibilite a geração automática de código para a arquitetura escolhida. O desenvolvimento de um sistema em MDA, utilizando o caminho mais curto para gerar código a partir de um modelo UML, procede de um CIM para um PIM, de um PIM para um PSM e de um PSM para código. Existem atualmente muitas ferramentas de desenvolvimento de projetos de software que convertem modelos UML em código e que aplicam alguns dos conceitos da MDA.

MDD – Model Driven Development

As pesquisas relacionadas com geração automática de código concentram-se hoje desenvolvimento de software dirigido ao modelo MDD, ou desenvolvimento dirigido a modelo. A promessa do MDD é que compiladores de uma plataforma específica produzirão implementações para múltiplas plataformas a partir de uma única especificação.

Além disso, eles fazem uma retrospectiva das técnicas utilizadas de alguns anos pra cá para obter um processo de desenvolvimento de software rápido, em que para transformação de um modelo UML em código executável utiliza-se das ferramentas CASE de uma linguagem de programação específica, mas salienta que estas ferramentas pouco auxiliaram, até o momento, no desenvolvimento de um bom código e pouco possibilitaram o aceleramento do desenvolvimento de software

RUP

É necessário um processo que integre as muitas facetas do desenvolvimento. Uma solução é o UP (*Unified Process*). O UP é um *framework* genérico de um processo de desenvolvimento, baseado em componentes e utiliza a definição da UML . Ele é dirigido pelos casos de uso, centrado na arquitetura, iterativo e incremental.

O RUP (*Rational Unified Process*) denota três conceitos bem diferentes:

- O RUP é um **Processo de Engenharia de Software**. Fornece uma abordagem disciplinada para distribuir tarefas e responsabilidades em uma organização voltado para o desenvolvimento. O seu objetivo é garantir a produção de *softwares* de alta qualidade que atendem às necessidades do cliente, no tempo e com o *budget* previsto.

- O RUP é um **Produto de Processo** desenvolvido e mantido pela Rational Software. A equipe de desenvolvimento do RUP trabalha continuamente e de forma dedicada com clientes, parceiros e com os grupos de desenvolvedores de Produtos da Rational para assegurar que este processo e padrão seja atualizado e aprimorado constantimente refletindo as experiências mais recentes comprovando este guia/padrão.
- O RUP é uma **Abordagem de Desenvolvimento de Software**, que é iterativa, centrada a arquitetura, e dirigida a casos de uso. A maior parte das informações sobre essa abordagem pode ser encontrado no próprio Produto, que contém detalhes, exemplos, *templates* e todo o ciclo de vida

As atividades do RUP criam e mantém modelos. Ao invés de focar a produção de grandes quantidades de documentos em papel, ele enfatiza o desenvolvimento e manutenção de modelos de forma semântica com ricas representações do sistema / software em desenvolvimento.

Não existe uma maneira exata de aplicar o RUP, pois ele pode ser aplicado de várias formas e será diferente em cada projeto e organização. Porém, existem alguns princípios que podem caracterizar e diferenciar o RUP de outros métodos iterativos:

- Atacar os riscos cedo e continuamente;
- Certificar-se de entregar algo de valor ao cliente;
- Focar no software executável;
- Acomodar mudanças cedo;
- Liberar um executável da arquitetura cedo;
- Construir o sistema com componentes;
- Trabalhar junto como um time;
- Fazer da qualidade um estilo de vida, não algo para depois.

Algumas vantagens ao utilizar o Processo Unificado são:

- Fornece uma base sólida para a construção do software
- Melhor compreensão do sistema e organização do desenvolvimento
- Prescreve um refinamento sucessivo à arquitetura
- É um processo iterativo e incremental, logo antecipa riscos e há um aprendizado facilitado.

Melhores Práticas do RUP

São seis as melhores práticas de um processo RUP:

- **Desenvolver *software* iterativamente:** dado o tempo para desenvolver um *software* sofisticado, não é possível definir o problema e construir a solução em um único passo. Iteração permite refinar o projeto, e priorizar as atividades de alto risco, fazendo com que cada iteração termine idealmente com uma nova *release*.
- **Gestão de requisitos:** uma documentação apropriada é essencial para qualquer grande projeto; note-se que o RUP descreve como documentar a funcionalidade, restrições de sistema, restrições de projeto e requisitos de negócio. Os casos de uso (também conhecidos como Use Cases) e os cenários são exemplos de artefatos dependentes do processo, que têm vindo a ser considerados muito mais eficazes na captura de requisitos funcionais.
- **Uso de arquitetura baseada em componentes:** a arquitetura baseada em componentes cria um sistema que pode ser facilmente extensível, promovendo a reutilização de software e um entendimento intuitivo. Um componente normalmente se relaciona com um objeto na programação orientada a objetos. O RUP oferece uma forma sistemática para construir este tipo de sistema, focando-se em produzir uma arquitetura executável nas fases iniciais do projeto, ou seja, antes de comprometer recursos em larga escala. Estes componentes são normalmente incluídos em infraestruturas existentes como o CORBA e o COM (Modelo de Componentes de Objetos).
- **Uso de software de modelos visuais:** ao abstrair a programação do seu código e representá-la utilizando blocos de construção gráfica, o RUP consegue uma maneira efetiva de se ter uma visão geral de uma solução. O uso de modelos visuais também pode permitir que indivíduos de perfil menos técnico (como clientes) tenham um melhor entendimento de um dado problema, e assim se envolvam mais no projeto como um todo. A linguagem de modelação UML tornou-se um padrão industrial para representar projetos, e é amplamente utilizada pelo RUP.

- **Verificação da qualidade do software:** não assegurar a qualidade do software é a falha mais comum em todos os projetos de software. Normalmente, pensa-se em qualidade de software após o término dos projetos, ou a qualidade é responsabilidade por uma equipe diferente da equipe de desenvolvimento. O RUP intenciona assistir no controle do planejamento da qualidade, verificando-a na construção de todo o processo e envolvendo todos os membros da equipe de desenvolvimento.
- **Gestão e Controle de Mudanças do Software:** em todos os projetos de software a mudança é inevitável. O RUP define métodos para controlar e monitorizar mudanças. Como uma mudança pode afetar aplicações de formas imprevisíveis, o controle de mudanças é essencial para o sucesso de um projeto. O RUP também define áreas de trabalho seguras, garantindo a um programador que as mudanças efetuadas noutro sistema não irão afetar o seu sistema.

O RUP possui cinco elementos principais: **papéis**, **atividades**, **artefatos**, **fluxos de trabalho** e **disciplinas**.

Elementos Estáticos do RUP

Um **papel** (*worker*l) define o comportamento e as responsabilidades de um determinado indivíduo ou grupo de indivíduos trabalhando como uma equipe. Papéis não são indivíduos e nem títulos de trabalho. Um indivíduo pode assumir vários papéis.

- **Analista de sistema:** o indivíduo que assume este papel coordena a obtenção dos requisitos e a modelagem dos casos de uso identificando funcionalidades do sistema e estabelecendo limites do sistema;
- **Projetista:** esse indivíduo define responsabilidades, operações, atributos, relacionamentos de uma ou mais classes e determina como elas devem ser ajustadas para serem implementadas no ambiente;
- **Projetista de testes:** responsável pelo planejamento, projeto, implantação e avaliação de testes, incluindo a geração de plano e modelo de teste, implementando procedimentos de testes e avaliando a abrangência dos testes, resultados e a efetividade.

Uma **atividade** é uma unidade de trabalho que um indivíduo executa quando está exercendo um determinado papel e produz um resultado importante para o contexto do projeto. Cada atividade pode ser dividida em passos. São exemplos de atividades:

- Planejar uma iteração: realizada pelo papel gerente de projeto;
- Encontrar casos de uso e atores: realizada pelo papel analista de sistemas;
- Rever o projeto: realizada pelo papel revisor de projeto;
- Executar um teste de performance: realizado pelo papel testador de performance.

Um **artefato** é um pedaço de informação que é produzido, modificado ou utilizado em um processo. Os artefatos são os produtos de um projeto. São as coisas produzidas durante o desenvolvimento do projeto. Artefatos são utilizados como entradas de atividades e são produzidos como saída. Os artefatos podem ter várias formas:

- Um modelo, como um modelo de caso de uso, um modelo de projeto;
- Um elemento de um modelo, como uma classe, um caso de uso, um sub-sistema ;
- Um documento, como um caso de negócio, glossário, visão;
- Código fonte e Executáveis.

A enumeração de atividades, papéis e artefatos não constituem um processo. É necessário saber a seqüência do desenvolvimento das atividades para que se produzam artefatos de valor para o projeto. Isso é definido pelo fluxo de trabalho.

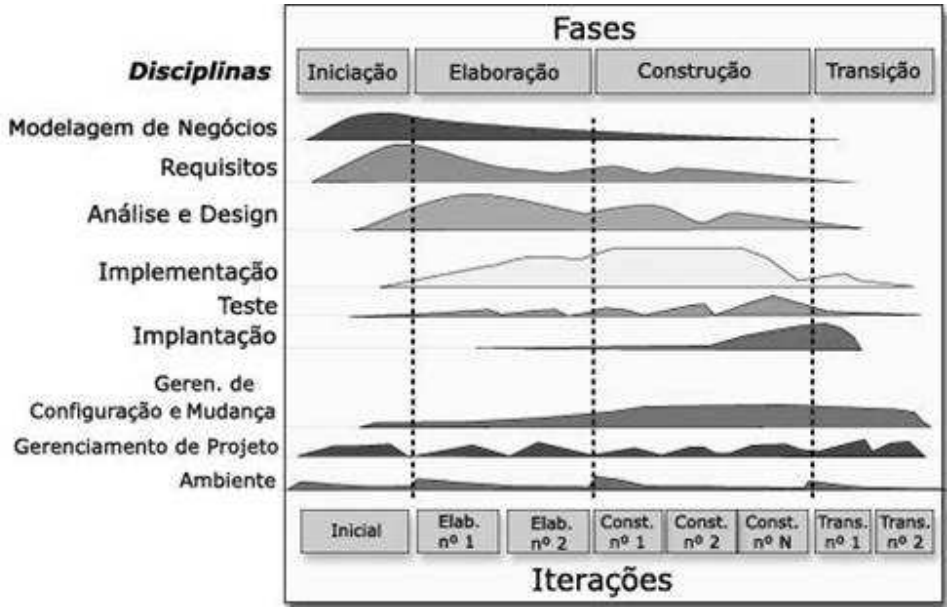
Um **fluxo de trabalho** (*workflow*) é uma seqüência de atividades que são executadas para a produção de um resultado valioso para o projeto. Fluxos de trabalho podem ser representados por diagramas de seqüência, diagramas de colaboração e diagramas de atividades da linguagem UML. O RUP utiliza três tipos de fluxos de trabalho:

- Fluxos de trabalho principais (*core workflow*), associados com cada disciplina;
- Fluxos de trabalho de detalhe (*detail workflow*), para detalhar cada fluxo de trabalho principal;
- Planos de iteração, que mostram como a iteração deverá ser executada.

Disciplinas do RUP

Uma disciplina é uma coleção de atividades relacionadas que fazem parte de um contexto comum em um projeto. As disciplinas proporcionam um melhor entendimento do projeto sob o ponto de vista tradicional de um processo cascata. A separação das atividades em disciplinas facilita a compreensão das atividades, porém dificulta mais o planejamento das atividades.

O RUP possui nove disciplinas, divididas em disciplinas do processo e de suporte. As **Disciplinas de Processo** são a modelagem de negócios, requisitos, análise e projeto, implementação, teste e implantação. As **Disciplinas de Suporte** são o gerenciamento de configuração e mudanças, o gerenciamento de projeto e o ambiente (*environment*).



Fases do RUP

O Processo Unificado repete vários ciclos até a aposentadoria do sistema cada ciclo gera um produto liberado para uso. Cada ciclo possui 4 fases: **Concepção**, **Elaboração**, **Construção** e **Transição**.

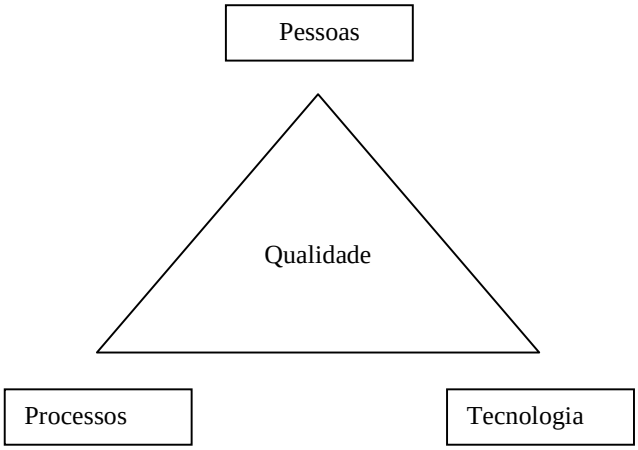
Cada fase é então subdividida em iterações. Um conjunto de artefatos (release) é gerado a cada iteração. Um milestone é gerado a cada fase. As fases indicam a ênfase que é dada no projeto em um dado instante. Para capturar a dimensão do tempo de um projeto, o RUP divide o projeto em quatro fases diferentes:

- **Concepção:** ênfase no escopo do sistema
- **Elaboração:** ênfase na arquitetura
- **Construção:** ênfase no desenvolvimento
- **Transição:** ênfase na implantação

Modelos de melhoria de qualidade de processo e produto

O grande desafio da Engenharia de Software é desenvolver softwares de qualidade assegurada, com elevada produtividade, dentro do prazo estabelecido, e sem necessitar de mais recursos que os alocados.

Um esforço surgiu com o trabalho pioneiro de *Yourdon* para elaborar modelos para desenho e implementação de sistemas. Surgiu o conceito do Triângulo da Qualidade (figura), que depende de três fatores (pessoas, processos e tecnologia). Outros modelos mais elaborados surgiram para elevar a qualidade dos softwares.



A ISO (*International Organization for Standardization*) é a organização internacional para a padronização, estabelecida em 1947. O seu principal objetivo é o desenvolvimento de padrões mundiais, facilitando o intercâmbio mundial de produtos e serviços, e cooperação intelectual, científica, econômica e técnica.

No Brasil a ABNT (Associação Brasileira de Normas Técnicas) é a entidade responsável pela normalização técnica no país, além de representar o Brasil nas entidades de normalização internacional, como a ISO.

CMM/CMMI

Em 1991, o SEI (*Software Engineering Institute*) publicou o CMM (*Capability Maturity Model*) para software, um framework de maturidade que estabelece fundamentos de engenharia de software e de gerenciamento de projeto para controle quantitativo do processo de software.

O CMM (*Capability Maturity Model*, ou Modelo de Maturidade da Capacitação) é uma estrutura conceitual que propõe um caminho evolutivo de melhoria, para que organizações venham a praticar Engenharia de Software de forma sistemática. Este caminho evolutivo é definido por cinco níveis de maturidade. As organizações passam a evoluir nestes níveis na medida em que definem, implementam, implantam, medem, controlam e melhoram seus processos de software, saindo de um processo de desenvolvimento pouco sistemático e alcançando um processo maduro e otimizado;

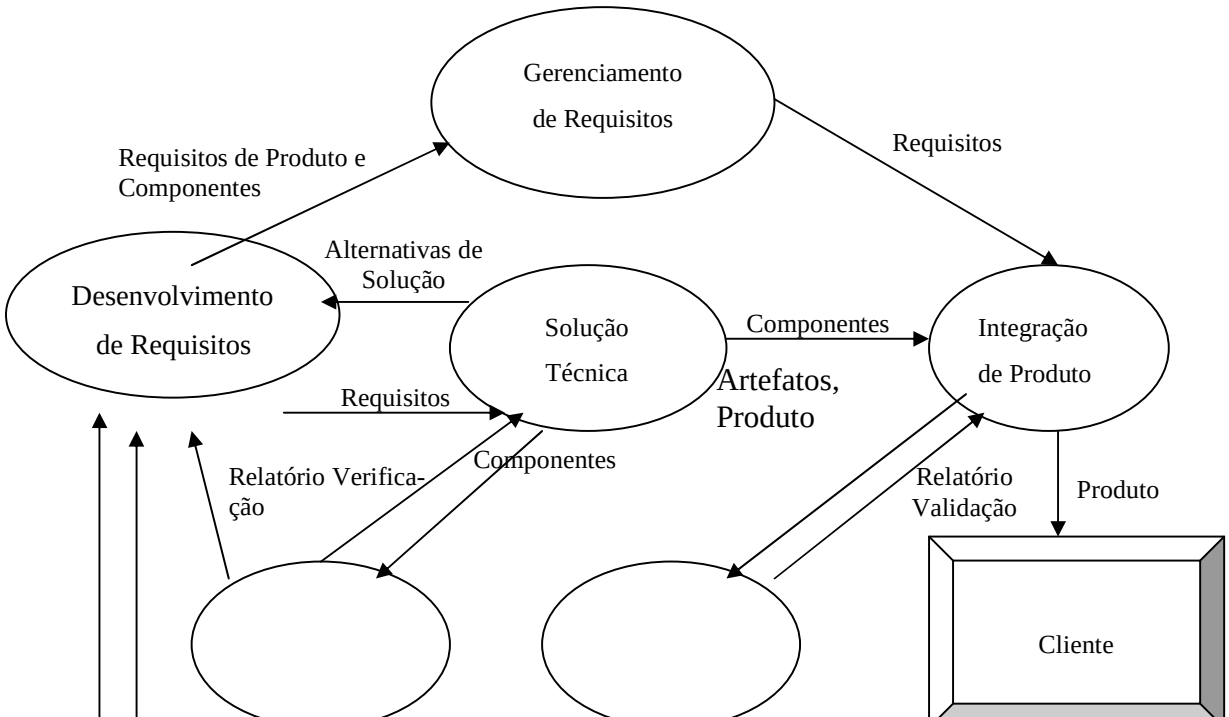
O CMM estabelece práticas de Engenharia de Software relacionadas com aspectos gerenciais, organizacionais e técnicos. Quando estas práticas são seguidas rotineiramente, as organizações se capacitam a atingir metas estabelecidas de controle de custo, cronograma e produtividade. O CMM pode ser utilizado:

- Como um guia para ajudar gerentes e técnicos a definirem e melhorarem os processos de software da organização, adotando um processo sistemático e efetivo de desenvolvimento de sistemas
- Para identificar os pontos fracos e os de melhoria nos processos de desenvolvimento e de manutenção de software praticados na organização, viabilizando a tomada de ação para aprimorar os processos
- Para avaliar o risco de contratar um projeto de software de uma outra organização, bem como para monitorar os contratos com esta organização

Segundo o CMM, a principal causa dos problemas de qualidade e produtividade é a falta de um processo de desenvolvimento de software claramente definido e efetivo. O CMM propõe um caminho gradual que leva as organizações a se aprimorarem continuamente em busca da solução dos seus problemas. Este caminho se dá em cinco níveis de maturidade:

- **1 – Inicial:** a improvisação rege o processo de desenvolvimento.
- **2 – Repetitivo:** focado nos aspectos gerenciais, como controles básicos mais imediatos.
- **3 – Definido:** ênfase na descrição do processo padrão da organização.
- **4 – Gerenciado:** técnicas para controlar os processos de desenvolvimento e a qualidade final.
- **5 – Otimizado:** apresenta uma melhoria contínua nos processos, através de análises do nível anterior.

Já o CMMI foi desenvolvido para integrar três disciplinas: engenharia de software, engenharia de sistemas e desenvolvimento integrado de produto e processo, utilizando como fonte primária os bem sucedidos padrões do modelo CMM. Os seus modelos podem ser em estágio (*staged*) ou contínuos (*continuous*).



Abaixo, estão relacionadas as Áreas-chave de Processo para o sistema de qualidade, como descritos no item 3 do Technical Report CMU/SEI-93 - Capability Maturity Model for Software:

Nível 1 - Inicial

Não tem Áreas-chave de Processo.

Nível 2 - Repetível

Mais do que qualquer outro nível de maturidade, o nível repetível têm interesse especial, pois além do fato da grande maioria das empresas estar no nível 1 (Inicial), o caminho para o nível 2 é talvez o mais difícil de ser percorrido, pois trata-se de colocar o trem em movimento, enquanto a passagem para os níveis seguintes pode se beneficiar do embalo gerado pelos esforços de melhoria para se alcançar o nível 2.

Iniciando o Processo de Melhoria: a partir de uma avaliação formal ou informal, em que se encontram os principais pontos fracos no processo de desenvolvimento, é necessário fazer com que todos dentro da empresa comprem a idéia da melhoria, e isto inclui a alta administração (diretoria), profissionais de informática (gerentes, analistas programadores, AD, DBA, produção, suporte etc) e usuários (gerência e pessoal operacional). As negociações de planejamento são o teste crítico do time gerencial. Este time deve tratar o plano inicial como um ponto de partida e, quando o prazo ou o custo tiverem de ser reduzidos, o escopo do trabalho deve ser igualmente cortado

O foco do nível 2: é o estabelecimento de procedimento básicos de Gerenciamento de Projetos. A falta de planejamento é a primeira coisa a ser atacada, sem a qual qualquer outro esforço de melhoria é baldado. Uma equipe que desenvolve seu trabalho aleatoriamente não poderá se beneficiar de técnicas de inspeção, metodologias de desenvolvimento ou métricas.

As diversas atividades de melhoria de software devem ser conduzidas por um grupo especialmente constituído para isto, normalmente chamado de SEPG (*Software Engineering Process Group* - Grupo de Processos de Engenharia de Software). A existência de um grupo especificamente encarregado destas tarefas é fundamental para garantir os projetos de melhoria. Obviamente, não são os desenvolvedores já pressionados que disporão de tempo para estas atividades. As KPAs do nível 2 são:

- **Gerenciamento de Requisitos:** o seu propósito é estabelecer um entendimento comum entre o usuário e a equipe de projeto a respeito dos requerimentos que serão atendidos pelo projeto. Além disso, são estabelecidos procedimentos para a mudança dos requisitos, garantindo que os planos, cronogramas e recursos alocados estejam sempre coerentes com os requisitos. Estes requisitos devem também ser revisados por diversas áreas envolvidas (para verificar sua viabilidade) e ter sua qualidade controlada. Observe-se que a documentação dos requisitos e sua revisão devem ser feitas antes da elaboração do plano do projeto. Em seguida, a organização precisa instituir procedimentos para a alteração dos requisitos. A existência de um procedimento formal tem dois benefícios: garante que os planos serão revistos para acomodar as alterações (isto é, renegociação de prazos e recursos) e funciona como um freio para alterações desnecessárias ou postergáveis (normalmente, a maioria), instituindo uma disciplina de controle de prioridades.
- **Planejamento de Projetos:** o seu propósito é garantir que, a cada projeto, sejam elaborados planos razoáveis para o desenvolvimento. Isto envolve a realização de estimativas e o estabelecimento de compromissos. Também se incluem aqui atividades de gerenciamento de riscos e alocação de recursos.A empresa deve instituir políticas e procedimentos padronizados para o planejamento de projetos, incluindo a necessidade de envolver todas as áreas interessadas (técnicas e usuárias) no processo de planejamento. Esta KPA também pressupõe que a empresa tenha definido um ciclo de vida de sistemas para ser usado no planejamento.
- **Acompanhamento e Supervisão de Projetos:** planejar o projeto não basta. É preciso garantir, ao longo de sua execução, que os planos estão sendo cumpridos. O propósito desta KPA é dar visibilidade sobre o progresso real do projeto em relação ao planejado, permitindo a tomada de ações corretivas quando a realidade estiver se afastando muito

do que foi inicialmente planejado. A organização deve instituir procedimentos para garantir que o progresso do projeto é continuamente comparado com o planejado, em termos de prazos, custos e qualidade. Além disso, os procedimentos devem estabelecer gatilhos a serem disparados quando houver muita distância, indicando as ações corretivas a serem tomadas em cada caso, incluindo alterações nos compromissos assumidos. A evidência, por exemplo, de que um prazo não será cumprido deve levar à renegociação de prazo, recursos, escopo ou nível de qualidade previamente acordados. Esta KPA, naturalmente, inclui o acompanhamento dos fatores de risco identificados nas atividades de gerenciamento de riscos incluídas na KPA de Planejamento de Projetos. Outro ponto importante coberto por esta KPA é o estabelecimento da obrigatoriedade da existência de pontos de revisão formal ao longo do projeto. Isto significa que, em determinados momentos do desenvolvimento, os produtos gerados até então são revisados formalmente pelas áreas envolvidas, como forma de garantia que o projeto está andando conforme deveria. Quando ocorrem alterações nos planos por conta de ações corretivas tomadas, é fundamental que os planos originais sejam mantidos. Isto permitirá futuramente que os erros de estimativas sejam conhecidos e, portanto, reduzidos. Sabendo-se onde se errou, pela comparação entre planejado e real, futuros projetos poderão ser estimados com mais precisão.

- **Gerenciamento de Subcontratação:** cada vez mais são usados recursos externos (*outsourcing*) para colaborar no desenvolvimento do software. O propósito desta KPA é garantir a seleção e gerenciamento eficaz destes recursos terceirizados. A organização deve estabelecer procedimentos para garantir que a seleção de subcontratados se faz de forma apropriada, que as atividades a serem desenvolvidas pelos terceiros estejam bem definidas e planejadas, que o contrato entre a empresa e os terceiros é adequado e que a comunicação entre empresa e terceiros é adequada. Além disso, devem ser estabelecidos procedimentos especificando como se fará a comunicação (por exemplo, documentos de trabalho a serem trocados) e, naturalmente, procedimentos para revisão e controle de qualidade dos produtos gerados pelos terceiros.

- **Garantia de Qualidade de Software:** o propósito desta KPA é prover visibilidade sobre a qualidade tanto dos processos utilizados pela equipe de projeto quanto dos produtos gerados. Isto inclui revisões formais e informais dos produtos e auditorias dos processos. Tipicamente, em empresas com recursos suficientes, existe um grupo especialmente dedicado a estas tarefas. É importante observar que esta KPA diz respeito tanto a produtos quanto a processos. Além disso, as revisões de qualidade se dão em todas as fases do desenvolvimento. Quanto à revisão dos processos, deve-se dizer que não basta revisar produtos. Na maioria das vezes, produtos sem qualidade são resultado de processos sem qualidade. Não basta descobrir que um programa tem um monte de *bugs*. É necessário descobrir por quê estes *bugs* existem. Geralmente, a razão será algo como o não uso dos padrões definidos de programação, documentação de programação insuficiente, comunicação inadequada entre analistas e programadores etc. A empresa deve instituir um grupo de garantia de qualidade (ou ao menos um comitê) e procedimentos que garantam que as atividades necessárias de garantia de qualidade são planejadas, executadas e revisadas. Estes procedimentos devem incluir medidas objetivas de verificação de qualidade, e também padrões de qualidade que possam ser usados como medida de comparação com os produtos e processos efetivamente utilizados. A cada projeto, portanto, são desenvolvidos planos de garantia de qualidade, dentro do planejamento geral do projeto. Este planos incluem pontos e procedimentos de verificação, tais como revisões de documentos, inspeções de código e, naturalmente, testes.

- **Gerenciamento de Configuração:** o propósito desta KPA é identificar, organizar, controlar modificações de software que está sendo construído por uma equipe, maximizar a produtividade e minimizar erros e defeitos. É aplicada a todo o processo de Engenharia de Software, do início do projeto até quando produto é retirado de operação, garantindo que as alterações estejam sendo corretamente implementadas e relatadas a outras pessoas que possam ter interesse. Procedimentos de Controle de Mudanças são estabelecidos, garantindo que alterações em produtos (documentos, dados de teste, programas) são feitas de forma consistente e sob o conhecimento e aprovação de todos os envolvidos. O uso de software de Controle de Configuração (*Librarians*) pode facilitar a execução destas atividades. Também é necessária a criação de um grupo (ou comitê) responsável pelas atividades de gerenciamento de configuração. Toda manutenção passa a ser considerada como a implantação de uma nova versão ou release do software.

Nível 3 - Definido

Focalização dos Processos da Organização

Definição dos Processos da Organização

Programa de Treinamento

Gerenciamento Integrado de Software

Engenharia de Produto de Software

Coordenação Inter Grupos

Revisões Detalhadas para Prevenção de Defeitos

Nível 4 - Gerenciado

Gerenciamento Quantitativo dos Processos

Gerenciamento da Qualidade de Software

Nível 5 - Otimizado

Prevenção de Falhas

Gerenciamento das Mudanças nos Processos

Gerenciamento das Mudanças Tecnológicas

Características Comuns

Verificação da Implantação (revisões e auditorias)

Mensuração e Análise (medidas e avaliações)

Compromisso para Realizar (políticas e responsabilidades)

Capacidade para Realizar (recursos, estruturas e treinamento)

Ações e Atividades Realizadas (planejamento, procedimentos e ações corretivas)

ISO 9126

A ISO/IEC 9126 trata das características da qualidade de software e métricas, e divide-se em quatro partes:

- **ISO/IEC 9126-1:** Modelo de qualidade.
- **ISO/IEC 9126-2:** Métricas externas.
- **ISO/IEC 9126-3:** Métricas internas.
- **ISO/IEC 9126-4:** Métricas de qualidade em uso.

De acordo com a ISO 9126-1, a mais antiga das normas de qualidade de software, alguns conceitos são:

- **Qualidade interna:** refere-se principalmente ao ambiente de programação, e é a totalidade dos atributos de um produto que satisfazem as necessidades quando utilizado em condições especificadas.
- **Qualidade externa:** refere-se principalmente à qualidade de entrega do produto, e constitui o quanto um produto satisfaz as necessidades quando utilizado em condições especificadas.
- **Qualidade em uso:** é a visão do usuário do ambiente de qualidade que o software está inserido.

De acordo com esta norma, um produto possui características e sub-características que medem a qualidade:

- **Funcionalidade:** O produto de software satisfaz as necessidades? É o conjunto de atributos que evidenciam se as funções satisfazem as necessidades.

  Adequação (se as funções são apropriadas)

  Acurácia (evidencia a geração de resultados corretos)

  Interoperabilidade (capacidade de interagir com outros sistemas)

  Conformidade (esteja de acordo com as normas e regulamentações)

  Segurança de Acesso (evita acesso não autorizado a programas e dados)

- **Confiabilidade:** O produto de software é imune à falhas? Atributos que evidenciam se o software é capaz de manter um nível de desempenho durante um período de tempo.

  Maturidade (evidência a freqüência de falhas)

  Tolerância à Falhas (capacidade do software de manter o nível do desempenho em falhas)

  Recuperabilidade (capacidade de restabelecer e restaurar dados após a falha)

- **Usabilidade:** O produto de software é fácil de usar? Atributos que evidenciam o esforço para se poder utilizar o software, bem como o julgamento individual deste uso.

  Operabilidade (evidencia a facilidade de operar e controlar as funções do software)

  Apreensibilidade (facilidade de aprendizado do software)

  Inteligibilidade (facilidade de entendimento dos conceitos utilizados pelo software)

- **Eficiência:** O produto de software é rápido? Mede o nível de desempenho em relação aos recursos.

  Comportamento em relação ao tempo (tempo de resposta, processamento e execução)

  Comportamento em relação aos recursos (quantidade de recursos e a duração do seu uso)

- **Manutenibilidade:** O produto de software é fácil de modificar? Atributos que evidenciam o esforço necessário para fazer modificações especificadas no software.

  Analisabilidade (evidencia a facilidade de modificação e remoção de defeitos)

  Modificabilidade (a facilidade de modificação e remoção dos defeitos)

  Estabilidade (evidencia a ausência de riscos de efeitos inesperados)

  Testabilidade (evidencia a facilidade de testar o software)

- **Portabilidade:** O produto de software é fácil de usar em outro ambiente? Atributos que evidenciam a capacidade de um software de se transferir para outro ambiente.

  Adaptabilidade (capacidade do software de ser adaptado a ambientes diferentes)

  Capacidade para ser instalado (facilidade de instalação do software)

  Conformidade (conformidade com o software com padrões ou convenções de portabilidade)

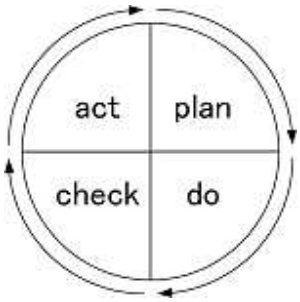
  Capacidade para substituir (capacidade do software de substituir outro software)

Além de todas estas características, utiliza-se uma outra ISO (14598) para descrever detalhadamente todos os passos para a avaliação destas características e sub-características de um software.

Gestão da Qualidade

Destaca-se o método PDCA, que pode ser aplicado a todos os processos, e é uma ferramenta que fornece o caráter científico à administração moderna, equiparado ao método científico moderno.

- **Planejamento("Plan"):** estabelecimento dos objetivos e dos processos necessários para a obtenção de resultados, de acordo com os requisitos do cliente e com a política da qualidade da organização.
- **Execução ("Do"):** implementação dos processos.
- **Verificação ("Check"):** monitoramento e medição de processos e produtos em relação à política, objetivos e requisitos para o produto, bem como comunicação dos resultados.
- **Ação ("Act"):** tomada de ações a fim de melhorar continuamente o desempenho dos processos.



Atividades da Qualidade Total (ISO 9001)

1 - Responsabilidade da administração

2 - Sistema da qualidade

3 - Análise crítica de contrato

4 - Controle de projeto

5 - Controle de documentos e de dados

6 - Aquisição

7 - Controle de produto fornecido pelo cliente

8 - Identificação e rastreabilidade de produto

9 - Controle de processo

10 - Inspeção e ensaios

11 - Controle de equipamentos de inspeção, medição e ensaios

12 - Situação de inspeção e ensaios

13 - Controle de produto não-conforme

14 - Ação corretiva e ação preventiva

15 - Manuseio, armazenamento, embalagem, preservação e entrega

16 - Controle de registros da qualidade

17 - Auditorias internas da qualidade

18 - Treinamento

19 - Serviços associados

20 - Técnicas estatísticas

Modelagem de Processos de Negócio

Conceitos básicos

Os negócios de qualquer tipo de organização necessitam do apoio da tecnologia e isso faz com que seja importante que o sistemas de informação sejam projetados para suportar e atender aos objetivos de negócio. Um negócio pode ser entendido como qualquer tipo de operação em andamento que tem ou usa recursos e tenha um ou mais objetivos. Para se projetar, entender ou se buscar melhorias no funcionamento de um negócio, torna-se essencial fazer o modelo de processos de negócio, que é uma visão simplificada da realidade complexa da organização. A modelagem permite, por exemplo, a abstração de detalhes irrelevantes em determinados momentos e concentrar foco em aspectos importantes.

A Modelagem de Processos de Negócio é um ponto central para os que gerenciam ou influenciam mudanças. Assuntos como Melhoria Contínua de Processos, Qualidade Total, Gerenciamento de Processos, Gerenciamento do Conhecimento, Melhoria da Qualidade, ISO 9000, e outros temas semelhantes, requerem habilidades de Modelagem de Processos de Negócio.

O Modelo de Negócios é uma abstração de como um negócio funciona. Ele se propõe a:

- prover uma visão simplificada da estrutura do negócio;
- atuar como uma base para a comunicação, melhorias ou inovações;
- definir os requisitos do sistema de informação que são necessários para apoiar o negócio.

Do ponto de vista de metodologias para a realização da modelagem de processos de negócio, existem diversas abordagens. Cada uma delas utiliza uma determinada notação e linguagem, mas abrangem o mesmo conjunto de passos/atividades:

1) definir o escopo da ação;
2) documentar a missão, estratégia, metas e objetivos da organização;
3) compreensão do processo/organização “como está” (“asis”);
4) avaliação do modelo obtido;
5) decidir quanto a: abandonar, contratar, manter como está, melhorar ou redesenhar os processos;
6) discutir e gerar idéias;
7) determinar características do processo desejado;
8) projetar os processos desejados (“to-be”).

Identificação e delimitação de processos de negócio

Um processo de negócio é um procedimento onde documentos, informações e tarefas são passadas entre participantes de acordo com um conjunto de regras definidas a serem alcançadas ou realizadas para o objetivo de negócio.

Os elementos de um processo são:

- **Trabalho / Processo:** é o objetivo direcionado. Como exemplo, a "Revisão de Currículos de Candidatos".
- **Objetivo:** razão para a realização do trabalho. Um exemplo "Seleção Criteriosa de empregados".
- **Atividades:** é a decomposição do trabalho em tarefas a serem realizadas. "Revisão do Currículo" e "Envio de Carta de Recusa" e "Agendamento da Entrevista" são exemplos de atividades.
- **Atores / Agentes:** se encarregam das atividades. No exemplo: "Gerente de RH", "Analista de RH" e "Secretária".
- **Entradas / Saídas:** são os produtos necessários / gerados a cada atividade / processo. "Currículo" e "Avaliação".
- **Regras:** são as dependências entre atividades. Podem aparecer em um fluxograma. "Candidato Qualificado?"

A identificação dos processos permitem um entendimento uniforme do negócio. Existem alguns modelos que permitem melhorar essa capacidade, como o **Modelo da Organização** (estrutura da organização), o **Modelo de Objetivos** (liga os objetivos da empresa com os processos de negócio), o **Modelo de Processos** (modela a estrutura de processos e atividades) e outros, como o **Modelo de Workflow, Modelo de Interação** e **Modelo de Casos de Uso**.

Técnicas de mapeamento de processos (modelos AS-IS)

A modelagem de processos de negócio visa a otimização dos processos executados em uma organização. Contudo, para haver melhorias, é fundamental que se saiba a forma como o negócio está sendo executado hoje ou, em outras palavras, quais os processos que atendem ao negócio atualmente. Este procedimento é conhecido como modelagem AS-IS.

Normalmente durante o mapeamento AS-IS são identificados problemas na execução, como por exemplo, a mesma tarefa sendo executada mais que uma vez e de formas diferentes; ou ainda a ordem de execução das tarefas originando gastos adicionais. Também inclui a **Análise de Risco** (participantes faltantes, interação entre eles, desacordos, resolução de conflitos).

Técnicas de análise e simulação de processos

Construção e mensuração de indicadores de processos

Técnicas de modelagem de processos (modelos TO-BE)

Para realizar a otimização dos processos de negócio, são elaborados os modelos TO-BE que identificam quais os processos que efetivamente devem ser executados, além da ordem de execução.

O modelo TO-BE pode ser originado a partir do AS-IS quando este for criado. Note, no entanto, que não existe obrigatoriedade da elaboração de ambos: se for considerado suficiente, pode ser realizado apenas o mapeamento TO-BE.

Modelagem de Processos em UML

A UML é adequada tanto para a modelagem de sistemas de software, quanto para modelagem de negócio pelos motivos:

- Muitos desenvolvedores já estão familiarizados com a linguagem, o que facilita a utilização.
- A utilização da mesma linguagem tanto para a modelagem de negócio quanto para a modelagem do sistema de software faz com que a documentação seja consistente e ainda facilita a comunicação entre as duas modelagens
- Existe um grande número de ferramentas disponíveis para a modelagem de negócio usando a UML.
- Com a UML é possível descrever:
    - Aspectos tanto estruturais quanto de negócio (como a organização, hierarquia de objetivos, ou recursos).
    - Aspectos comportamentais do negócio (como os processos).
    - Regras de negócio que afetam tanto a estrutura quanto o comportamento.

O uso da UML para modelar processos de negócio facilita uma melhor compreensão dos negócios da organização, na medida que diagramas como os de caso de uso e de seqüência podem ser usados.

http://www.uniandrade.br/publicacoes/revista/cientifica/MontaArtigo.asp?ID=156

Modelagem estrutural: diagramas de classe, pacotes lógicos, diagramas de objetos

Modelagem comportamental: diagramas de casos de uso, diagramas de interação, diagramas de atividade, DTEs (estados)

Notação

Artefatos

Atividades

Gerência de Projetos

Gerência de Projetos (ou Gestão de Projetos) é a aplicação de conhecimentos, habilidades e técnicas na elaboração de atividades relacionadas para atingir um conjunto de objetivos pré-definidos. O conhecimento e as práticas da gerência de projetos são melhores descritos em termos de seus processos componentes.

Reduzida à sua forma mais simples, a gerência de projetos é a disciplina de manter os riscos de fracasso em um nível tão baixo quanto necessário durante o ciclo de vida do projeto. O risco de fracasso aumenta de acordo com a presença de incerteza durante todos os estágios do projeto.

A gerência de um projeto de software pode influenciar o desempenho da organização de diversas maneiras:

- prover uma vantagem competitiva, permitindo respostas rápidas às mudanças de mercado;
- prover informação necessária, acurada e no tempo para permitir melhor tomada de decisão;
- reduzir o custo do negócio, substituindo capital por trabalho e automatizando transações da empresa;

Gerenciar um projeto requer a realização das seguintes atividades básicas:

- **Planejamento:** Identificar e planejar as atividades e tarefas, preparando planos de trabalho, estimativas e cronogramas viáveis. Identificar, planejar ações e eliminar fontes de risco, antes que aconteçam. Planejar as necessidades e a obtenção dos recursos humanos e técnicos, e definir os produtos intermediários e suas entregas.
- **Execução:** Criar condições para que as atividades previstas possam ser realizadas. Atuar para que o grupo realize as tarefas com sinergia e supere as dificuldades previstas e imprevistas, de forma a atingir os objetivos (cronogramas). Estabelecer regras para as solicitações de mudanças nas especificações do projeto e negociar as condições para a aceitação destas. Conduzir o processo para a obtenção de máxima qualidade do produto.
- **Controle:** Medir e avaliar o progresso do projeto (físico e financeiro) em relação ao planejado, tomar ações corretivas e comunicar o andamento do projeto.

PMBOK

PMBOK (*Project Management Body of Knowledge*) é um padrão de Gerência de Projetos desenvolvido pelo PMI (*Project Management Institute*). Ele é largamente aceito por diversas indústrias como sendo o padrão de fato de Gerência de Projetos.

O propósito principal do PMBOK é identificar o subconjunto de conhecimentos sobre a profissão, sendo aplicáveis para a maior parte dos projetos na maior parte do tempo. Outro propósito é prover um vocabulário único para a profissão, padronizando seus termos. Também é usado como referência básica para os exames de certificação do PMI.

Conceitos

Um projeto é um empreendimento temporário com o objetivo de criar um produto ou serviço único. Projetos possuem muitas características comuns: são executados por pessoas, têm recursos limitados e são planejados, executados e controlados.

Na abordagem tradicional, distinguimos cinco estágios no desenvolvimento de um projeto:

- Iniciação de projeto
- Planejamento de projeto
- Produção de projeto
- Monitoramento de projeto
- Fechamento (compleção) de projeto

Áreas de Conhecimento

O gerenciamento de projetos tenta adquirir controle sobre quatro variáveis: tempo, custo, qualidade e escopo. De acordo com a abordagem PMBOK, pode ser classificado em 9 (nove) áreas de conhecimento:

- **Gerência de Integração:** Envolve os processos necessários para garantir que os vários elementos do projeto sejam coordenados de forma apropriada. Envolve as negociações dos conflitos entre objetivos e alternativas concorrentes, com a finalidade de atingir ou exceder às necessidades e expectativas dos *stakeholders* interessados. Envolve:
    - Desenvolvimento do plano do plano do projeto
    - Execução do plano do projeto
    - Controle integrado de alterações
- **Gerência de Escopo:** Envolve os processos necessários para assegurar que o projeto contém todo o trabalho necessário para completar o projeto com sucesso. O seu foco principal é na definição e controle do que está ou não considerado no projeto. O escopo do projeto difere-se do escopo do produto na medida em que o escopo do projeto define o trabalho necessário para fazer o produto, e o escopo do produto define os recursos (atributos e comportamentos) do produto que está sendo criado.
    - Iniciação
    - Definição do escopo
    - Verificação de escopo
    - Controle de alterações de escopo
- **Gerência de Tempo:** Envolve os processos requeridos para garantir o término do projeto no tempo certo. Os processos desta fase são: identificar as atividades que devem ser feitas, arrumá-las em uma seqüência lógica, estimar a duração de cada atividade, desenvolver um calendário que exibe estas atividades em função do tempo, e monitorar, controlar e modificar o calendário com o passar do tempo.
    - Definição de atividades
    - Sequenciamento de atividades
    - Estimativa de duração das atividades
    - Desenvolvimento de cronograma
    - Controle de cronograma
- **Gerência de Custo:** Envolve os processos requeridos para garantir o término do projeto dentro do orçamento aprovado. Inclui o planejamento de recursos, estimativas e controle do *budget*. Na apuração de custos de software, o modelo de custos é uma fórmula, ou uma série de fórmulas, usada para prever os custos que provavelmente recairão sobre o projeto.
    - Planejamento de recursos
    - Estimativa de custos
    - Orçamento de custos
    - Controle de custos
- **Gerência da Qualidade:** Envolve os processos requeridos para assegurar que o projeto irá satisfazer as necessidades para o qual foi criado. Isto inclui "todas" as atividades de gerência geral que determina os objetivos, a política e as responsabilidades em relação à qualidade e suas implementações tais como: planejamento, controle, garantia e melhoria de qualidade dentro do sistema de qualidade.
    - Planejamento de qualidade
    - Garantia de qualidade
    - Controle de qualidade
- **Gerência de Recursos Humanos:** Envolve os processos requeridos para tornar o uso mais efetivo das pessoas que estão envolvidas no projeto. Isto inclui todos os *stakeholders*. Faz parte desta área os processos de aquisição de membros, desenvolvimento das equipes e motivação.
    - Planejamento organizacional
    - Aquisição de equipe (staff)
    - Desenvolvimento de equipe

- **Gerência de Comunicação:** Envolve os processos requeridos para assegurar a geração, coleção, disseminação, dissertação, armazenamento e disposição final de informação de projeto adequada e apropriadamente. Provê as ligações acerca de pessoas, idéias e informação que são necessárias para o sucesso do projeto. Todos os envolvidos devem enviar e receber comunicações na "linguagem" do projeto e devem entender como as comunicações individuais afetam o projeto como um todo.
    - Planejamento de comunicações
    - Distribuição de informações
    - Relatórios de desempenho
    - Encerramento administrativo
- **Gerência de Risco:** Evolve os processos relacionados à identificação, análise e resposta aos riscos de projetos. Isso inclui maximizar os resultados de ocorrências positivas e minimizar as conseqüências de eventos adversos. Um plano de risco inclui as técnicas para identificar o risco, identificar responsabilidades, estabelecimento dos resultados e relatórios detalhados.
    - Planejamento do gerenciamento de riscos
    - Identificação de riscos
    - Análise quantitativa de riscos
    - Monitoramento e controle dos riscos
- **Gerência de Aquisição:** Envolve os processos requeridos para adquirir bens e serviços externos à organização. Também chamado de Gerência de Compras e Subcontratação.
    - Planejamento das aquisições
    - Planejamento das solicitações
    - Seleção dos fornecedores
    - Administração do Contrato
    - Encerramento do Contrato

Análise de Risco

A análise de risco do PMBOK envolve definir a tripla (risco, probabilidade, impacto) para cada risco, bem como os impactos de ocorrências conjuntas de alguns riscos. Deve-se ainda definir limiares dos riscos, ou seja, a região ou valores em que pode ser tomada tanto a decisão de prosseguir quanto a de interromper o projeto diante da ocorrência dos riscos.

Um produto do processo de análise de risco é a tabela de riscos. A análise de risco ainda envolve a definição de planos de contingência para orientar as ações que devem ser tomadas diante da ocorrência dos riscos. Esses riscos envolvem tanto riscos tecnológicos quanto de negócio e organizacionais.

Stakeholders

Stakeholders são indivíduos e organizações ativamente envolvidos no projeto, cujos interesses são afetados (positiva ou negativamente) por ele, ou que exercem influência sobre o mesmo. Incluem o gerente de projeto, o cliente, a organização que fará o projeto, os membros da equipe de projeto, o patrocinador (indivíduo/grupo interno ou externo que provê os recursos financeiros para o projeto). Inclui também partes internas e externas, como fundadores, vendedores, fornecedores, agências governamentais, comunidades afetadas pelo projeto e a sociedade em geral. É uma boa prática identificar cada uma das partes envolvidas no projeto, identificar e gerenciar possíveis áreas de conflito entre elas. Uma orientação geral é resolver as diferenças entre as partes favorecendo o cliente.

O projeto deve levar em conta as seguintes questões ambientais ou sócio-econômicas:

- Padrões e regulamentos
- Questões pertinentes à internacionalização, quando for o caso.
- Questões de diferenças culturais (políticas, econômicas, éticas, étnicas, religiosas, etc), quando for o caso.
- Sustentabilidade social (econômica e ambiental)

Ciclo de Vida do Projeto

O conjunto de fases do projeto é chamado **"ciclo de vida do projeto"**. De um modo geral, as fases do projeto apresentam as seguintes características:

- Cada fase do projeto é marcada pela entrega de um ou mais produtos (deliverables), como estudos de viabilidade ou protótipos funcionais.
- No início de cada fase define-se o trabalho a ser feito e o pessoal envolvido na sua execução. O fim da fase é marcado por uma revisão dos produtos e do desempenho do projeto até o momento
- Uma fase começa quando termina a outra. Quando há *overlapping* entre as fases, chamamos essa prática de "fast tracking": nesses casos, começa-se a trabalhar nas próximas fases do projeto antes do fim da fase corrente (entrega e revisão dos produtos).
- Os custos são geralmente crescentes a medida em que a fase avança.
- Os riscos são geralmente decrescentes a medida em que a fase avança.
- A habilidade das partes envolvidas alterarem os produtos de cada fase é decrescente a medida em que a fase avança.

Estrutura da Organização

A estrutura da organização executora freqüentemente restringe a disponibilidade ou as condições sob as quais os recursos se tornam disponíveis para o projeto. Podem existir diversos modelos de estruturas:

- **Funcional:** É a clássica hierarquizada onde cada funcionário tem um superior bem definido, e são agrupados por especialidade (*engenharia, marketing, contabilidade*). Os departamentos são independentes, e os projetos dependem dos **gerentes funcionais**, onde está a coordenação do projeto.
- **Projetizada:** Os membros das equipes freqüentemente trabalham juntos, e a maioria dos recursos está envolvida em projetos, e **os gerentes de projeto** têm grande autoridade e independência.
- **Matricial:** É uma mistura das características funcional e projetizada.
    - **Fraca:** o papel do gerente de projetos é mais de um coordenador. Parecido com funcional.
    - **Forte:** o papel do gerente de projetos é mais autoritário e dedicado. Parece com projetos.

Projetos de Software

Estrutura de decomposição de trabalho (WBS)

Em Gerência de projetos, uma *Work breakdown structure* (WBS) é uma ferramenta de decomposição do trabalho do projeto em partes manejáveis. É estrutura em árvore exaustiva, hierárquica (de mais geral para mais específica) de *deliverables* e tarefas que precisam ser feitas para completar um projeto. Em português, às vezes é traduzida como Estrutura Analítica do Projeto, embora o termo WBS seja mais amplamente utilizado.

O objetivo de uma WBS é identificar elementos terminais (os itens reais a serem feitos em um projeto). Assim, a WBS serve como base para a maior parte do planejamento de projeto.

A *Work Breakdown Structure* é uma ferramenta bastante comum. Várias resoluções de trabalho do governo dos Estados Unidos o têm como requerimento. A WBS não é criada apenas para o gerente do projeto, mas para toda a equipe de execução do projeto.

A WBS deve ser completa, organizada e pequena o suficiente para que o progresso possa ser medido, mas não detalhada o suficiente para se tornar, ela mesma, um obstáculo para a realização do projeto.

Uma boa heurística a seguir é a regra do 8-80: exige-se que uma tarefa ocupe entre 8 e 80 horas de duração. É uma das partes mais importantes no plano do projeto. Ela serve como entrada para o desenvolvimento da agenda, atribuir funções e responsabilidades, gerenciar riscos, entre outros.

Um exemplo simples de work breakdown structure para pintar um quarto (orientado a atividades) é:

- **Preparar materiais:** Comprar tinta, Comprar escada, Comprar pincéis / rolos / removedor de papel de parede.
- **Preparar sala:** Remover papel de parede antigo, Remover decorações destacáveis, Cobrir chão com jornais velhos, Cobrir tomadas com fita, Cobrir móveis com lençóis velhos, Pintar a sala.
- **Limpar a sala:** Jogar fora, ou guardar a tinta que sobrou, Limpar pincéis e rolos, Jogar fora jornais velhos, Remover e limpar lençóis.

O tamanho da WBS não deve exceder 100-200 elementos terminais (se mais elementos terminais são requeridos, use subprojetos). A WBS deve ter de 3 a 4 níveis de profundidade. Essas sugestões derivam do fato que a nossa memória de curto prazo é limitada de 5 a 9 itens, e tendo terminado o tempo de planejamento de um projeto, quanto mais elementos terminais existirem, menos tempo sobra para prestar atenção em cada um deles. Conseqüentemente, as estimativas são menos pensadas.

Planejamento

O planejamento é uma das atividades fundamentais do processo de gerenciamento de software. No planejamento são colocados estimativas de esforço humano (pessoas-mês), duração cronológica (em tempo de calendário) e custo (em moeda). Em suma, planejar é **estimar**.

Acompanhamento e Controle

O acompanhamento e controle de um projeto de software é uma atividade de análise de risco.

Na especificação de um cronograma devemos estabelecer prazos para o esforço de desenvolvimento.

Dois métodos de determinação de cronograma são:

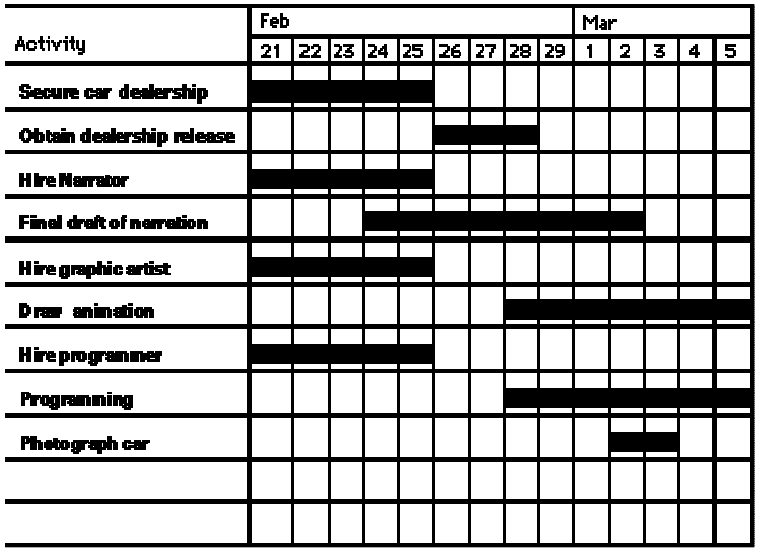
- **PERT** – *Program Evaluation and Review Technique* (método de avaliação e revisão de programa).
- **CPM** – *Critical Path Method* (método do caminho crítico).

Em ambas as técnicas existem uma rede de tarefas a serem desenvolvidas desde o começo até o final do projeto. A rede é definida ao se desenvolver uma lista de todas as tarefas associadas a um projeto específico e uma lista de disposições que indica em que ordem as tarefas devem ser executadas. As duas técnicas permitem:

- Determinar o caminho crítico – cadeia de tarefas que determina a duração do projeto.
- Estabelecer as estimativas de tempo prováveis para tarefas individuais ao aplicar modelos estatísticos.
- Calcular limites de tempo que definam uma janela de tempo para uma tarefa em particular.

Gráfico de Gantt

O **gráfico de Gantt** constitui-se de uma ótima ferramenta para representar o cronograma do plano do projeto. Permite uma comparação entre o previsto e o realizado, mostrando o andamento do projeto no tempo. Ele deve aparecer em dois níveis distintos, no mínimo:



- **Cronograma Mestre:** representando as atividades sumárias. Este cronograma dá uma visão geral dos prazos do projeto, mostra sua duração total e interessa principalmente ao usuário ou superiores nos níveis estratégicos ou tático de decisões.
- **Cronograma Parcial:** representando o detalhamento das atividades sumárias. Este cronograma relaciona as atividades e usa uma escala de tempo menor do que no cronograma mestre, ele interessa mais aos componentes do projeto para controle operacional.

Ferramentas de Controle de Qualidade

Algumas das ferramentas básicas para as melhorias da qualidade em uma organização são:

- **Formulário de dados:** os dados devem traduzir os fatos e devem ser a base da discussão e das ações de projetos de melhoria. Isto mostra a importância de se planejar muito bem o tipo de formulário que deverá ser usado para coletar os dados, o qual deve ser adaptado a cada situação. Existem, porém, alguns tipos básicos de formulários, que podem ser úteis em muitos casos: *check-sheet*, *data sheet* e *checklist*.
- **Diagrama de Pareto:** é uma figura simples que visa dar uma representação gráfica à estratificação. O modelo econômico de Pareto foi traduzido para a área da qualidade. Este princípio também conhecido como "Lei 20/80" pode ser detalhado nas mais variadas formas. Dentre elas, podem ser citadas: 20% do tempo gasto com itens importantes é responsável por 80% dos resultados; 20% do tempo gasto em planejamento economiza até 80% do tempo de execução; 20% dos clientes representam 80% do faturamento global; 20% dos correntistas são responsáveis por 80% dos depósitos; 20% das empresas detêm 80% do mercado; 20% dos defeitos são responsáveis por 80% das reclamações; e 20% dos clientes são responsáveis por 80% das vendas. Em linhas gerais, o que o diagrama de Pareto sugere é que existem elementos críticos e a eles deve-se prestar total atenção. Usa-se, assim, um modelo gráfico que os classifica em ordem decrescente de importância, a partir da esquerda.
- **Fluxograma:** é um diagrama que representa o fluxo (ou seqüência) das diversas etapas de um processo qualquer. Ao iniciar um projeto de melhoria, sua grande utilidade é fazer com que todos os participantes adquiriam uma visão completa do processo, ao mesmo tempo que permite que cada pessoa tenha melhor percepção de qual o seu papel no processo e de como seu trabalho influi no resultado final. Uma outra forma de utilizá-lo é fazer o fluxograma de como as atividades estão sendo feitas na prática e compará-lo com o fluxograma de como as atividades deveriam estar sendo feitas. Isto pode revelar a origem de alguns problemas. Podem ser usados quaisquer símbolos ou desenhos, desde que o entendimento seja fácil para todos.
- **Diagrama de Ishikawa:** também conhecida por vários outros nomes: diagrama causa-efeito, diagrama espinha de peixe, diagrama 4M, diagrama 5M, sendo uma ferramenta de valor indispensável, pois permite conhecer os problemas cada vez mais a fundo. Pode ser facilmente aprendida e imediatamente posta em prática por pessoas de qualquer nível dentro da empresa. Embora possa ser utilizada individualmente, a principal qualidade do diagrama de Ishikawa é sua capacidade de "focalizar" a discussão em grupo, estimando a participação de todos e direcionando o conhecimento de cada pessoa no sentido de identificar as causas ou os fatores responsáveis por um dado problema ou situação (efeito). Permite, assim, a organização das idéias e sua visualização agrupada destacando as áreas mais significativas. Existem três tipos de diagrama causa-e-efeito, definidos por Ishikawa: análise de dispersão, classificação de processo e enumeração de causas.
- **Histograma:** baseia-se na idéia de que cada fenômeno tem seu jeito próprio de variar. Então, pode-se visualizar esta variação, obtendo muita informação útil sobre o fenômeno. O histograma é exatamente isto: uma representação gráfica que nos permite visualizar a distribuição característica de um fenômeno ou processo.
- **Gráfico de Shewhart:** uma das mais fortes ferramentas para controlar processos e indicar oportunidades de melhorias nos mesmos. A capacidade das coisas variarem e formarem um padrão típico de variação, é uma das leis mais fundamentais da natureza: tudo varia, é impossível prever um resultado individual, contudo, um grupo de resultados, vindos do mesmo conjunto de causas, tende a ser previsível, seguindo uma certa distribuição; quando um conjunto de causas é perturbado por causas externas, a distribuição de resultados se altera.
- **Gráfico de correlação:** permite a avaliação da relação existente entre duas variáveis, parâmetros ou características de interesse. Como exemplo, pode ser citado o estudo da relação entre o peso de uma pessoa e sua altura. Se for obtido o peso e a altura de várias pessoas, o gráfico de correlação poderá representar em cada ponto uma pessoa. A característica altura é uma variável independente, pois a idéia é obter o peso de uma pessoa dada sua altura. O peso é a variável dependente. À medida que os dados apresentem uma tendência, ou comportamento razoável previsível, pode-se dizer que as variáveis têm correlação. A tendência mais comum é o comportamento linear, ou seja, os pontos tendem a se alinharem e com isso pode-se imaginar uma reta que representa a correlação entre essas variáveis.

Coleta de Métricas de Software

Medições e métricas auxiliam a entender o processo usado para se desenvolver um produto. O processo é medido a fim de melhorá-lo e o produto é medido para aumentar sua qualidade.

As **medidas** são uma forma clara de avaliação de produtividade no desenvolvimento de software. Através da obtenção de medidas relativas à produtividade e à qualidade, é possível que metas de melhorias no processo de desenvolvimento sejam estabelecidas como forma de incrementar estes dois fatores.

Métricas de software referem-se a uma ampla variedade de medidas de software. O gerenciamento de projeto preocupa-se com métricas de produtividade e de qualidade – medidas do resultado do desenvolvimento de software como uma função do esforço aplicado e medidas da adequação ao uso do resultado que é produzido.

Podemos dividir o domínio das métricas de software em:

- **Métricas de produtividade:** concentram-se na saída do processo de engenharia de software com o objetivo de avaliar o próprio processo.
- **Métricas de qualidade:** oferecem uma indicação de quão estreitamente o software conforma-se às exigências implícitas e explícitas do cliente.
- **Métricas técnicas:** concentram-se nas características do software (exemplo: complexidade lógica e modularidade).

Ou ainda podemos dividir o domínio das métricas de uma outra forma:

- **Métricas orientadas ao tamanho:** medidas são derivadas a partir de atributos de tamanho do software como linhas de código (LOCs), esforço, custo, quantidade de documentação (PODs), etc...
- **Métricas orientadas para a função:** oferecem medidas indiretas. Ex: a Análise de Pontos por Função (APF)
- **Métricas orientadas às pessoas:** usam informações sobre a maneira pela qual as pessoas desenvolvem software e percepções humanas sobre a efetividade das ferramentas e métodos.

Estimativa de tamanho de software

As métricas de tamanho de software surgiram com o objetivo de estimar o esforço(número de pessoas-hora) e o prazo associados ao desenvolvimento de sistemas. Para saber o custo de um projeto de software precisamos saber o esforço necessário para desenvolvê-lo e para determinar o esforço precisamos saber o tamanho do projeto de software.

Existem várias técnicas de estimativas de tamanho de software, e a seguir são apresentadas as mais importantes:

- **COCOMO:** modelo desenvolvido para estimar o esforço de desenvolvimento, prazos e tamanho da equipe para projetos de software. Utiliza equações desenvolvidas por Boehm (BARRY,1981) para prever o número de programadores-mês e o tempo de desenvolvimento; podem ser calculados usando medidas de linhas de código ou Pontos de Função. Devem ser realizados ajustes nas equações a fim de representar as influências sobre os atributos, hardware e software durante o ciclo de vida do projeto. Uma desvantagem desta técnica é que os coeficientes da métrica (a,b,c,d) não são aplicáveis a tamanho ou seja a produtividade é diferente, o que torna difícil realizar comparações.
- **Linhas de Código (LOC):** a técnica de mensuração por linhas de código é uma das mais antigas medidas de tamanho de projeto de desenvolvimento de software. Ela consiste na contagem da quantidade de número de linhas de código de um programa de software. Além de ser muito simples é também muito fácil automatizar sua implementação, mas, apresenta algumas desvantagens dentre as quais citamos: a dependência da linguagem de software e do desenvolvedor; ausência de padrão de contagem e o fato de somente poder ser aplicada na fase de codificação.
- **Metricas de Hasltead:** é um conjunto de métricas proposto por Maurice Halstead (HASLTEAD,1977). O princípio desse método está na análise e quantificação de operandos e operadores e no conceito de que a partir do conhecimento das medidas, consegue-se quantificar os vocábulos e a extensão do algoritmo do estudo.
- **Puttnam´s Slim Model:** é um modelo de estimativa que busca medir esforço e prazo através da dinâmica de múltiplas variáveis que pressupõe distribuição de esforços específicos ao longo da existência de um projeto de software. Relaciona o número de linhas de código ao tempo e esforço de desenvolvimento. Uma desvantagem da técnica é sua vinculação à linguagem usada e a exigência de certo tempo para obter valores reais para os parâmetros da fórmula.
- **Delphi:** é uma técnica que se resume à consulta de especialistas de determinada área, em determinada linguagem e/ou determinado assunto para que, usando sua experiência e entendimento do projeto proposto, façam estimativas devidas. Devem ser feitas várias estimativas do mesmo projeto, pois é comum que elas carreguem influências e tendências dos especialistas. É um método empírico, baseado em experiências profissionais que podem ser subjetivas.
- **PSP (Personal Software Process):** é uma técnica derivada do SEI-CMM (Software Engineering Institute – Capability Matutiry Model) que foi desenvolvida com a função de capacitar, melhorar e otimizar o processo individual de trabalho. A técnica divide-se em sete etapas, sendo que nas etapas PSP0, PSP0.1 e PSP1 estima-se o tamanho e o tempo necessário para o desenvolvimento do produto.
- **PCU (Pontos por Caso de Uso):** foram criados por Gustav Karner em 1993 como uma adaptação específica dos Pontos de Função para medir o tamanho de projetos de software orientados a objeto. Explora o modelo e descrição do caso de uso, substituindo algumas características técnicas proposta pelos Pontos de Função. É um método simples

e de fácil utilização mas ainda esta em fase de pesquisas e não existem regras de contagem padronizadas. Têm se estudado a aplicação em conjunto da PCU e APF tentando explorar a relação entre elas existente.(EDMÉIA,2004)

- **Análise por Pontos de Função:** Busca medir a complexidade do produto pela quantificação de funcionalidade expressa pela visão que o usuário tem do mesmo. O modelo mede o que é o sistema, o seu tamanho funcional e não como este será, além de medir a relação do sistema com usuários e outro sistemas. É independente da tecnologia usada e mede uma aplicação pelas funções desempenhadas para/e por solicitação do usuário final.; podendo também ser usada em estimativas.
- **Contagem de Pontos de Função segundo o NESMA:** Além do IFPUG, o NESMA também promove o uso de pontos de função e publica o seu próprio manual de contagem complacente com o manual do IFPUG. O manual da NESMA apresenta três tipos de contagens por pontos de função: a contagem indicativa de ponto de função, a contagem estimada de ponto de função e a contagem detalhada de pontos de função. A contagem indicativa é muito usada, nela são identificados os grupamentos de dados relativos à natureza do negócio, conforme a visão do usuário. Estes grupamentos são classificados como Internos (mantidos pela aplicação e Externos (referenciados ou consultados pela aplicação). Para calcular o tamanho de uma aplicação em Pontos de Função não Ajustados (PFNA) a NESMA recomenda a seguinte fórmula: PFNA = (35 * I) + (15 * E).

Análise de Pontos por Função (APF)

É um método padronizado para a medição de projetos de desenvolvimento de software, visando estabelecer uma medida de tamanho, em Pontos de Função (PF), considerando a funcionalidade implementada, sob o ponto de vista do usuário (produtividade). Os objetivos da APF são:

- medir a funcionalidade requisitada e recebida pelo usuário;
- medir projetos de desenvolvimento e manutenção de software, independente da tecnologia a ser implementada.

As organizações podem aplicar a Análise de Pontos por Função como:

- uma ferramenta para determinar o tamanho do pacote de software adquirido através da contagem de todos os Pontos por Função incluídos no pacote.
- uma ferramenta para apoiar a análise da qualidade e da produtividade.
- um mecanismo para estimar custos e recursos envolvidos em projetos de desenvolvimento e manutenção de software, além de um fator de normalização para comparação de software.

A técnica Análise de Pontos por Função é de grande utilidade para garantir a completeza da especificação de requisitos. Uma contagem de Pontos de Função é realizada usando a informação contida na especificação do sistema sob a visão do usuário, que representa uma descrição formal das necessidades do negócio na sua linguagem. Assim, contagem pode servir como uma revisão dos requisitos do sistema com o usuário.

A APF também pode ser utilizada para estabelecer medições para a gerência de requisitos, como indicadores para apoiar o controle da estabilidade e rastreabilidade de requisitos e análise de impacto das mudanças.

O gerente de projetos pode utilizar APF para aferir as mudanças de requisitos no decorrer do desenvolvimento, realizando uma recontagem e gerando novas estimativas. Para melhorar a precisão das estimativas dos novos projetos, a organização deve utilizar um banco de dados histórico, contendo atributos de projetos anteriores.

Parâmetro de medida	Contagem	Fator de ponderação			Pontos
		Simples	Médio	Complexo	
Número de entradas do usuário		X 3	X 4	X 6	
Número de saídas do usuário		X 4	X 5	X 7	
Número de consultas do usuário		X 3	X 4	X 6	
Número de arquivos		X 7	X 10	X 15	
Número de interfaces externas		X 5	X 7	X 10	
Contagem – total					

COCOMO

Um método de estimativas é o COCOMO (*COnstructive COst MOdel*), que foi desenvolvido por Barry Boehm, para estimar esforço, prazo, custo e tamanho da equipe para um projeto de software.

O método foi derivado de um data set que compreendia 63 projetos cobrindo áreas como: negócios, controle, científica, suporte e sistema operacional. O COCOMO classifica o projeto em 3 tipos:

- **Modelo Orgânico (Convencional):** software de aplicação, equipes pequenas, desenvolvimento "in-house".
- **Semidestacado (Difuso):** software utilitário ou básico, equipe com experiência média em aplicações relacionadas.
- **Embutido (Restrito):** software de sistema, grandes restrições de operação, especificações rígidas, prazos curtos.

Além disso algumas de suas características são:

- O COCOMO supõe que não haverá alterações substanciais nas especificações de requisitos do Sistema.
- É necessário levantar as características de todos os projetos já desenvolvidos na instalação.
- A medida do tamanho do projeto é dada em KDSI (*Thousands of Delivered Source Instructions*).
- Permite estimativas com 20% de variação em relação ao custo real e com 30% de variação em relação ao prazo.
- Além disso consiste em três implementações (COCOMO Básico, Intermediário e Avançado)

Implementação Básica

É um modelo estático que calcula o esforço de desenvolvimento de software e seu custo, em função do tamanho de linhas de códigos desenvolvidas.

$$E = {}_{ab}(KLOC)^{bb}$$
$$D = {}_{cb}(E)^{db}$$
$$P = E / D$$

Onde E é o esforço aplicado pela pessoa no mês, D é o tempo de desenvolvimento em meses cronológicos, KLOC é o número calculado de linhas de código para o projeto (expressado em milhares), e P é o número das pessoas necessário. Os coeficientes ab, bb, cb e db são dados na seguinte tabela:

Projeto de Software		**ab**	**bb**	**cb**	**db**
Início	2.4	1.05	2.5	0.38	
Meio	3.0	1.12	2.5	0.35	
Fim	3.6	1.20	2.5	0.32	

Cocomo básico é bom por ser rápido em estimativas e custos de software, mas sua exatidão é limitada por causa de sua falta de fatores para explicar as diferenças entre ferramentas, qualidade de pessoal e experiência, uso de ferramentas modernas e técnicas, e outros atributos de projeto que influenciam nos custos de software.

Pode-se conseguir um software interativo auxiliar na estimativa de custos e prazos de projetos de sistemas. A partir de um conjunto de atributos, premissas e modos de desenvolvimento o COCOMO estima os prazos, custos e recursos necessários para cada estapa do ciclo de vida do produto.

Implementação Intermediária

Calcula o esforço de desenvolvimento de software em função do tamanho do programa, que inclui custo, avaliação subjetiva do produto, ferramentas, pessoal e atributos de projeto.

$$E = {}_{ai}(LOC)^{(bi)}.EAF$$

Onde E é o esforço aplicado em pessoas por mês, LOC é o número de linhas de código para o projeto e EAF é o fator calculado acima. Os coeficientes ai e o bi são dados na próxima tabela.

Projeto de Software	ai	bi
Início	3.2	1.05
Meio	3.0	1.12
Fim	2.8	1.20

O método intermediário é uma extensão do método básico, mas com mais categorias de controle como: Atributos do produto, Atributos de hardware, Atributos pessoais, Atributos do projeto.

Implementação Avançada

Também conhecido como Implementação Detalhada. São incorporadas características da versão intermediária com uma avaliação de impacto de custo em cada passo de todo o projeto. Apresenta técnicas para estimar tanto em nível de módulo, como subsistema e Sistema, individualizando, a cada fase do projeto, os atributos de Custo.

Análise de Desempenho

A avaliação de desempenho de sistemas computacionais consiste em um conjunto de técnicas e metodologias que permitem responder à questão de como se obter o melhor desempenho de um sistema computacional a um dado custo. A atividade de avaliação de desempenho compreende:

- Seleção de técnicas de avaliação, métricas de desempenho e cargas de trabalho;
- A correta condução das medidas de desempenho e das simulações;
- A utilização de técnicas estatísticas apropriadas para comparar as diversas alternativas;
- O desenvolvimento dos experimentos de medição e simulação visando prover a maior quantidade de informação com o menor esforço;
- A utilização de modelos de filas para analisar o desempenho dos sistemas.

Modelos Analíticos

Um modelo analítico é uma construção matemática que representa aspectos-chave de um sistema computacional. Estes modelos são uma ferramenta excelente para uma rápida avaliação de um produto novo ou modificado.

- **Modelo de Fila:** um modelo de fila representa um sistema como um conjunto de centros de serviço que processam requisições de clientes, denominadas tarefas, transações ou chegadas. Os centros de serviços podem ser qualquer recurso ativo como a CPU, o disco ou a conexão de rede. Os modelos de fila podem prever a latência, a vazão e a utilização do sistema e seus componentes.
- **Modelo Assintótico:** é a aplicação mais simples de redes de filas. São utilizados para prever o melhor e o pior caso de latência e vazão de um único centro de serviços. Um uso típico é examinar o comportamento do ponto de contenção de um conjunto de recursos sob diferentes cargas.
- **Modelo de Rede Aberta:** neste modelo, os clientes passam através do sistema uma única vez. Por exemplo, um dispositivo de rede processando um fluxo de pacotes pode ser modelado com um sistema aberto. O número de clientes do sistema é calculado pela taxa de chegada de clientes e as características do centro de serviço. Se existirem tipos diferentes de clientes, um modelo aberto de múltiplas classes poderá ser utilizado.
- **Modelo de Rede Fechada:** neste caso, o número de clientes é fixo. O exemplo clássico é o sistema computacional de tempo compartilhado com um número fixo de terminais de usuário. Neste modelo, o número de clientes é conhecido e a vazão é calculada. Um método interativo conhecido como Análise do Valor Médio (*Mean Value Analysis* ou MVA) é usado para calcular vazão e latência. A exemplo dos modelos abertos, os modelos fechados podem ser de classe única ou de múltiplas classes. As duas variações do algoritmo MVA são: solução exata e solução aproximada.
- **Cadeias de Markov:** são usadas para modelar sistemas como um conjunto finito de estados com uma taxa conhecida de transições entre os estados. Este modelo é adequado para prever a disponibilidade de um sistema, dadas as ta-

xas de falha de um componente e as taxas de recuperação. Também podem ser usadas para modelar *buffers* de filas que armazenam um pequeno número de itens idênticos.

Simulação

Conceitualmente, a simulação modela um sistema do mundo real como um programa de computador. A simulação permite que um sistema seja modelado em qualquer nível de detalhe: de uma tradução direta de um modelo de redes de filas à captura de todo aspecto do comportamento do sistema. A simulação suporta qualquer coleção de métricas de desempenho que possam ser definidas. Existem três técnicas básicas para gerar cargas de trabalho para realizar simulações:

- **Simulação Estocástica:** descreve a chegada de padrões de clientes e outros aspectos da carga por amostragem a partir de uma distribuição probabilística. Muitas cargas podem ser descritas precisamente pelo uso de uma distribuição apropriada. As cargas estocásticas são uma boa escolha quando a informação detalhada sobre a carga não está disponível ou quando existe necessidade de variar as características da carga. A geração de carga é eficiente e não necessita de grandes arquivos de dados.
- **Simulação por rastreamento:** representa a carga como uma sequência de operações ou requisições. Por exemplo, para a simulação de um servidor Web, uma sequência de requisições http deve ser rastreada, enquanto na simulação de uma CPU, as operações do micro-código devem ser rastreadas. Se dados de rastreamento que representam corretamente a carga estão disponíveis, obteremos bons resultados de simulação, sendo desnecessário escrever o código que modela a carga. A desvantagem desta técnica é que montar uma coleção de rastreamentos é um esforço não trivial e o tamanho dos arquivos de dados é muito grande.
- **Simulação baseada em execução:** é utilizada para modelar processadores em detalhes. A entrada de uma simulação baseada em execução é o mesmo código executável que seria processado no sistema real.

A principal desvantagem de utilizar qualquer técnica de simulação é o esforço necessário para escrever e validar o programa de simulação e os consideráveis requisitos de processamento (tempo de CPU para todas as simulações e espaço em disco para rastreamentos). Em geral, a simulação é significativamente mais lenta do que um sistema real: uma hora de simulação pode significar alguns poucos segundos do tempo real, mas pode ocorrer o contrário.

Medição de Desempenho

Devemos considerar, ao planejar uma medição de desempenho, o propósito da medição, a seleção da carga de trabalho e sua implementação, os dados a serem coletados, a instrumentação utilizada na coleta destes dados e a forma de validação dos resultados. Todos os experimentos de medição de desempenho requerem três elementos básicos: o sistema sob teste, os geradores de carga de trabalho e a instrumentação que irá coletar os dados de desempenho.

Benchmarks são uma categoria importante de cargas sintéticas, definindo não somente a carga de trabalho, mas as métricas de desempenho a serem coletadas e relatadas. *Benchmarks* desenvolvidos independentemente são sempre preferidos porque permitem aos clientes compararem resultados de múltiplos fornecedores.

O sistema sob teste pode ser instrumentado para coletar uma variedade de métricas. Muitos *benchmarks* simplesmente medem a latência e a taxa de serviço. Entretanto, a informação sobre a utilização do processador e de outros recursos é muito útil na identificação de pontos de contenção no sistema ou para futuros esforços de modelagem. Existem contadores implementados tanto em software quanto em hardware que fornecem informação útil de desempenho. Por exemplo, falhas de página podem ser gravadas em um contador na CPU, no kernel do sistema operacional ou em ambos. Rastreamento das operações ou eventos também podem ser coletados.

O SPEC (*Standard Performance Evaluation Corporation*) é uma organização que busca estabelecer, manter e endossar um conjunto padrão relevante de *benchmarks* e métricas de avaliação de desempenho.

Métricas de Desempenho

Todas as métricas de desempenho baseiam-se no comportamento do sistema ao longo do tempo. As três classes principais de métricas que podem ser observadas por um usuário ou uma outra entidade externa ao sistema são:

- **Latência ou Tempo de Resposta:** que mede o atraso entre a requisição de alguma ação e a obtenção do resultado. É medida em unidades de tempo decorrido, e deve especificar um evento de início e um evento de término (exemplo: tempo decorrido entre digitar um novo endereço em um navegador Web e a página ser completamente exibida).

- **Taxa de serviço:** que mede quantidade de trabalho realizado por unidade de tempo ou a taxa em que novos resultados são obtidos. Como exemplo, número de transações completas por minuto, gigabytes de dados escritos em fita por hora, número de acesso à memória por segundo, e assim por diante.
- **Disponibilidade:** que mede quanto tempo o sistema está disponível para operação normal. Embora seja medida em fração de tempo (ex: 0,96), ela sozinha não é completa, necessitando da métrica de confiabilidade MTBF (*Mean Time Between Failures*) que indica o período médio em que um sistema é utilizável, e a métrica MTTR (*Mean Time to Repair*), que quantifica o tempo necessário para recuperar o sistema de uma falha..

A quarta classe de métrica, que seria a métrica de utilização, só pode ser observada de dentro do sistema. A informação de utilização é vital para entender e prever o desempenho do sistema. Utilização é a fração de tempo em que um componente do sistema, por exemplo, a CPU, o disco ou a rede realizou serviço, em relação a um período de tempo observado. Segue dessa definição que os valores de utilização estão entre 0 e 1. Na prática, o tempo de resposta aumenta rapidamente quando a utilização atinge o máximo, por isso sistemas são projetados para manter uma utilização abaixo da capacidade máxima (60-80%).

Planejamento de Capacidade

A metodologia de planejamento de capacidade de um software envolve o entendimento do ambiente, caracterização da carga, previsão da carga e análise de custo / desempenho, utilizando os seguintes modelos:

- **Modelo de custo:** é representado em termos monetários, onde se avalia o custo do sistema atual (aquisição e manutenção). É utilizado para avaliar o custo / benefício de uma possível atualização do sistema, para definir se é viável ou não financeiramente. Se trata de uma visão mais gerencial do processo de decisão.
- **Modelo de carga:** é baseado na utilização dos recursos do sistema levando em conta a quantidade de recursos e suas respectivas taxas de utilização. Esses modelos são fortemente apoiados em modelos matemáticos, para fazer uma representação o mais fiel possível do ambiente real, para que o processo de avaliação de desempenho seja válido.
- **Modelo de desempenho:** trata de avaliar a situação atual do sistema, confrontando os recursos disponíveis e a sua utilização pelos usuários. O modelo é baseado em observações sobre o comportamento do sistema, por um período de tempo, de forma a caracterizar um padrão de utilização, para que se possa ter uma base de comparação que seria utilizada como referência em um cenário de planejamento de capacidade.

Gestão e Recursos Informacionais

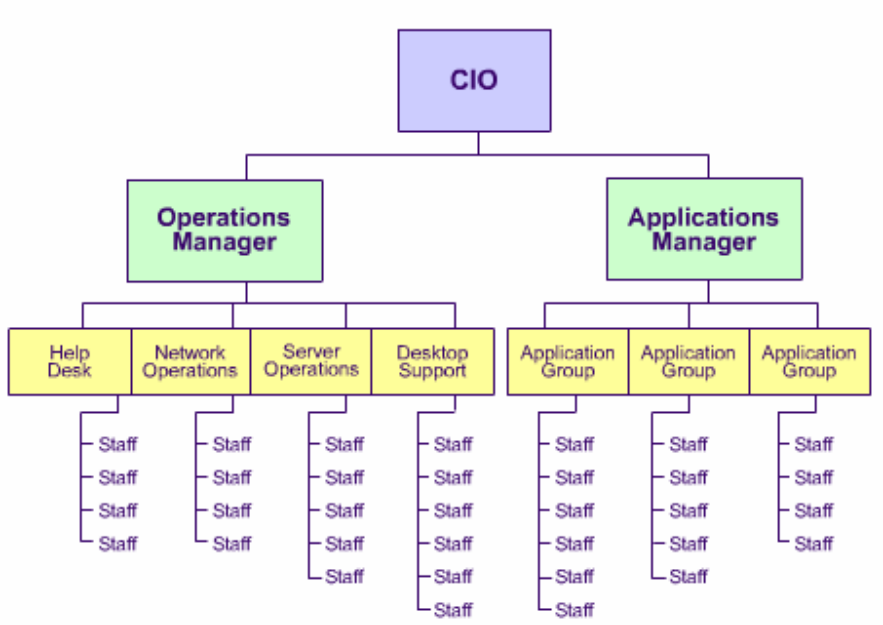
A área de TI ampliou o seu domínio com o crescimento das redes de computadores e a sua utilização voltado para negócios.

Na área dos negócios on-line (*E-business*) existem as tecnologias de *Data Warehousing* e *Data Mining*, que baseiam-se na busca de informações estratégicas em grandes bancos de dados. Na mesma área existem também aplicativos de atendimento ao cliente, ou CRM, de planejamento de recursos empresariais, ou ERP e gerenciamento de recursos humanos, ou HRMS. Na área de comércio eletrônico existem sistemas B2B, que são processos automatizados entre parceiros de negócios, e sistemas B2C, que leva o negócio diretamente ao cliente, exemplificado pelas lojas virtuais (*Amazon*). Existe também a área do marketing eletrônico, que além de incluir os sistemas CRM, também permite o marketing personalizado e promoções através da Internet, diretamente ao consumidor, atuando às vezes de forma indesejável, como por e-mails não solicitados (*Spams*).

Serviço de TI

Um serviço bem sucedido em TI é um conjunto de facilidades sustentadas pelo fornecedor (do serviço) que completa um objetivo do negócio do cliente.

As organizações buscam na tecnologia da informação meios para ajudá-las a se tornar mais eficientes e efetivas no seu negócio. Usuários finais são os consumidores dos serviços de TI, que cada vez exigem mais dos serviços que utilizam em seu trabalho. Simultaneamente, as organizações procuram em TI um aumento no valor do seu negócio e retorno do investimento.



O objetivo de uma cultura de serviços é garantir que a organização de TI fornece os serviços que alcançam as expectativas e necessidades da organização a um custo apropriado. Além disso existem políticas que devem estar claras para cada membro das equipes, que devem saber como conduzir uma tarefa específica e lidar com diversas situações. Também devem possuir funções e responsabilidades bem definidas.

Funções de TI

De forma geral, as funções da área de TI recaem em três categorias:

- **Desenvolvimento:** envolve o desenvolvimento dos softwares propriamente ditos. O CMMI é um exemplo de modelo de qualidade de processos dessa área.
- **Operações:** refere-se às atividades diárias que devem ser executadas para manter a infra-estrutura de TI funcionando adequadamente. A ISO e o ITIL são modelos aplicáveis a essa função.
- **Controle:** envolve a habilidade para medir e auditor a performance de uma infra-estrutura de TI. O COBIT (segurança, etc) e o SixSigma oferecem modelos para a função de controle.

Modelos de Governança em TI: ITIL

ITIL (*Information Technology Infrastructure Library*) é um conjunto de melhores práticas que ajudas as organizações de TI a entregar serviços de alta qualidade que agregam valor a sua organização.

ITIL consiste em uma biblioteca de livros que aborda tópicos relacionados ao Gerenciamento de Serviços (*Service Management*). É recomendado que organizações adotem uma visão holística das capacidades de seus serviços, tendo como principal objetivo o provisionamento ótimo dos processos operacionais e de gerenciamento, para suportar os requisitos de negócio a um custo justificável. Segundo o modelo definido pelo ITIL, toda disciplina de um modelo de TI deve adotar técnicas de segurança dentro de seus próprios processos.

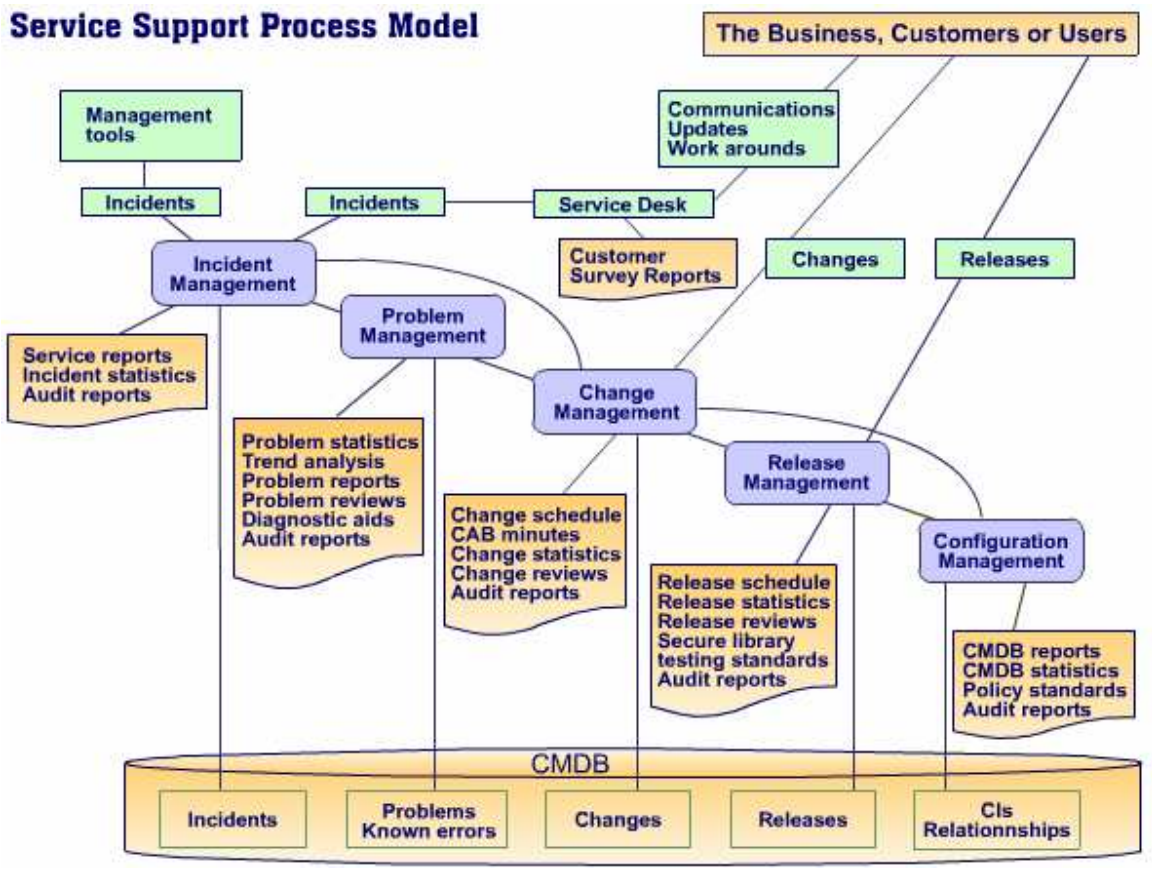
TI deve ajudar uma organização em alcançar os seus objetivos estratégicos da maneira mais eficiente possível. ITIL complementa essa tarefa, ajudando as organizações a manter o foco em seus objetivos. ITIL ajuda as organizações a gerenciar de forma apropriada toda a sua infra-estrutura, com uma contribuição e suporte ótimos. Alguns dos seus benefícios são:

- Aumento da Qualidade e Melhora Contínua
- Gerenciamento Financeiro Efetivo
- Cultura de Serviços
- Processos trabalhando juntos e de forma consistente
- Foco nos Serviços e Negócios
- Interface definida e Ferramentas integradas

As áreas do Service Management Process são duas:

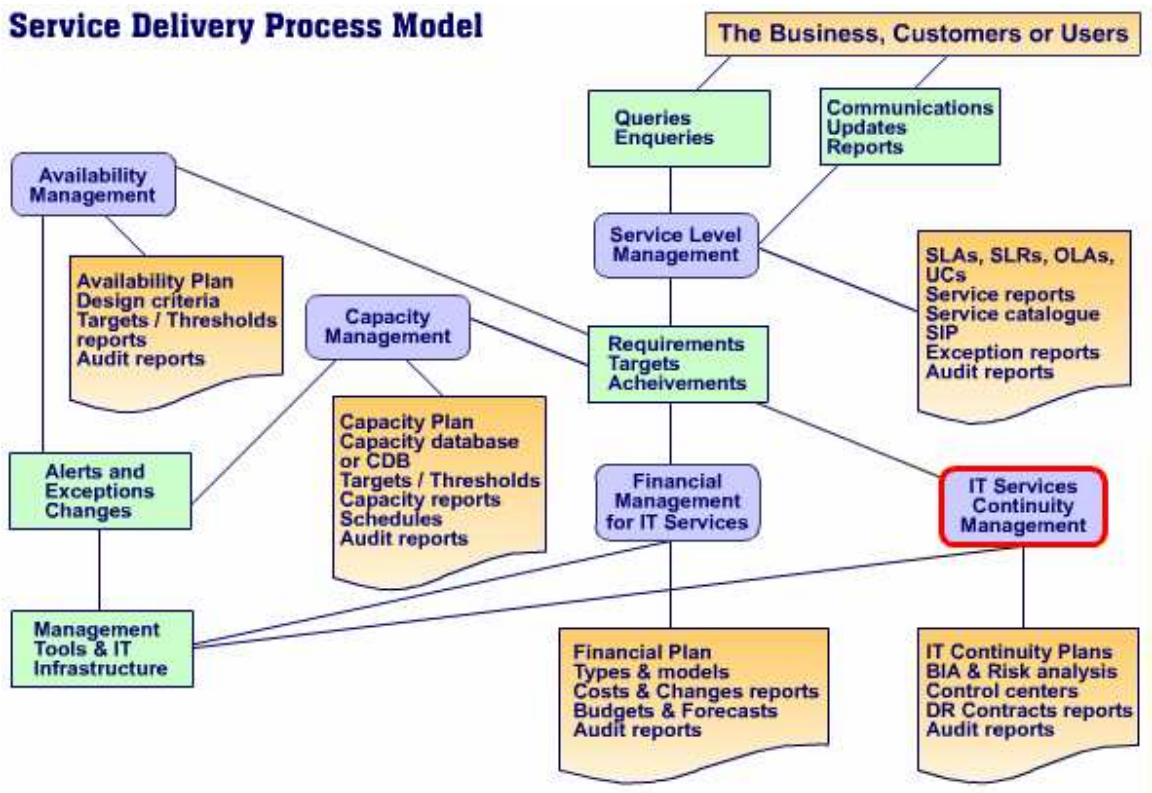
- Service Support
- Service Delivery

Service Support



- **Service Desk:** atua como um ponto focal de contato para os usuários finais de uma organização de TI, ajudando os usuários a resolver problemas e restaurar a operação rapidamente. É o comunicador entre os processos.
- **Incident Management:** é o primeiro processo executado pela função do *Service Desk*, onde incidentes são detectar e resolvidos. Um incidente pode ser definido como um evento que não é parte da operação normal de um serviço, e que causa (ou pode causar) interrupção ou redução na qualidade do serviço.
- **Problem Management:** é o processo de detectar e remover erros de uma infra-estrutura de TI, analisando a causa (*root cause*). Um problema é um incidente ou conjunto de incidentes significativos com sintomas em comum que indicam um erro simples. Problemas são erros na infra-estrutura, podendo levar a mais incidentes.
- **Change Management:** é o processo de controlar uma mudança aos itens de configuração (CI), garantindo que os procedimentos apropriados são seguidos. Um Comitê de mudanças (*Change Advisory Board*) é responsável por assessorar o impacto técnico e funcional da requisição da mudança (*Change Request*).
- **Release Management:** após as mudanças serem aprovadas no processo anterior, deve ser garantido que apenas itens adequadamente testados e aprovados existam na infra-estrutura de TI, seguramente armazenada, controlada e distribuída para os itens de *hardware* e *software* correspondentes. Envolve também preparação e treinamento.
- **Configuration Management:** é o processo de rastrear todos itens da infra-estrutura de TI, identificando e relatando os seus componentes, incluindo as versões, relacionamentos e componentes constituintes. A informação de todos os itens de configuração (CI) devem ser mantidos em um banco de dados apropriado.

Service Delivery



- **Capacity Management:** é um processo que procura garantir as necessidades atuais e futuras da capacidade e desempenho dos recursos. Aumentar a capacidade gera custos, mas mantê-la abaixo do esperado pelos clientes também, e o objetivo desse processo é garantir um custo eficiente dimensionando a capacidade de forma correta.
- **Availability Management:** é o processo que busca otimizar a disponibilidade dos serviços através de suporte próativo e reativo, em alinhamento com os requerimentos de negócios a um custo justificável. Algumas de suas atividades são: a monitoração, necessidade de manutenção e rastreamento da informação de gerenciamento (produção).

- **Service Level Management:** é um processo que alinha o provisionamento dos serviços com os requerimentos de negócio, quantificando negociando e monitorando o nível dos serviços. É responsável pela melhora no relacionamento entre os serviços de TI e os negócios. Um termo chave para este é o SLA (*Service Level Agreement*).
- **IT Financial Management:** processo que identifica, monitora, e se necessário, revê os custos para fornecer serviços de TI. Um modelo de custo e sistema de *budget* permitem que decisões referentes à custos de TI sejam tomadas como qualquer decisão de negócio. O modelo de custo registra os custos dos clientes, atividades e serviços.
- **IT Service Continuity Management:** é um processo que garante que a organização possui a capacidade de restaurar os seus serviços no caso de um desastre, além de procurar reduzir riscos. Os serviços críticos são identificados, as suas vulnerabilidades e ameaças medidos.

Modelos de Auditoria em TI – COBIT

O CobiT (*Control Objectives for Information and related Technology*) é uma ferramenta eficiente para auxiliar o gerenciamento e controle das iniciativas de TI nas empresas.

O CobiT é um guia para a gestão de TI recomendado pelo ISACF (*Information Systems Audit and Control Foundation*). O CobiT inclui recursos tais como um sumário executivo, um framework, controle de objetivos, mapas de auditoria, um conjunto de ferramentas de implementação e um guia com técnicas de gerenciamento. As práticas de gestão do CobiT são recomendadas pelos peritos em gestão de TI que ajudam a otimizar os investimentos de TI e fornecem métricas para avaliação dos resultados. O CobiT independe das plataformas de TI adotadas nas empresas.

O CobiT é orientado ao negócio. Fornece informações detalhadas para gerenciar processos baseados em objetivos de negócios. O CobiT é projetado para auxiliar três audiências distintas:

- Gerentes que necessitam avaliar o risco e controlar os investimentos de TI em uma organização.
- Usuários que precisam ter garantias de que os serviços de TI que dependem os seus produtos e serviços para os clientes internos e externos estão sendo bem gerenciados.
- Auditores que podem se apoiar nas recomendações do CobiT para avaliar o nível da gestão de TI e aconselhar o controle interno da organização.

O CobiT está dividido em quatro domínios:

- Planejamento e organização.
- Aquisição e implementação.
- Entrega e suporte.
- Monitoração.

Processos do COBIT

Os mapas de controle fornecidos pelo CobiT auxiliam os auditores e gerentes a manter controles suficientes para garantir o acompanhamento das iniciativas de TI e recomendar a implementação de novas práticas, se necessário. O ponto central é o gerenciamento da informação com os recursos de TI para garantir o negócio da organização.

Cada domínio cobre um conjunto de processos para garantir a completa gestão de TI, somando 34 processos:

- **Planejamento e Organização**
    - Define o plano estratégico de TI
    - Define a arquitetura da informação
    - Determina a direção tecnológica
    - Define a organização de TI e seus relacionamentos
    - Gerencia os investimento de TI
    - Gerencia a comunicação das direções de TI
    - Gerencia os recursos humanos
    - Assegura o alinhamento de TI com os requerimentos externos

    - Avalia os riscos
    - Gerencia os projetos
    - Gerencia a qualidade
- **Aquisição e implementação**
    - Identifica as soluções de automação
    - Adquire e mantém os softwares
    - Adquire e mantém a infra-estrutura tecnológica
    - Desenvolve e mantém os procedimentos
    - Instala e certifica softwares
    - Gerencia as mudanças
- **Entrega e suporte**
    - Define e mantém os acordos de níveis de serviços (SLA)
    - Gerencia os serviços de terceiros
    - Gerencia a performance e capacidade do ambiente
    - Assegura a continuidade dos serviços
    - Assegura a segurança dos serviços
    - Identifica e aloca custos
    - Treina os usuários
    - Assiste e aconselha os usuários
    - Gerencia a configuração
    - Gerencia os problemas e incidentes
    - Gerencia os dados
    - Gerencia a infra-estrutura
    - Gerencia as operações
- **Monitoração**
    - Monitora os processos
    - Analisa a adequação dos controles internos
    - Prove auditorias independentes
    - Prove segurança independente
    - Desenvolvimento do CobiT

Ferramentas de Gerenciamento do COBIT

Na era da dependência eletrônica dos negócios e da tecnologia, as organizações devem demonstrar controles crescentes em segurança. Cada organização deve compreender seu próprio desempenho e deve medir seu progresso. O benchmarking com outras organizações deve fazer parte da estratégia da empresa para conseguir a melhor competitividade em TI. As recomendações de gerenciamento do CobiT com orientação no modelo de maturidade em governança auxiliam os gerentes de TI no cumprimento de seus objetivos alinhados com os objetivos da organização.

Os *guidelines* de gerenciamento do CobiT focam na gerência por desempenho usando os princípios do balanced scorecard. Seus indicadores chaves identificam e medem os resultados dos processos, avaliando seu desempenho e alinhamento com os objetivos dos negócios da organização.

Os modelos de maturidade de governança são usados para o controle dos processos de TI e fornecem um método eficiente para classificar o estágio da organização de TI. A governança de TI e seus processos com o objetivo de adicionar valor ao negócio através do balanceamento do risco e returno do investimento podem ser classificados da seguinte forma:

- 0) Inexistente
- 1) Inicial / Ad Hoc
- 2) Repetitivo mas intuitivo
- 3) Processos definidos
- 4) Processos gerenciáveis e medidos
- 5) Processos otimizados

Essa abordagem deriva do modelo de maturidade para desenvolvimento de software, *Capability Maturity Model* for Software (SW-CMM), proposto SEI. A partir desses níveis, foi desenvolvido para cada um dos 34 processos do CobiT um roteiro:

- Onde a organização está hoje
- O atual estágio de desenvolvimento da industria (*best-in-class*)
- O atual estágio dos padrões internacionais
- Aonde a organização quer chegar

Os fatores críticos de sucesso definem os desafios mais importantes ou ações de gerenciamento que devem ser adotadas para colocar sobre controle a gestão de TI. São definidas as ações mais importantes do ponto de vista do que fazer a nível estratégico, técnico, organizacional e de processo.

Os indicadores de objetivos definem como serão mensurados os progressos das ações para atingir os objetivos da organização, usualmente expressos nos seguintes termos:

- Disponibilidade das informações necessárias para suportar as necessidades de negócios
- Riscos de falta de integridade e confidencialidade das informações
- Eficiência nos custos dos processos e operações
- Confirmação de confiabilidade, efetividade e conformidade das informações.

Indicadores de desempenho definem medidas para determinar como os processos de TI estão sendo executados e se eles permitem atingir os objetivos planejados; são os indicadores que definem se os objetivos serão atingidos ou não; são os indicadores que avaliam as boas práticas e habilidades de TI.

Sistemas de Gerenciamento eletrônico de documentos (GED)

O GED ou EDM (*Electronic Document Management*) é a tecnologia que provê um meio fácil de armazenar, localizar e recuperar informações existentes em documentos e dados eletrônicos, durante todo o seu "Ciclo de Vida".

A possibilidade de controlar e gerenciar documentos torna-se hoje uma necessidade estratégica para a competitividade. **Capturar**, **gerenciar**, **armazenar**, **preservar** e **distribuir** documentos eficientemente possibilita às empresas tornarem os seus negócios aptos às rápidas mudanças de mercado, aos mesmo tempo que aumenta a produtividade e reduz custos.

É a tecnologia do GED que torna o eBusiness uma realidade, pois alicerça todas as informações referentes a qualquer etapa de qualquer processo de negócio. Não são simplesmente sistemas de gerenciamento de arquivos. O GED é mais, pois ele implementa categorização de documentos, tabelas de temporalidade, ações de disposição e controla níveis de segurança. É vital para a manutenção das bases de informação e conhecimento das empresas.

Aspectos Técnicos

A duração do ciclo de vida de um documento, bem como cada estágio, depende muito do tipo de documento, dos processos de negócios, dos padrões da indústria, e até mesmo do caráter legal. Portanto, o correto entendimento do ciclo de vida e dos padrões de documentação em uma empresa são fatores críticos para se determinar os requisitos de um sistema de GED.

Quando se analisa sistemas GED, alguns aspectos técnicos importantes devem ser cuidadosamente avaliados:

- **Gerenciamento:** O sistema GED deve ser capaz de gerenciar qualquer formato eletrônico, incluindo diferentes formatos de imagens, textos, documentos compostos, arquivos eletrônicos, áudio digital, vídeos etc.

- **Armazenamento:** Um sistema GED deve suportar, idealmente, tanto o arquivamento off-line, quanto o on-line, em diferentes mídias de armazenamento, desde mídias digitais como discos ópticos, CD-ROM, discos magnéticos, fitas, até mídias tradicionais como microfilme e papel. O sistema deve gerenciar de forma segura o uso da mídia para o armazenamento do documento, migrar documentos entre mídias quando definido pelo ciclo de vida, e fazer uso de cache para acelerar a recuperação dos documentos mais utilizados;
- **Recuperação:** O sistema deve permitir que os documentos armazenados sejam recuperados a partir de um conjunto de atributos previamente definidos (metadados), mas também deve suportar a recuperação através de palavras e frases contidos nos documentos. A pesquisa deve ser realizada de forma unificada, de modo que os documentos sejam acessados, recuperados, e visualizados independente do local de armazenamento;
- **Intercâmbio:** Visando o intercâmbio de documentos entre a empresa e entidades sem acesso ao sistema GED, o mesmo deve permitir o acesso a aplicações de terceiros, permitindo que os documentos sejam extraídos em seu formato nativo, e transportados como desejado;
- **Internet:** Todas as funcionalidades de um sistema GED, devem estar disponíveis via Web.

Evolução do GED

O GED é um conjunto de tecnologias. Ele é formado pelas seguintes tecnologias:

- **Document Imaging (DI):** enfatiza basicamente a digitalização de documentos de origem papel
- **Document Management (DM):** ferramentas para controle de localização, atualização, versões e de temporalidade de guarda dos documentos. É uma das exigências da ISO 9000, e permite a rastreabilidade das alterações.
- **Workflow:** substituí o processo humano de trâmite de documentos em papel, muitos ganhos são obtidos.
- **COLD/ERM (*Computer Output to Laser Disk*):** substituí os microfilmes, redução no custo de armazenamento. Devido a abrangência dessa tecnologia, ela passou a ser chamada de ERM (*Enterprise Report Management*).
- **Forms Processing:** em vez da utilização de digitadores para a retirada das informações, utilizam-se sistemas digitais. Outra tecnologia complementar é o **RIM (*Records and Information Management*)**.

Algumas das aplicações do GED são: apoio aos processos de fiscalização, auditoria, importação e exportação, cartões de assinatura, processos judiciais, disponibilização de diários oficiais, controle de bilhetes, impostos, taxas, multas, extratos, entre muitos outros. O GED também impulsiona outras tecnologias, como o *Knowledge Management*, CRM, ERP e SCM.

Sistemas Integrados de Gestão

ERP - Enterprise Resource Planning

ERP (*Planejamento de Recursos Empresariais*) são sistemas de informações transacionais(OLTP) cuja função é armazenar, processar e organizar as informações geradas nos processos organizacionais agregando e estabelecendo relações de informação entre todas as áreas de uma companhia. É uma plataforma de software desenvolvida para integrar os diversos departamentos de uma empresa, possibilitando a automatização e armazenamento de todas as informações de negócios.

Algumas vantagens e mudanças mais palpáveis que um sistema de ERP propicia a uma corporação são:

- Maior confiabilidade dos dados
- Monitoração em tempo real dos dados
- Diminuição do retrabalho

O passo a passo de um projeto para implantação do ERP é:

- **Raio X:** Esta é a fase do projeto onde os processos e as práticas de negócio são analisados. É o momento em que a companhia é profundamente observada e quando é definida a necessidade de uma solução ERP.
- **Desenvolvimento**: É neste momento que uma aplicação é escolhida e configurada para uma companhia. Também são definidos o modelo de funcionamento da solução e outros aspectos do ambiente.
- **Teste:** Aqui a solução de ERP é colocada em um ambiente de teste. É quando os erros e falhas são identificados.

- **Treinamento**: Todos os profissionais são treinados no sistema para saber como utiliza-lo antes da implementação ser concluída.
- **Implementação**: O software de ERP é finalmente instalado na companhia e se torna funcional aos usuários.
- **Avaliação**: A solução de ERP é avaliada, observando-se o que é necessário melhorar e o que está ou não funcionando adequadamente. Esta é apenas uma avaliação geral do projeto ERP para referências futuras.

Mesmo tendo implantado sistemas de ERP, as empresas ainda têm as seguintes quatro principais dificuldades:

- Quando necessitam de um documento, ele está arquivado em papel com as tradicionais limitações do seu manuseio.
- Os processos são lentos e não têm um "motor" que os movimente e que controle seu andamento, prazos e tarefas dentro da empresa.
- A quantidade de dados gerados pelo ERP aumentou muitas vezes o volume de informações disponível dentro da empresa. Armazenar e consultar essas informações de forma rápida e inteligente tornou-se um fator crítico.
- A quantidade de documentos eletrônicos aumentou. É necessário o controle de versões, histórico de uso, segurança de acesso e, principalmente, um enfoque corporativo de utilização integrado aos processos de negócios do ERP.

A única alternativa para superar essas dificuldades é a utilização das soluções de GED - Gerenciamento Eletrônico de Documentos, integradas ao ERP. É por isso que vários fornecedores de ERP já estabeleceram parcerias com empresas de GED.

Implementar recursos de imagem, Gerenciamento Eletrônico de Documentos, workflow, COLD/ERM e Knowledge Management integrados ao ERP, e via web, é a solução para aumentar a eficiência e produtividade na gestão do seu negócio, gerando os indispensáveis diferenciais de competitividade.

CRM - Customer Relationship Management

CRM é um conceito que implica em mudanças nos processos e na cultura das empresas. Para sua efetivação, utiliza várias tecnologias que objetivam conhecer o cliente e atendê-lo melhor, fazê-lo comprar mais e retê-lo.

CRM é uma estratégia de negócio voltada ao entendimento e antecipação das necessidades dos atuais clientes e daqueles potenciais de uma empresa. Envolve captar todos os seus dados, consolidá-los em um banco de dados, analisá-los para identificar padrões, distribuir resultados para todos os pontos de contato e usar essas informações para interagir com os clientes.

Entre as tecnologias usadas estão *call center*, a integração de sistemas legados e sistemas de suporte à decisão, como o GED.

O CRM, visto como uma aplicação, deve integrar módulos de automação de vendas, gerência de vendas, telemarketing, televendas, atendimento ao consumidor, soluções para informações gerenciais, *web* e comércio eletrônico. A base de dados deve ser alimentada com o apoio da documentação que diz respeito aos clientes. Isso cabe ao COLD, GED, *workflow*, ERP. Essas tecnologias servem para tornar utilizáveis as informações sobre os clientes captadas das mais diferentes formas.

O *workflow* é parte fundamental para a eficácia de um sistema de CRM. Com uma ferramenta de *workflow* ligando o *call center* aos processos de negócio todas as solicitações jamais se perderão. Se não forem atendidas no prazo estipulado, o *workflow* se incumbirá de disparar alarmes para os mais diferentes níveis organizacionais.

Identificar e conhecer o cliente e suas necessidades, além de garantir sua satisfação através de: atendimento personalizado, maior atenção dedicada ou simplesmente o aumento da eficiência nos serviços prestados; são alguns dos objetivos a serem atingidos através do CRM. Podemos distinguir um projeto de CRM de acordo com sua abordagem ou tema, que pode ser:

- **Operacional:** que visa principalmente melhorar o relacionamento direto entre a empresa e o cliente através de canais como a internet ou Call Centers. Tais melhorias são conseguidas agrupando informações antes espalhadas pelos diversos setores da empresa, definindo com maior precisão o perfil do cliente, o que permite que a empresa esteja melhor preparada na hora de se relacionar com o mesmo;
- **Analítico:** que trata da análise das informações obtidas sobre o cliente nas várias esferas da empresa, permitindo descobrir entre outras informações o grau de fidelização dos clientes, seus diferentes tipos, preferências e rejeições quanto a produtos e serviços. A comparação entre um CRM de abordagem analítica com um *Data Mart* para o setor Marketing e/ou vendas é inevitável, pois ambos auxiliam a responder importantes questões de negócio. Mas é importante lembrar da necessidade de se ter bem definidas as estruturas de *datamarts* e *datawarehouse*, antes de começar a construção da parte física ou ferramental de um CRM;
- **Colaborativo:** esta abordagem do CRM procura integrar as estruturas e benefícios dos outros dois temas descritos. Enquanto o CRM operacional está mais focado nos níveis tático e operacional, e o CRM analítico nos níveis estratégico e tático, o CRM colaborativo procura gerar melhorias nos três níveis. A principal característica deste dessa abordagem está na possibilidade de criar, aumentar e gerenciar a interação com o cliente. Para isso é necessário que a

empresa possua um meio adequado para a interação - abordada no CRM operacional - e que possua informações suficientes sobre seus clientes - obtidas através do CRM analítico - de forma centralizada e, é claro, integrada.

Dentro das ferramentas de *Business Intelligence,* o CRM é uma das mais importantes soluções a ser implementada utilizando as plataformas de *datamarts* e do *datawarehouse*.

Automação de processos de trabalho (Workflow)

Fluxo de Trabalho. É a tecnologia que permite gerenciar de forma pró-ativa qualquer processo de negócio das empresas. Garante o acompanhamento constante de todas as atividades e um aumento de produtividade com objetividade e segurança.

O Workflow também atua como um integrador dos mais diversos sistemas e tecnologias: ERP, SCM (*Supply Chain Management*), CRM, eBusiness e outras.

Sistemas de Workflow são programas destinados a suportar processos de negócio:

- controlando a lógica que governa as transições entre as tarefas do processo.
- suportando as tarefas individuais: ativando recursos humanos e recursos necessários para completar a tarefa.

O Workflow melhora a eficiência do negócio, pois dá a possibilidade de balancear as tarefas entre os funcionários da empresa, reduzindo custos (pessoal, papel) e ainda diminui o tempo para o usuário perceber que tem uma atividade.

São quatro os conceitos básicos da tecnologia de Workflow:

- **Lógica do Processo:** é a representação computacional de um conjunto de passos de atividades discretas, associadas com operações humanas e/ou realizadas por computador e as regras que governam a progressão dessas atividades.
- **Relacionamento entre Pessoas e Tarefas:** o sistema de workflow é responsável por atribuir tarefas que precisam ser realizadas para as pessoas necessárias para executá-las.
- **Disponibilização de recursos de informação necessários para executar tarefas:** os dois tipos de recursos podem ser classificados em recursos humanos e recursos computacionais.
- **Gerenciamento de processos:** torna a lógica dos processos explícita, permite coletar e avaliar métricas relativas ao tempo, custo, qualidade da execução do processo e suas tarefas constituintes.

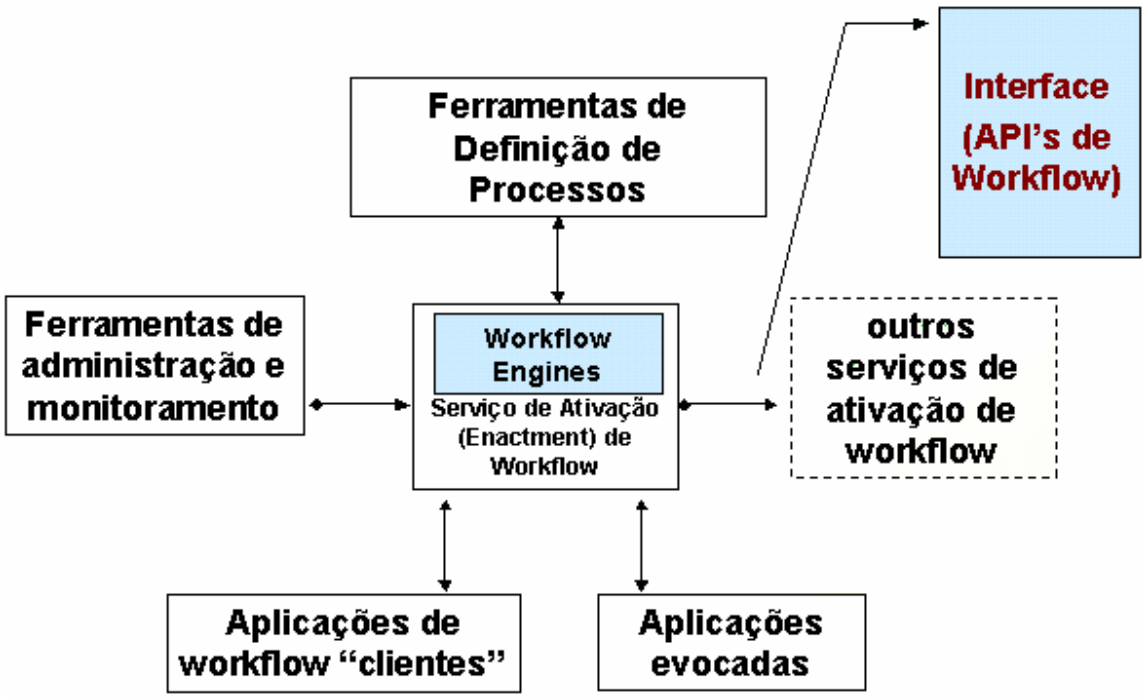
Modelo de Referência

O WfMC (Workflow Management Coalition) é uma organização com mais de 225 membros, localizados em países em todo o mundo. Constituído por fabricantes, universidades, organizações governamentais, e corporações interessadas na tecnologia de workflow. O WfMC define: a Terminologia para elementos de sistemas de workflow, Padrões abertos para interoperação de sistemas, WAPI (Workflow API) e o Modelo de Referência (Workflow Reference Model).

A definição de Workflow segundo o WfMC é a "Automação de uma parte ou de todo um processo de negócio através de recursos computacionais".

Dentro do contexto do Workflow, há o **WFM (*Workflow Management System*)**, que é o sistema que define, gerencia e executa “workflows” através da execução de software cuja ordem de execução é controlada por uma representação computacional da lógica de workflow.

O modelo de referência define que o WFM é composto por cinco subsistemas funcionais, exposto na figura.



Workflow no contexto do ERP

Workflow e ERP, independentes, proporcionam benefícios no aumento da eficiência nos negócios.

Trabalhando juntos possibilitam uma abrangência maior promovendo benefícios para toda a organização.

Quando falamos de workflow estamos nos referindo a um software que automatiza e controla os processos de negócio de uma empresa. Como exemplo de processos de negócios, temos departamento de compras, área de crédito em bancos, sistemas de sinistro em seguradoras e até mesmo os sistemas de reembolso de despesas.

Os sistemas de ERP permitem que as empresas tenham seus sistemas de planejamento e controle juntos. Geralmente uma solução de ERP tem integrada os módulos de finanças, fabricação, distribuição, logística e de recursos humanos.

Numa solução de ERP, tem-se um workflow embutido permitindo agilidade dentro de cada módulo, bem como proporcionando a integração entre todos os módulos. Esses fluxos de workflow são de propriedade exclusiva dos fabricantes de ERP.

Os módulos de ERP implantados estão totalmente integrados entre si, mas muitas vezes tem-se a necessidade de integração com outras áreas da empresa, como aplicações legadas, ERM, CRM, gerenciamento de documentos e outras. Nesses casos, o uso de ferramentas de workflow de terceiros permite a integração do ERP a todas as aplicações existentes na empresa, gerando ganhos muitos maiores. O uso de workflow soluciona uma das grandes dificuldades do ERP

Gerenciamento de processos de negócio (BPM)

É uma forma genérica de descrever um conjunto de serviços e ferramentas que oferecem gerenciamento de processos (análise, definição, execução, monitoração e administração), incluindo suporte a interações tanto humanas como automatizadas. BPM surgiu de diversas origens como soluções de workflow, ferramentas colaborativas, brokers de integração, servidores de aplicação, ferramentas de desenvolvimento, etc.

BPM aproveita as ferramentas para analisar e modelar os processos, utilizando normalmente um modelador gráfico direcionado aos Analistas de negócio que levantam os processos atuais e definem novos processos. Um núcleo de execução executa o processo passo a passo como definido a medida que este fluxo é executado aplicações ou tarefas manuais podem ser invocadas. Além disso, este núcleo mantém a situação dos processos para que enquanto múltiplas instancias dos processos estejam sendo executadas, elas podem ser monitoradas e administradas. Análises posteriores dos processos também são permitidas já que estas informações podem ser armazenadas.

E-commerce

Comércio eletrônico ou *e-commerce* é um tipo de comércio feito especialmente por um equipamento eletrônico como por exemplo um computador. Sistema comercial eletrônico, realizado através da Internet para transações de bens ou serviços entre duas ou mais partes.

A capacidade de comprar e vender produtos e serviços através da Internet. Inclui a exposição de bens e serviços on-line, bem como a colocação de pedidos, faturamento, atendimento e ainda todo o processamento de pagamentos e transações ou seja, usar a Internet, comunicações digitais e aplicativos de TI para possibilitar o processo de compra ou venda. Alguns especialistas definem e-commerce como todas as etapas que ocorrem em qualquer ciclo de negócios usando a tecnologia acima. Outros, como compras feitas por consumidores e empresas pela Internet. Podemos também encontrar a expressão, também válida, E-marketplace que pode ser genérico (abarcar todos os setores de atividade) ou temático (apenas um setor de atividade).

O ato de vender ou comprar pela Internet já é um bom exemplo de comércio eletrônico. O mercado mundial está absorvendo o comércio eletrônico em grande escala, muitos ramos da economia agora são ligadas ao comércio eletrônico. Existem diversas modalidades de comércio eletrônico, entre elas o B2B, B2C, C2C, G2B e G2C.

- **B2B**: Acrônimo de Business to business ou Negócios para negócios. É um modelo de comércio geralmente praticado por fornecedores e clientes empresariais.
- **B2C**: Acrônimo de Business to Consumers. O chamado B2C é o comércio eletrônico efetuado diretamente entre a empresa produtora e o consumidor final. Em geral o consumidor adquire os produtos a preço mais competitivo pois evita o 'atravessador'.
- **C2C**: Acrônimo de Consumer to Consumer Comercio eletrônico que se desenvolve entre usuários particulares, ou seja, entre internautas
- **G2B**: Acrônimo de Government to Business. Relação de negócios pela Internet entre Governo e empresas ex: Compras pelo Estado através da Internet através de pregões e licitações, tomada de preços, etc.
- **G2C**: Acrônimo de Government to Consumers. Relação comercial pela Internet entre Governo e consumidores, ex: pagamento via Internet de Impostos multas e tarifas públicas.

Podemos encontrar também as expressões: Empresas e empresas (B-B) , Empresas e consumidores (B-C) , Consumidores e consumidores (C-C), Governo e consumidores (G-C), Governo e empresas (G-B) ao invés de B2B, B2C, etc.

E-business (*Electronic Business*), é o termo que se utiliza para identificar os negócios efetuados por meios eletrônicos, geralmente na Internet. E-business consiste na utilização da Web para ajudar as empresas a simplificarem os seus processos, aumentarem a sua produtividade e melhorar a sua eficiência. Permite que as empresas se comuniquem com facilidade com parceiros, fornecedores e clientes, que se conectem com sistemas de dados de back-end e que realizem transações de maneira segura. Observe que o e-commerce é apenas uma parte do processo mais amplo do e-business que pode abranger projeto, produção, interação de internautas no desenvolvimento de algo que, no futuro poderá utilizar o e-commerce para venda.

Técnicas e Linguagens de Programação

Lógica

A lógica é o estudo de argumentos. Um argumento é uma seqüência de enunciados, na qual um dos enunciados é a conclusão, derivado a partir dos outros enunciados, chamados premissas.

Lógica Formal

A Lógica Formal, também chamada de Lógica Simbólica, se preocupa basicamente com a estrutura do raciocínio. Ela lida com a relação entre conceitos rigorosamente definidos e fornece um meio de compor provas de declarações. As sentenças são transformadas em notações simbólicas precisas, compactas e não ambíguas.

Operador	Símbolo
NÃO (Negação)	$\neg$
AND (Conjunção)	$\wedge$
OR (Disjunção)	$\vee$
se ... então	$\rightarrow$
se e somente se	$\leftrightarrow$

Enunciados Condicionais e Implicações Materiais

Formas de enunciado, de raciocínio e de argumentos

Algoritmos e Estruturas de Dados

Um algoritmo é um processo discreto e determinístico, que termina quaisquer que sejam os dados iniciais. Isso quer dizer que um algoritmo possui uma seqüência de ações indivisíveis, e que para cada passo da seqüência e para cada conjunto válido de dados, corresponde uma e somente uma ação.

Um algoritmo é constituído por um conjunto de expressões simbólicas que representam ações (atribuir, escolher, etc.), testes de condições (verdadeiro, falso) e estruturas de controle (ciclos e saltos na seqüência do algoritmo), de modo a especificar o problema e sua respectiva solução.

Noções de complexidade de algoritmo

Algoritmos podem ser classificados através de quatro critérios:

- **Completeza:** Se o algoritmo sempre encontra uma solução, se esta existir.
- **Otimalidade:** Se o algoritmo garante encontrar a melhor solução.
- **Complexidade de Espaço:** A quantidade de memória necessária para a execução do algoritmo.
- **Complexidade de Tempo:** Quanto tempo o algoritmo demora para rodar.

Quando se faz um algoritmo para resolver determinado problema, não basta que o algoritmo esteja correto. É importante que ele possa ser executado em um tempo razoável e dentro das restrições de memória existentes. Além disso, ele deve permanecer viável, à medida que o tempo passa, quando a quantidade de dados envolvida normalmente cresce. Os parâmetros de complexidade estudados normalmente são os seguintes:

- **Complexidade de pior caso:** caracterização do tempo de execução máximo, para determinado tamanho da entrada, bem como das características da entrada que levam a esse tempo máximo. Este é o principal parâmetro para se avaliar um algoritmo.
- **Complexidade de caso médio:** caracterização do tempo de execução médio do algoritmo, para determinado tamanho da entrada, considerando a média de todas as possibilidades. Em muitas situações este parâmetro é útil.
- **Complexidade de melhor caso:** caracterização do tempo de execução mínimo, para determinado tamanho da entrada, bem como das características da entrada que levam a esse tempo mínimo.

A determinação da complexidade teria que ser feita contando-se todas as instruções executadas e o tempo de execução de cada delas, considerando-se determinada entrada. Normalmente isso não é viável. O que se faz é determinar um limite superior para esse tempo, o mais próximo possível da realidade. Para tanto, fixa-se o estudo na instrução mais executada do algoritmo e determina-se uma função t(n), que dá a variação do tempo de execução em função de n, o tamanho da entrada.

Tipos de dados

Existem várias classificações para os tipos de dados utilizados por uma linguagem de programação:

- **Primitivos:** normalmente, linguagens de programação possuem tipos de dados numéricos inteiros e reais e tipos caracter (uma letra, número ou símbolo). Outros tipos, como o tipo de dados booleano (boolean) podem ser utilizados para guardar os valores verdadeiro (*true*) ou falso (*false*).
- **Derivados:** São definidos a partir dos tipos de dados primitivos. Exemplos são os vetores, matrizes e cadeia de caracteres (*Strings*), classificados como **homogêneos**, e registros e uniões, classificados como **heterogêneos,** pois agrupam dados primitivos de tipos diferentes..

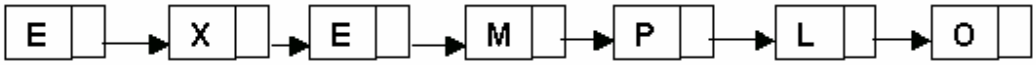
Tipo **abstrato de dado** é um tipo de dado que esconde a sua implementação de quem o manipula, de maneira que determinadas operações sobre estes dados podem ser executadas sem que se saiba como isso é feito. Tipos abstratos de dados abrangem dados e operações sobre estes dados. Exemplos: objetos, listas, árvores binárias...

Listas Encadeadas

Listas Lineares são um conjunto de **n** nós referentes a **n** objetos, onde as propriedades estruturais decorrem unicamente das posições relativas dos nós, dentro de uma seqüência linear. Essas estruturas são tratadas sob o ponto de vista de como é a alocação de memória para os dados: **Alocação Sequencial** e **Alocação Encadeada**.

Vetores são as estruturas mais conhecidas de alocação seqüencial. Mas as listas implementadas através de vetores podem levar a uma pobre alocação de memória. Uma alternativa é utilizar a alocação encadeada.

Na alocação encadeada (ou dinâmica) os nós logicamente vizinhos de uma lista não são mais necessariamente vizinhos em termos de memória. Os nós podem estar aleatoriamente dispostos e a indicação de vizinhança é feita através de um campo de endereço (chamado ponteiro). Desta forma, o nó da lista contém pelo menos dois campos: a chave e o ponteiro.



O último nó contém um ponteiro nulo, que pode receber várias denominações: **λ**, **Null**, **Nil** ou **Nulo**.

Em um nó pode existir mais de um ponteiro. As listas onde existe apenas um ponteiro são denominadas Listas Simplesmente Encadeadas. Qualquer estrutura encadeada necessita de um apontador para o início da lista.

Uma **lista circular encadeada** é uma lista com cabeça, onde o último elemento, ao invés de apontar para Nil, aponta para o nó cabeça. Uma **lista duplamente encadeada** é uma lista encadeada nos dois sentidos. Cada nó, então tem dois links, um para a frente (*prox*) e outro para trás (*ante*).

Pilhas

Uma pilha (*stack*) é uma estrutura de dados linear, em que o último elemento a ser inserido é o primeiro a ser retornado pela estrutura. Funciona como uma pilha de pratos em um restaurante. Ao serem limpos, os pratos são empilhados (armazenados). Quando alguém precisa usar um prato, o último a ser empilhado será o escolhido. Este é o princípio LIFO (*Last In First Out*).

Uma pilha pode ser implementada por um vetor, onde o início da pilha é o primeiro elemento do vetor, e o topo é a quantidade de itens inseridos no vetor (começa com 0).

As duas principais operações de uma pilha são:

- **Push (empilhar):** é a operação de adicionar um elemento ao topo da pilha, aumentando o seu tamanho em uma unidade. Caso a pilha esteja cheia (sem espaço para novos elementos), um erro de *overflow* ocorre. Esta operação não retorna nada, e o elemento a ser inserido é passado como parâmetro.
- **Pop (desempilhar):** é a operação de retirar um elemento do topo da pilha, diminuindo o seu tamanho em uma unidade. Caso a pilha esteja vazia, um erro de *underflow* ocorre. Esta operação retorna o elemento da pilha, e não recebe argumentos como parâmetro.

Estas duas operações são suficientes para realizar qualquer alteração na pilha. Por exemplo, para inverter a ordem dos últimos dois elementos, basta a seguinte seqüência de instruções:

```
a = pop ()
b = pop ()
push (a)
push (b)
```

Filas

Uma fila (*queue*) é uma estrutura de dados linear, em que o primeiro elemento inserido é o primeiro a ser retornado pela estrutura. Funciona como filas de espera. Os elementos vão sendo colocados na lista e retirados (ou processados) por ordem de chegada. Este é o princípio FIFO (*First In First Out*).

Uma fila pode ser implementada por um vetor, utilizando dois apontadores, um para o início da fila, e outro para o seu final. Duas operações devem ser implementadas: **enfila** e **desenfila**.

Vetores e Matrizes

Vetores guardam um número fixo de elementos, normalmente do mesmo tipo. Elementos são acessados utilizando um índice, inteiro, que referencia uma posição seqüencial do vetor. Nesta seção serão estudadas as vantagens, desvantagens e aplicações desta estrutura durante a programação.

Vetores permitem o uso eficiente da memória, pois são armazenados seqüencialmente na memória, não precisando de quase nenhum (ou até nenhum) espaço para controle da estrutura. Assim, um vetor com 1000 inteiros utilizará apenas 1000 vezes o tamanho necessário para armazenar um único inteiro.

Uma das grandes vantagens no uso de vetores, em linguagens de programação, é o uso de apenas um identificador para referenciar vários elementos. Suponha que uma aplicação possui o nome dos funcionários armazenados nas seguintes variáveis: Nome1, Nome2, Nome3, Nome4 e Nome5. Não seria nada prático manter um código desta maneira, pois a cada atribuição ou acesso a uma destas variáveis, ela deve-ria ser referenciada explicitamente (por exemplo, Nome1 = Nome5).

Utilizando vetores, basta declarar um vetor de 5 posições, com o identificador Nome, e cada funcionário teria o seu nome armazenado em uma das posições deste vetor.

Outro benefício no uso de vetores, é a propriedade de localidade de referência. Arquiteturas de computadores modernas utilizam técnicas de paginação de memória (será visto com mais detalhes no capítulo de Sistemas Operacionais) que são beneficiadas pelo armazenamento seqüencial de dados, pois o acesso a uma grande quantidade de dados vizinhos em um vetor fará menos acessos à páginas do que na situação de cada elemento do vetor estar espalhado em posições distintas da memória.

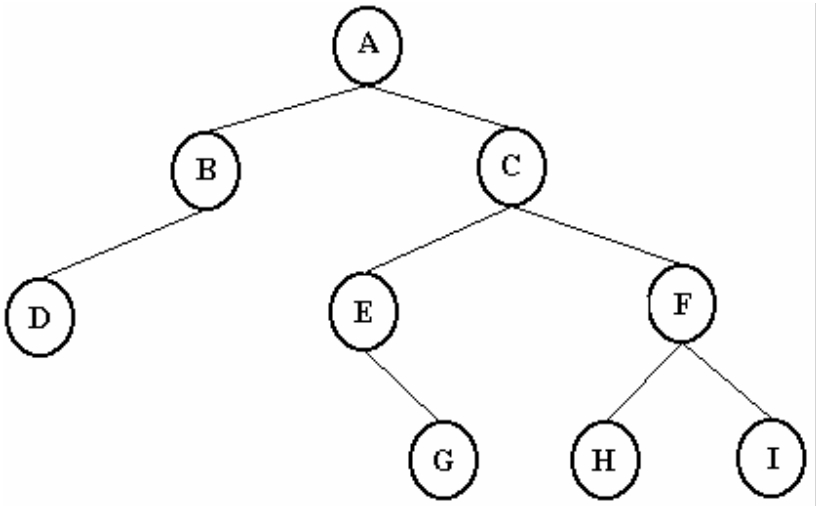
A maior desvantagem dos vetores é possuir um tamanho fixo. Embora em muitos ambientes seja possível alterar o limite superior do vetor (vetores dinâmicos), esta operação não é trivial. Uma possibilidade é fazer o vetor crescer não apenas uma posição, mas sim várias posições sempre que o limite for atingido. Desta forma, ao inserir os próximos elementos não será necessário expandir o vetor novamente.

Árvores balanceadas

Muitas vezes precisa-se de estruturas mais complexas do que as seqüenciais, especialmente quando se trabalha com grandes volumes de dados. Uma das classes de estruturas desse tipo são as árvores, que servem para representar estruturas hierarquizadas, onde um elemento pode apontar para vários outros.

Uma árvore enraizada T, ou simplesmente árvore, é um conjunto finito de elementos denominados de nós ou vértices, tais que: T é um conjunto vazio e então a árvore é dita vazia OU existe um nó especial r, chamado raiz de T; os restantes constituem um único conjunto vazio ou são divididos em m >= 1 conjuntos disjuntos não vazios (subárvores de r), ou simplesmente subárvores, cada qual por sua vez uma árvore

Árvores de busca são famílias de árvores utilizadas para armazenamento e busca de dados. O objetivo dessas estruturas é obter uma performance média bem melhor que aquela obtida com listas seqüenciais. O mecanismo básico utilizado nas árvores de busca é o de, numa procura, comparar o argumento de busca com determinada chave de determinado nó da árvore. Se a chave for maior, a busca prossegue pela subárvore da direita; se for menor, pela subárvore da esquerda



Árvore Binária de Busca (ABB) é uma árvore binária onde as chaves dos nós da subárvore direita de cada nó são maiores que as chaves desse nó e as chaves da subárvore esquerda são menores que a mesma.

São ideais para buscas e inserções, principalmente se não for uma árvore completa. Em compensação, operações de deleção são complexas, pois deve-se manter a estrutura da árvore.

A complexidade de busca e inserção é de $O(\log_2 n)$. O número de nós de uma árvore deste tipo varia entre $\log_2 n + 1$ e n, dependendo do seu balanceamento.

Alguns exemplos de árvores ABB são as árvores AVL e as Árvores B, utilizadas para se ter árvores balanceadas em disco e como base para sistemas gerenciadores de banco de dados.

Percorrer uma árvore visitando cada nó uma única vez gera uma seqüência linear de nós, e então passa a ter sentido falar em sucessor e predecessor de um nó segundo um determinado percurso. Há três maneiras recursivas de se percorrer árvores binárias. São elas:

Travessia em Pré-Ordem (ABDCEGFHI)

1. se árvore vazia; fim
2. visitar o nó raiz
3. percorrer em pré-ordem a subárvore esquerda
4. percorrer em pré-ordem a subárvore direita

Travessia em In-Ordem (DBAEGCHFI)

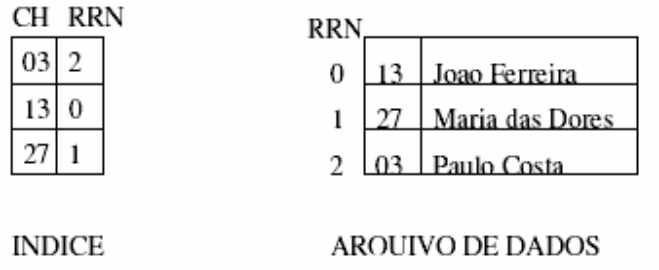
1. se árvore vazia, fim
2. percorrer em in-ordem a subárvore esquerda
3. visitar o nó raiz
4. percorrer em in-ordem a subárvore direita

Travessia em Pós-Ordem (DBGEHIFCA)

1. se árvore vazia, fim
2. percorrer em Pós-Ordem a subárvore esquerda
3. percorrer em Pós-Ordem a subárvore direita
4. visitar o nó raiz

Listas invertidas e Índices

Um índice é uma estrutura de dados contendo dois campos: uma chave e uma referência. A chave está associada a um campo (ou combinação de campos) do registro de um arquivo de dados. Um índice é dito primário quando a chave identifica um único registro apenas no arquivo e secundário no caso de múltiplos registros.



RRN significa *Relative Record Number*. Uma vantagem da utilização de índices é que o seu tamanho é bem menor, e pode ser mantido ordenado para acelerar a busca, ou até mesmo ficar em memória.

Para arquivos de tamanho variável, ao invés de guardar o número do registro, é melhor guardar a sua posição em bytes.

Métodos de acesso

Existem três maneiras básicas de acessar um registro:

- **Acesso seqüencial:** o arquivo é lido registro por registro até o registro desejado ser encontrado. Essa é a forma menos eficiente de acesso, mas algumas situações favorecem o seu uso, como a busca por texto (comando **grep**) ou quando um arquivo possui poucos registros.
- **Acesso direto:** o acesso direto a um dado registro (ou grupo de registros) em disco requer conhecer o endereço do registro no arquivo de dados. Assim, comandos como **fseek**, podem ser usados para localizar diretamente o registro para leitura/gravação, criando um índice.
- **Endereçamento:** o endereço de um registro é o seu deslocamento em bytes (*offset*) desde o início do arquivo (ou após o cabeçalho). No caso de registros de tamanho fixo (como *arrays*), basta armazenar o número de registros que antecedem cada registro, ou seja, a sua posição no arquivo.

Métodos de ordenação e pesquisa

O problema de ordenação de dados pode ser dado como: " Dada uma lista contendo n chaves, distintas ou não, encontrar uma permutação dessas chaves tal que elas fiquem em ordem crescente (ou decrescente) na lista".

Há várias idéias diferentes para ordenação de dados e a maioria delas usa vetores como estrutura de dados:

- **Seleção:** Criar, passo a passo, uma lista ordenada a partir de um conjunto de dados, retirando do conjunto a cada passo o menor elemento presente, até esvaziá-lo. Para implementar essa idéia num vetor, ele é dividido em dois subvetores, o primeiro inicialmente nulo, tal que se retire dados do segundo e transfira para o primeiro. A complexidade do melhor caso é igual a do pior caso que é igual ao caso médio: $O(n^2)$
- **Inserção:** Criar, passo a passo, uma lista ordenada a partir de um conjunto de dados, retirando do conjunto, a cada passo, o primeiro elemento encontrado. Para implementar essa idéia num vetor, ele é dividido em dois subvetores. O primeiro inicialmente contém uma chave, sendo a transferência de dados feita “in place”, onde a cada passo o elemento inserido pode ter que deslocar para a direita elementos maiores. A complexidade é $O(n^2)$, sendo O(n) para o vetor ordenado.
- **Bubblesort:** Também conhecido como o método da bolha, o vetor é percorrido, e testam-se as posições i e i+1. Caso i seja maior que i+1, esses elementos são trocados de posição. A varredura continua até o fim do vetor, e deve recomeçar N vezes (onde N é o tamanho do vetor), para garantir que todos elementos foram ordenados em suas posições. A complexidade é $O(n^2)$.
- **Quicksort:** É considerado o melhor algoritmo de ordenação e foi um dos primeiros algoritmos recursivos propostos. A idéia é fazer, sucessivamente, partições em subvetores, de forma que a parte esquerda contenha sempre elementos menores ou iguais aos da direita. O problema simples é quando o tamanho de uma partição é 1.

Hashing

Hashing, ou mais apropriadamente Transformação Chave-Endereço, é um conjunto de métodos que busca diminuir o tempo médio de busca em tabelas. Está baseado na idéia de se trabalhar com um vetor e calcular o índice de busca através de uma função sobre a chave (função *hash*), que, idealmente, permitiria encontrar a chave em uma tentativa.

A situação ideal corresponderia a se ter funções *hash* injetoras. Entretanto, funções injetoras são relativamente raras. Além disso, o universo de chaves normalmente é extremamente maior que o número de endereços. Isso leva a que se trabalhe com a possibilidade de coincidência de endereços. Nesse caso, as chaves são chamadas de sinônimos. O uso de *hashing*, conseqüentemente, exige que se tenham métodos de tratamento de sinônimos, o que dá origem a várias alternativas. Outro ponto a ser notado é que, a aplicação da função *hash* sobre uma chave pode ser vista como um processo de redução dessa chave

Alguns critérios que devem ser utilizados para um *hashing* eficiente são:

- A imagem da função deve ser o intervalo 1- M (ou 0 - (M-1)).
- Ela deve ser de cálculo rápido.
- A distribuição de sinônimos deve ser uniforme. Isso quer dizer que os endereços devem ter probabilidades iguais (ou muito próximas) de ocupação.
- Deve-se ter cuidado para a função *hash* não preservar propriedades específicas das chaves.

Algumas funções úteis: **módulo da divisão** (com M primo) e **XOR** (nas duas metades da chave).

Para tratar sinônimos, são utilizados os métodos do Encadeamento Interior ou Exterior (uma lista encadeada para cada entrada do vetor), e Endereçamento Aberto (onde não se utilizam *links*, mas os sinônimos são colocados na primeira posição disponível a partir do endereço calculado).

Programação

Programação Estruturada

Imagine que tenha que processar as vendas feitas por cada vendedor de cada filial de uma empresa. Se você vai programar em um ambiente procedural, como COBOL, então vai criar algo assim:

- Um módulo de controle, que invocará os procedimentos de inicialização e o loop principal.
- Um módulo que é invocado para ler e processar cada filial.
- Uma sob-rotina que é invocada do módulo das filiais para poder tratar cada funcionário de cada filial.

Problemas decorrentes:

- 1) Falta de isolamento entre os módulos: alteração em um módulo pode resultar em danos nos outros.
- 2) Dificuldade de manutenção. Imagine o problema que seria se alguns tipos de vendedores tivessem regras diferentes... e se algumas filiais tivessem o mesmo problema?
- 3) Dificuldade de reaproveitamento de código. Se outro programa quisesse utilizar a mesma estrutura haveria “copy-paste” e customização.

A “unidade” de um programa procedural é a sub-rotina, ou parágrafo em COBOL. Para cada coisa é criada uma sub-rotina que processará os dados. Modularizar um programa é criar uma rotina para cada função.

Modularização

A modularização é a decomposição de um conjunto de componentes de software em sub-partes, denominadas módulos. Espera-se que os módulos tenham uma forte coesão interna e um pequeno acoplamento exterior;:

- **Alta Coesão:** todas as partes de um processo (ou módulo) são fortemente relacionadas.
- **Baixo Acoplamento:** o número de interfaces entre os processos (ou módulos) é mantido ao mínimo, visando facilitar modificações futuras.

Um bom módulo deve cumprir uma única função, e todos os seus arquivos devem ser suficientes para cumprir esta função. Além disso, ele também deve possuir uma interface simples e bem especificada.

Sub-rotinas

Uma sub-rotina é uma rotina de código que é chamada pelo programa principal para realizar uma tarefa. Normalmente chama-se de procedimentos as rotinas que não retornam valores, e funções as que retornam.

- **Passagem por valor:** os valores que uma função passa para outra não são alterados pela função chamada, pois as variáveis do programa chamador são copiadas para as correspondentes variáveis da função. A passagem por valor deve ser preferida ao invés do uso de variáveis globais.
- **Passagem por referência:** significa que passamos de uma função para outra o endereço de uma variável (sua localização de memória). Qualquer alteração no conteúdo apontado pelo ponteiro será uma alteração no conteúdo da variável original.
- **Passagem por endereço:** antigamente, os termos passagem por endereço e por referência eram sinônimos. Em C só existe a passagem por valor e por referência, mas em Algol e C++ as três estão disponíveis. Um vetor em C++ é passado por endereço, assim todo o seu conteúdo pode ser manipulado .

Escopo de Variáveis

O escopo de uma variável é a sua área de atuação, normalmente o lugar onde foi definida. Para que não existam ambigüidades na utilização dos identificadores, não pode existir nenhum caso onde existam dois identificadores com o mesmo nome dentro do mesmo escopo, i.e. dentro de um escopo existe uma unicidade de nomes. O mesmo vale para parâmetros passados para uma sub-rotina.

- **Escopo global:** são declaradas fora de todas as sub-rotinas, normalmente no início de módulos. Todo o módulo e as sub-rotinas declaradas em seu corpo possuem acesso a estas variáveis.
- **Escopo local:** são declaradas dentro de uma sub-rotina. Apenas a função possui acesso ao seu conteúdo, o que aumenta o reuso do subprograma e evita efeitos colaterais (maior controle).

Tipos de dados

De forma geral, podemos classificar as linguagens de programação quanto ao modelo de tipagem da seguinte maneira: tipagem forte e tipagem fraca. Essa questão manifesta-se principalmente em atribuições.

No caso de linguagens orientadas a objetos, também ocorre em invocações de métodos.

- **Tipagem forte:** o compilador garante que em tempo de execução ocorra a compatibilidade entre os elementos de uma atribuição, e também a existência do método para uma dada referência. Garante maior robustez aos sistemas desenvolvidos, minimizando erros em tempo de execução.
- **Tipagem fraca:** neste caso, o compilador não existe essa garantia durante a verificação de tipos, sendo responsabilidade do programador. Possibilita maior velocidade e facilidade no desenvolvimento.

Programação Orientada a Objetos

A principal tecnologia que facilitou a criação dos programas Windows foi a programação baseada em objetos ou OOP (*object-oriented programming*). Esta tecnologia tornou possível criar componentes reutilizáveis que se tornaram os blocos da construção dos programas. Os principais elementos da OOP que se usará são objetos reutilizáveis. O Delphi, o Visual Basic e o C++ usam objetos visuais denominados componentes para criar programas. Os componentes que serão usados na construção dos programas são objetos que têm propriedades e métodos, e que respondem a eventos, como controlar a aparência e o comportamento de um componente através de suas propriedades. Estes componentes possuem métodos predefinidos que facilitam a programação.

Alguns conceitos de análise e programação orientados a objetos são:

- **Abstração:** consiste em considerar apenas os aspectos essenciais de uma entidade, ignorando aqueles tidos por irrelevantes. É uma técnica importante para lidar com a complexidade de sistemas.

- **Herança:** habilidade de passar Métodos e Propriedades para as classes derivadas, promovendo, assim a reutilização de código.
- **Encapsulamento:** oculta a implementação das classes, expondo apenas sua interface. Ele separa o comportamento externo de um objeto do seu funcionamento interno, evitando interdependências.
- **Polimorfismo:** habilidade de ter métodos com o mesmo nome executando de maneira semelhante em Classes Derivadas. Ou seja, classes semelhantes, com métodos aparentemente idênticos, podem funcionar de maneira diferente. Assim a mesma operação pode ter comportamentos distintos.
- **Agregação:** também chamada de composição, representa uma forma especial de associação (que define-se quando um objeto recorre aos serviços de um outro), e ocorre quando um objeto contém, física ou logicamente, outros. Lembre-se que o agregado é mais do que a simples soma das suas partes.

Refactoring

Refactoring ou refatoração é a idéia de fazer várias pequenas alterações incrementais em um código, sem adicionar novas funcionalidades ou corrigir erros, mas sim melhorar o projeto ou facilidade de leitura do código.

Esta prática é uma oportunidade de melhorar a qualidade do código

Desenvolvimento J2EE

Especificação J2EE

A plataforma *Java 2 Enterprise Edition* define o padrão para desenvolver aplicações corporativas de várias camadas baseadas em componentes. Algumas das características dessas aplicações: portabilidade, escalabilidade e integração facilitada com aplicações legadas e dados. Além disso, também permite a construção e utilização de *web services* que podem se comunicar com *web services* de outras plataformas (por exemplo, .NET).

Algumas das principais tecnologias incluídas na especificação J2EE são:

- **JavaServer Pages:** páginas JSP são templates usados para produção automática de servlets. Escreve-se uma página HTML com comandos Java, que são traduzidos em tempo de execução em um *servlet*.
- **Java Servlets:** servlets são classes/objetos que funcionam como programas CGI. Com o suporte de um Web container um *servlet* tem acesso a dados de formulários e tem o compromisso de produzir um código HTML que será devolvido ao cliente.
- **Enterprise JavaBeans:** é um objeto que colocado em container adequado pode ser chamado e chamar objetos remotamente de forma (quase) transparente.
- **Java Message Service (JMS):** é um conjunto de APIs que permite acessar de forma padronizada serviços de mensagens. Dispõe de mecanismos para criar, enviar, receber e ler mensagens.
- **Java Naming and Directory Interface (JNDI):** é o elemento utilizado para cadastramento de responsáveis por serviços Java, e isso lhe dá a capacidade de responder a perguntas de clientes por esses serviços. Uma analogia é o mapeamento de nomes de sites para IPs (DNS faz o mapeamento).
- **Java Transaction API (JTA):** especifica uma interface padrão para aplicações utilizarem um serviço de suporte transacional. Esse gerenciamento é mais conhecido no contexto de banco de dados, mas o seu uso em aplicações distribuídas é mais amplo (falhas na rede, dependências, etc).
- **JDBC data access API:** um conjunto de APIs que direta ou indiretamente acessam bancos de dados. Programar em JDBC é escrever, explicitamente, chamadas a métodos cujos parâmetros são, grosseiramente, os comandos SQL que devem ser enviados ao SGBD.

Conceito de servidor de aplicação

Um servidor de aplicação é algo mais que um *servlet container*, que é algo mais que um *webserver*.

O *Servlet container* mais conhecido é o Apache Tomcat. Ele implementa as funções de um Web Server (como o apache httpd), ou seja, apresenta páginas HTML estáticas, mais a funcionalidade de interpretar on-the-fly programas java e apresentar sua saída pelo protocolo http. Ele apresenta páginas dinâmicas JSP e saídas de programas em java chamados de "servlets".

Um Servidor de Aplicação é um *Servlet Container* e mais vários serviços rodando junto. Acontece que o padrão J2EE - Java 2 Enterprise Edition - prevê uma série de funcionalidades a mais de servidor: um servidor de nomes de Java, um distribuidor automático de componentes, um balanceador de carga, entre outros. Bom, esses serviços, juntos, permitem que os *servlets* e JSP's tenham bem mais recursos, conexão mais escalável com um banco de dados e daí por diante. Os EJB's (*Enterprise Java Beans*) nada mais são do que componentes que utilizam de todos esses serviços pra comunicação com o Banco de Dados.

Existem alguns *Applications servers* livres, mas o único realmente sério e com funcionalidade comparável a servidores de aplicação comerciais (como o Sun One ou o IBM WebSphere) é o JBOSS.

Note que a expressão "Servidor de Aplicação" é mais ampla do que essa definição, pois de certo modo ela foi 'apropriada' pelo Java pra designar um Servidor de Aplicação J2EE. Existem na verdade outros esquemas completamente diferentes, em outras linguagens, para fazer "servidores de aplicação".

O conceito de J2EE foi agrupar as diversas tecnologias (JNDI, JCA, JSP, Servlets, EJB, JMS) em uma forma única e consistente. O *Application Server* é um programa que implementa essa J2EE. A Sun possui um programa de certificação para demais fornecedores.

A desvantagem de utilizar um Servidor de Aplicação, é que algumas empresas compram o pacote inteiro, para aplicações simples com JSP e Servlets. As demais tecnologias são desperdiçadas, e bastaria utilizar um simples *container* Web, como o *Tomcat*.

Container web e EJB

Toda aplicação Java executa sob um certo ambiente, o qual se convencionou chamar de container. Por enquanto, imagine um container como uma JVM, um Class Loader (capaz de buscar classes no sistema de arquivo ou na rede e agregá-lo ao ambiente). Quando se executa uma aplicação na linha de comando, o container se liga apenas ao sistema operacional nativo (*Application Container*) e apenas com esse interage.

Applet Container: Por exemplo, quando um *applet* é executado, a JVM funciona como um plugin do web browser, e para tanto, existe um conjunto de regras que permite ao browser interagir com a JVM para executar código Java dentro de seu ambiente. Aqui temos o browser como o Container (*Applet Container*).

Web Container: Uma outra possibilidade é o container Web (*Web Server*), que encapsula as APIs de *Servlets* e JSPs, cujo compromisso é produzir código HTML para envio ao cliente. Um servidor Web é um modelo inteiramente novo, introduzido pela Internet, e geralmente consiste em uma grande quantidade de clientes, que se comunicam com servidores pesados, responsáveis por atender a milhares de requisições. O container é a camada onde será hospedada a aplicação Java (no caso, no servidor web).

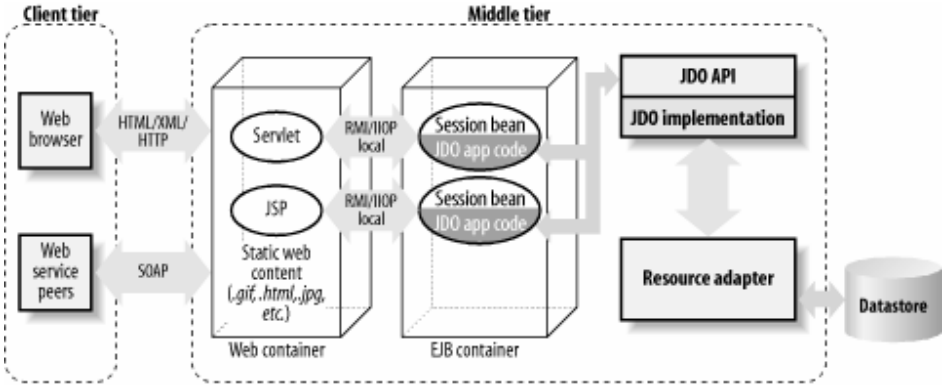
A comunicação entre eles é feita utilizando o protocolo HTTP, normalmente através do envio de requisições de um navegador (ou *browser*) para um servidor Web, que retorna as respostas através de conteúdo Web (como páginas) de volta para o cliente. Então a definição de um servidor Web é a seguinte: é um aplicativo capaz de fornecer ao cliente (navegador Web) os dados solicitados através de suas requisições.

Como exemplo de servidor Web está o Apache, que é um aplicativo livre.

EJB Container: EJBs são o coração da plataforma J2EE. Para caracterizá-los, são necessários: uma interface *home*, que oferece métodos para criar e encontrar instâncias EJB; uma interface *remote*, que é a visão que um cliente tem do objeto; e uma classe *bean*, que é a sua implementação no lado do servidor.

Do ponto de vista de arquitetura um container oferece muitos serviços e propriedades aos EJBs, o que permite caracterizá-los em várias categorias: Entity Beans, Session Beans ou Message-Driven Beans.

Exemplos de container EJB são o Websphere e o JBOSS.



Padrões e anti-padrões de projeto J2EE

Os padrões J2EE foram catalogados pela Sun para resolver problemas que já ocorreram no passado em aplicações baseadas na arquitetura J2EE

- **Intercepting Filter:** realiza o pré-processamento de uma requisição ou pós-processamento de uma resposta. O filtro intercepta as requisições feitas pelo cliente, podendo modificá-la ou redirecioná-la antes de enviá-la para a aplicação. O mesmo pode ser feito com as respostas antes de serem enviadas. Ele aumenta a reusabilidade e desacoplamento do componente e facilita a sua configuração.
- **Transfer Object:** permite que uma classe faça a comunicação de dados entre componentes, agrupando as suas informações em um conjunto único (serializável). Normalmente esses componentes são *Enterprise Java Beans*, que fazem chamadas a vários métodos *get* e *set*.
- **Business Delegate:** utilizado para desacoplar componentes de negócios de aplicações que utilizam os componentes. O *Business Delegate* é um objeto do lado do cliente que esconde do código do cliente os serviços da camada de negócio. Também é usado para tratar as exceções do serviço.
- **Service Locator:** geralmente utilizado em conjunto com o *Business Delegate*, é um componente para lidar com a busca de serviços, ocultando os detalhes do cliente (caso o serviço seja substituído). Reduz a complexidade do código no cliente e melhora a performance e manutenibilidade do código.
- **Front Controller:** especifica um componente central para processar as requisições dos clientes. A responsabilidade deste componente é receber as requisições e roteá-las de forma apropriada. Além de centralizar o controle dos serviços, reduz a quantidade de código e aumenta a segurança da aplicação.
- **Outros padrões:** Alguns outros padrões são: Session Façade, Fast Lane Reader, Composite Entity e Value List Handler.

Padrão MVC de Projeto

O padrão *Model-View-Controller* (MVC) separa as camadas de apresentação, de lógica de negócio e de gerenciamento do fluxo da aplicação, aumentando a reusabilidade e manutenibilidade do projeto.

É muito utilizado para casos em que podem existir múltiplas camadas de apresentação para clientes diversos. Por exemplo, para os usuários diretos de uma aplicação, os dados podem ser apresentados em páginas JSP, mas para os fornecedores pode existir uma interface WML, enquanto para os usuários remo-tos pode existir uma interface HTML simples.

É um dos padrões de desenvolvimento mais populares, favorecendo uma arquitetura em três camadas. O *Servlet* representa o *Controller* (Controlador), enquanto o JSP é a *View* (Visão) e *JavaBeans* são utilizados para mapear o *Model* (Modelo). O *servlet* recebe uma requisição de um cliente (navegador), instancia um *JavaBean* e encaminha a requisição para o JSP. Após esta operação, o JSP envia a resposta de volta para o cliente.

Algumas das características do padrão MVC são:

- Permite que os dados sejam disponibilizados de maneiras diferentes
- Encapsula as funções da aplicação, aumentando a manutenibilidade
- Desacopla a funcionalidade da aplicação da sua apresentação

- Facilita a distribuição da aplicação
- Permite maior flexibilidade através de módulos "plugáveis".
- Distribui as responsabilidades da aplicação de forma consistente

Desenvolvimento de sistemas Web

HTML

HTML é a sigla para *Hyper Text Markup Language*, ou Linguagem de Marcação de Hipertexto, tradicionalmente utilizada para a criação de páginas *Web*. Os seus elementos servem para definir a formatação, apresentação, objetos (imagens, sons) e *links*, mas não possui os recursos de uma linguagem de programação. A sua sintaxe é simples e universal.

A linguagem consiste em uma série de *tags* integradas em um documento texto. Para visualizar os documentos HTML, pode ser utilizado qualquer navegador *Web*, como o Internet Explorer, *Mozilla Firefox* ou o *Netscape*.

Uma *tag* é uma etiqueta (*label*) que executa uma função no documento. Toda página HTML deve ser iniciada pela *tag* <html> e terminada pela *tag* </html>. Note que as *tags* podem ser escritas tanto em minúsculas quanto em maiúsculas. O fechamento das *tags* é sempre feito com uma barra "/" precedendo o seu nome.

```
<html>
<head>
<title> Título do Site </title>
</head>
<body background="figura.gif" bgcolor="red">
<h1>Texto de tamanho maior</h1>
<h3>Texto de tamanho médio</h3>
<h6>Texto de tamanho menor</h6>
<i>Itálico</i> <b>Negrito</b>
<h3 align="center">Texto ao centro</h3>
<h2 align="left">Texto à esquerda</h2>
Parágrafo: <p>
Quebra de linha: <br>
Linha horizontal: <hr>
<a href="http://endereco.com">Nome do Link</a>
<img src="Caminho da imagem">
<img src="Img.jpg" alt="Texto" align="right">
</body>
</html>
```

Tabelas em HTML: TR (*Table Row*), TH (*Table Header*) e TD (*Table Data*)

```
<TABLE border="1" summary="Statistics about fruit flies">
<CAPTION><EM>A test table with merged
cells</EM></CAPTION>
<TR><TH rowspan="2"><TH colspan="2">Average
    <TH rowspan="2">Red<BR>eyes
<TR><TH>height<TH>weight
<TR><TH>Males<TD>1.9<TD>0.003<TD>40%
<TR><TH>Females<TD>1.7<TD>0.002<TD>43%
</TABLE>
```

A test table with merged cells

	Average		Red eyes
	height	weight	
Males	1.9	0.003	40%
Females	1.7	0.002	43%

CSS

Tags HTML foram projetadas originalmente para definir o conteúdo de um documento. Elas devem indicar "Isto é um header", "Isto é um parágrafo" ou "Isto é uma tabela", utilizando *tags* como <h1>, <p>, <table>, e assim por diante. O layout do documento deveria ser tratado pelo navegador, sem usar nenhuma *tag* de formatação.

Os principais navegadores (Internet Explorer e Netscape) começaram a adicionar novas *tags* HTML e atributos (como a *tag* <font> e o atributo *color*) à especificação original do HTML, aumentando a dificuldade de criar páginas Web onde o conteúdo dos documentos HTML estavam separados do seu *layout* (apresentação).

Para resolver esse problema, o World Wide Web Consortium (W3C) criou STYLES em adição ao HTML 4.0. A maioria dos navegadores atuais suportam o CSS (*Cascading Style Sheets*):

- Estilos definem como os elementos HTML devem aparecer
- Estilos são normalmente armazenados em *Style Sheets*
- *External Style Sheets* são armazenados em arquivos CSS, e economizam tempo de desenvolvimento
- Definições múltiplas de estilos se cascateiam em uma

Continuar em: http://www.w3schools.com/css/css_intro.asp

Javascript

JavaScript é uma linguagem que executa no Cliente, e permite injetar lógica em páginas HTML. Os parágrafos de lógica do *JavaScript* podem estar "soltos" ou atrelados a ocorrência de eventos. Os parágrafos soltos são executados na seqüência em que aparecem na página (documento) e os atrelados a eventos são executados apenas quando o evento ocorre. Para inserir parágrafos de programação dentro do HTML é necessário identificar o início e o fim do set de *JavaScript*, da seguinte forma:

```
<SCRIPT>
Set de instruções
</SCRIPT>
```

Para melhor visualização e facilidade de manutenção, recomenda-se que toda a lógica seja escrita no início do documento, através da criação de funções a serem invocadas quando se fizer necessário (normalmente atreladas a eventos). Se a lógica é escrita a partir de um determinado evento, não é necessário o uso dos comandos <SCRIPT> e </SCRIPT>

Comandos do Javascript

Note que o *Javascript* é sensível ao tipo de letra (maiúsculas e minúsculas). Caso seja cometido algum erro de sintaxe quando da escrita de um comando, o *JavaScript* interpretará, o que seria um comando, como sendo o nome de uma variável.

Operadores lógicos: são iguais ao Java: == (igual), != (diferente), > (maior), >= (maior ou igual), && (E), || (Ou), etc.

Operadores matemáticos: também similar ao Java: + (adição), - (subtração), * (multiplicação), / (divisão) e % (módulo).

Controles especiais: os caracteres de escape \n, \r, \t, etc. Comentários usam // e /* ... */, e *Strings* delimitam-se por "" ou ' '.

Comandos condicionais: os comandos *if*, *for*, *while*, *break* e *continue* e o operador ? funcionam como em Java.

```
var contador = 10
while (contador > 1)
{
     if (teste = 3) strTeste = "Fim\n";
     contador--;
}
```

As variáveis são criadas automaticamente, pela simples associação de valores a mesma. Ex. NovaVariavel = "Jose"

O *JavaScript* permite que o programador escreva linhas dentro de uma página (documento), através do método write. As linhas escritas desta forma, podem conter textos, expressões *JavaScript* e comandos Html.

```
<script>
valor = 30
document.write ("Minha primeira linha")
document.write ("Nesta linha aparecerá: " + (10 * 10 + valor))
</script>
```

Eventos

São fatos que ocorrem durante a execução do sistema, a partir dos quais o programador pode definir ações a serem realizadas pelo programa. Segue a lista dos eventos possíveis, indicando os momentos em que os mesmos podem ocorrer:

- **onload:** ocorre na carga do documento. Ou seja, só ocorre no BODY do documento.
- **onunload:** ocorre na descarga (saída) do documento. Também só ocorre no BODY.
- **onchange:** ocorre quando o objeto perde o foco e houve mudança de conteúdo. Válido para os objetos Text, Select e Textarea. O evento **onblur** ocorre quando o objeto perde o foco, independente de mudanças.
- **onfocus:** ocorre quando o objeto recebe o foco. Válido para os objetos Text, Select e Textarea.
- **onclick:** ocorre quando o objeto recebe um clique do Mouse. Válido para os objetos Buton, Checkbox, Radio, Link, Reset e Submit.
- **onmouseover:** ocorre quando o ponteiro do mouse passa por sobre o objeto. Válido apenas para Link.
- **onselect:** ocorre quando o objeto é selecionado. Válido para os objetos Text e Textarea.
- **onsubmit:** ocorre quando um botão tipo Submit recebe um clique do mouse. Válido apenas para o Form.

Funções do Javascript

O *JavaScript* permite que o programador escreva linhas dentro de uma página (documento), através do método *write*. As linhas escritas desta forma, podem conter textos, expressões *JavaScript* e comandos Html. Elas aparecerão no ponto da tela onde o comando for inserido. Pode ser utilizado
 e <p> para quebra de linha e parágrafo, respectivamente.

```
<script>
valor = 30
document.write ("A primeira linha <br>")
document.write ("Nesta linha aparecerá : " + (10 * 10 + valor) + "<p>")
</script>
```

Existem três formas de comunicação por mensagens: **alert** (aviso), **confirm** (mensagem Ok / Cancelar) e **prompt**:

```
if (confirm ("Algo está errado...devo continuar??"))  {
    Entrada = prompt("Informe o seu nome", "")
    alert("Bem-vindo, " + Entrada + "!")}
else
{ alert("Parando") }
```

Também é possível criar as suas próprias funções em *Javascript*, utilizando:

```
function NomeFunção (Parâmetros)
```

```
{ Ação }
```

Ou utilizar uma função interna (*eval*, *parseInt*, *date*, *Math.abs*, *Math.pow*, *Math.max*, *Math.min*, etc).

Também existem outras funcionalidades, como criação de *Arrays*, manipulação de *Strings* ou datas, instâncias de objetos, etc.

DHTML

Dynamic HTML, ou DHTML é uma forma de executar *scripts* no lado do cliente, permitindo que as páginas ganhem conteúdo dinâmico, através da execução de *scripts* simples e fáceis de desenvolver.

O DHTML não é uma extensão da linguagem HTML, e tampouco uma linguagem de programação. Uma das desvantagens é que as implementações entre os diferentes navegadores (Internet Explorer, Netscape Navigator) não é uniforme.

A implementação DHTML, apresentada pela Microsoft na versão 4.0 do Internet Explorer, cobre quatro áreas principais que, na perspectiva do criador de conteúdo para a Web, lhe oferecem novas funcionalidades extremamente interessantes.

- **Estilos Dinâmicos:** é possível alterar dinamicamente qualquer atributo CSS de qualquer texto. Entre esses atributos incluem-se, por exemplo: cores, visibilidade, fonts, etc. A utilização de CSS permite controlar a apresentação de uma página Web. Utilizando DHTML, torna-se possível criar efeitos interativos alterando qualquer um dos atributos de estilo de qualquer um dos elementos da página.
- **Posicionamento Dinâmico:** possibilita o posicionamento de objectos na página através da manipulação de determinados atributos CSS (top, left, width, right, etc). Isto para além da possibilidade de controlar a "Z-order" dos objetos (que define a sua sobreposição vertical), o que permite, por exemplo, colocar uma imagem sobre outra.
- **Conteúdo Dinâmico:** permite adicionar, eliminar ou modificar qualquer porção de texto ou gráficos em qualquer posição da página. Sempre que o conteúdo da página for modificado, esta será redesenhada, de forma a englobar o novo conteúdo da melhor forma.
- **"Data Binding":** é possível apresentar, manipular e atualizar os dados da página, estabelecendo ligações entre os dados e *tags* HTML. Essa fonte de dados pode ser tão simples como um arquivo de texto ou tão complicada como uma base de dados acessível via ODBC.

Essas áreas cobertas pelo Internet Explorer são diferentes de como foi implementado no Netscape Navigator, ou outros.

Arquitetura de aplicações para ambiente web

Servidor de aplicações

Servidor Web

Ambientes Internet, Extranet, Intranet e Portal - finalidades, características físicas e lógicas, aplicações e serviços

Servidor de Banco de Dados

Objetos Distribuídos

Arquitetura de software: arquitetura 3 camadas, modelo MVC

Web services

Web Services são o último conceito na computação distribuída, sendo funções (ou aplicações) de negócio modulares, auto-descritivos, que operam na rede. Eles facilitam a interação entre companhias através da simplificação das regras de comunicação, utilizando XML como o formato de troca de mensagens em comum (protocolo neutro).

Embora os *web services* possam ser acessados utilizando uma chamada de procedimento remota (RPC), a comunicação assíncrona também é possível, o que é ideal para grupos de *web services* com baixo acoplamento, operando independentemente.

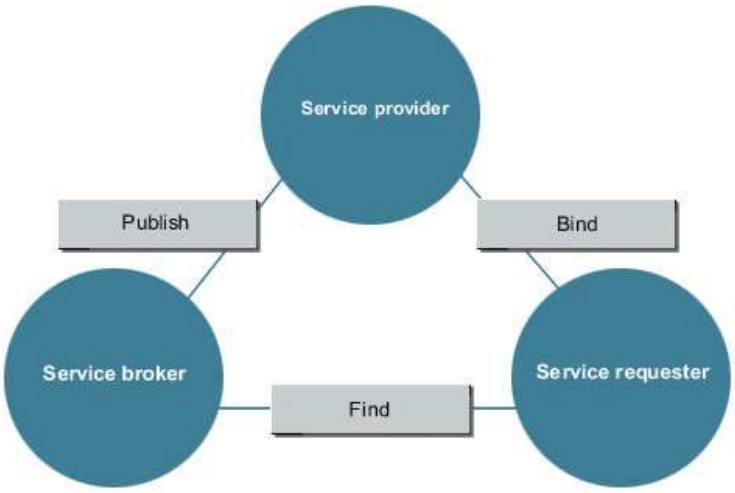
Web Services também oferecem vários benefícios às organizações, nos requisitos interação e integração, pois infra-estruturas diferentes podem se comunicar. Porém, a segurança é uma preocupação que deve ser levada em conta.

Um exemplo de aplicação para *web services:* uma empresa pode permitir que seus fornecedores monitorem o estoque de um produto, para que possam abastecê-lo sem necessidade de enviar ordens de serviço. Isso faz com que os fornecedores satisfaçam os seus clientes de forma mais rápida, além de reduzir custos (para todos os três: empresa, fornecedores e clientes).

Arquitetura Service-Oriented Architecture (SOA)

A arquitetura de *web services* pode ser descrita em termos da *Service Orientated Architecture* (SOA). Ela é uma evolução do modelo orientado a objetos, mas todas as entidades são entendidas como serviços, que publicam uma interface para uso de outros serviços na rede. A SOA é uma mudança do modelo tradicional cliente / servidor, onde as aplicações compartilham dados, acrescentando também o compartilhamento de serviços de negócios entre empresas.

A arquitetura SOA para *web services* compreende três papéis: *provider*, *broker*, e o *requestor* – e três operações entre estes papéis – publicação(*publish*), busca (*find*) e ligação (*bind*). Os próprios componentes dessa arquitetura são os *web services*, que consistem em duas partes: a sua implementação e a descrição ou interface do serviço.



Service Provider: fornece o *web service*, que é o componente de software que os outros serviços podem acessar e usar. Pode ser desde uma classe Java até uma aplicação em COBOL. O *service provider* também descreve o serviço utilizando uma interface, que inclui informações como o formato dos dados e protocolos de rede.

***Service Broker (ou registry)*:** a interface do *provider* é publicada para o *broker*, que é um diretório centralizado com as descrições dos *web services*. Define também a API para registrar e buscar os serviços.

Service Requestor: é o cliente em potencial do serviço, que é disponibilizado pelo *broker*. Quando o *requestor* encontra o serviço que deseja utilizar, ele invoca o serviço do *provider* (operação *bind*). Pode ser tanto um indivíduo que procura os serviços pela Internet, uma organização ou uma aplicação de *web services* que invoca os serviços.

Tecnologias dos Web Services

XML é a maneira de descrever os dados em um formato rígido, que impõe uma estrutura que pode ser interpretada. XML é baseada em texto, independente de plataforma e linguagem, e adequados para *web services*. Outras tecnologias são:

- **SOAP (*Simple Object Access Protocol*):** utiliza XML para descrever uma maneira em que as mensagens devem ser passadas em *web services*. Uma mensagem SOAP consiste de um ou mais cabeçalhos, um corpo, e um envelope que contém o próprio *web service* e os endereços de destino. As mensagens unidirecionais SOAP podem ser combinadas para gerar vários padrões de mensagens:
    - *Multicast:* o remetente pode fazer um broadcast da mensagem SOAP para múltiplos destinatários.
    - *Workflow:* a mensagem SOAP pode ser enviada para vários endpoints antes de retornar ao remetente.
    - *Request-response:* o remetente envia a mensagem para um recipiente, e recebe a confirmação (= RPC).
- **WSDL (*Web Services Definition Language*):** descreve os *web services*, permitindo que os usuários entendam como acessá-lo. Também é uma linguagem baseada em XML, e especifica também o protocolo de transporte, as capacidades do serviço, e os pontos de comunicação em ambos os canais de comunicação. A interface WSDL atua como uma interface remota Java, ou uma IDL Corba, e também é conhecida como XML-based IDL.
- **UDDI (*Universal Description, Discovery and Integration*):** é um registro centralizado dos *web services*, similar a um catálogo de telefones, utiliza SOAP e XML em suas APIs, além de usar a linguagem WSDL para promovê-los. Existem três tipos de registros definidos na especificação UDDI:
    - *White pages*: permite o registro de informações como o próprio serviço, IDS, contrato e dados relacionados.
    - *Yellow pages*: permite organizar o serviço pela indústria, geografia ou tipo de serviço. (Páginas Amarelas).
    - *Green pages*: inclui detalhes técnicos de como encontrar e executar um *web service*.

Para criar um *web service*, antes deve ser criado um "resumo" do serviço utilizando um arquivo WSDL. Então os detalhes do serviço devem ser registrados / publicados utilizando um registro UDDI. Um cliente envia uma requisição ao diretório, e é

capaz de contatar o servidor SOAP após obter diretamente os detalhes do WSDL do registro UDDI. Assim o cliente pode utilizar o *web service* através do envio e recebimento de mensagens SOAP.

Portais Corporativos

Portal Corporativo pode ser definido como uma aplicação tipicamente web, desenvolvida para funcionar como interface única e personalizada do ambiente eletrônico de trabalho, provendo aos usuários conteúdo, informação, acesso a aplicações, colaboração e conhecimentos necessários a plena atuação, envolvendo todo relacionamento com os *stakeholders* da empresa

Ainda recente e sem que tenhamos aprendido muito com eles, os Portais Corporativos, como uma resposta à evolução da Intranets, vieram sem dúvida para ficar e promover uma poderosa transformação no trabalho e nas empresas. Como um grande aliado da gestão do conhecimento, rapidamente se tornarão comuns à maioria das empresas.

Portal Corporativo ou Intranet de terceira geração (I3g) significa ao funcionário ter um único ponto para acesso a todas as informações relevantes, ferramentas de colaboração, aplicações e serviços que ele precisa para fazer o seu trabalho. Do ponto de vista da empresa, representa economia e produtividade aumentadas.

Cada funcionário passa a ter em seu desktop, uma janela única que agrupa todas as informações necessárias a execução do trabalho diário. Além disso, oferecem considerável independência de lugar, pois através de conexão com dispositivos móveis é possível aos funcionários e parceiros da empresa acessar o portal e sua riqueza de informação, onde quer que eles estejam.

Motivação e Características dos Portais

As razões que estimulam a criação de portais corporativos nas empresas são muitas. Algumas das principais são:

- a dificuldade de acesso à informação;
- muita gente perdendo tempo respondendo a dúvidas dos funcionários que deveriam ser respondidas pela Intranet;
- Intranet que oferece tudo para todo mundo (falta de personalização);
- falta de informação e conteúdos aplicáveis ao trabalho (resolução de problemas, comercialização, atendimento ao cliente, criação de produtos, etc...);
- vários ícones para se chegar a informação desejada e conseqüentemente várias senhas de acesso às aplicações;
- falta de padrão das interfaces de acesso a informação e entre as aplicações;
- baixa qualidade das informações;
- desperdício de tempo na procura de informações;
- várias Intranets na empresa, produtividade das equipes deficiente;
- pouco contexto entre as informações da Intranet e a realidade de negócios;
- baixos índices de colaboração entre os talentos da empresa e falta de informação para o trabalho.

Os portais corporativos altamente integrados com aplicações (desenvolvidas internamente e por terceiros), ferramentas de colaboração e serviços de informação de diferentes fontes, formatos e ambientes, permitem integrar processos de negócios e viabilizam práticas conhecidas como *Business to Employee* (B2E). Há a promessa de redução de despesas e desperdícios, aumento de produtividade, retenção de talentos, aumento de empowerment, melhoria de qualidade e atendimento, redução do tempo de respostas e resolução de problemas, prontidão de respostas, colaboração, trabalho em grupo, aprendizagem, etc.

Entre todas as razões o que torna mais urgente os serviços tipo B2E é a possibilidade de se resgatar aos funcionários da empresa, o bem mais precioso de nossos dias: o tempo. Quanto tempo seu pessoal desperdiça todos os dias procurando por informações que deveriam estar publicadas na Intranet? Quanto tempo as pessoas desperdiçam delas próprias e de outros funcionários, se ocupando mutuamente com assuntos que poderiam ser resolvidos pela Intranet? Quanto as pessoas consomem de telefone na resolução de problemas ou busca de informações que poderiam estar na Intranet? Quanto esforço deixa de ser aplicado na atividade fim da empresa por estas razões? Quanto isso representa ($) por ano em sua companhia?

Além de apoiar a comunicação empresarial e a aprendizagem dinâmica, os portais corporativos suportam muitas das iniciativas de gestão do conhecimento baseadas em Tecnologia da Informação, alavancando a integração dos processos de negócios, colaboração, comunidades de prática, e-learning, e-CRM, disseminação de melhores práticas, espaços virtuais de trabalho, relacionamento com os stakeholders, inteligência competitiva, repositório de documentos, grupos de discussão, web conference, instant messaging e muito mais, se transformando também em uma poderosa vitrine de acesso ao conhecimento.

Entre as características mais básicas de um portal corporativo temos:

- **Templates:** a As páginas de um portal sempre apresentam elementos comuns, tais como *banners*, barra de navegação, cabeçalho, rodapé entre outros. Estes elementos podem ser armazenados em um ou mais templates, os quais serão utilizados pelas páginas que compõem o portal. Isto permite que os construtores do portal atualizem a aparência e a estrutura das páginas de maneira centralizada, via templates, atualizando automaticamente todas as páginas relacionadas. Com templates, os desenvolvedores das aplicações não precisam preocupar-se com aspectos de montagem da página completa, mas apenas com as interfaces da sua aplicação.
- **Escalabilidade e Robustez:** além de um bom desempenho, um framework oferece escalabilidade para dar apoio às exigências dos portais de grande porte. Isto inclui trabalhar com um ambiente de balanceamento de carga, onde servidores web podem ser dinamicamente adicionados ou removidos, dependendo da demanda e das exigências de acesso. Escalando para múltiplos servidores, também aumenta a robustez e a disponibilidade da solução, dando suporte à tolerância de falhas de hardware.
- **Integração Total com Aplicações:** os frameworks fornecem uma camada de apresentação comum para todo o portal. As informações provenientes dos aplicativos são disponibilizadas para o portal através de subseções de uma página, que recebem diferentes nomes nos diferentes produtos: “interfaces”, “portlets”, “gadgets” e “webparts” são alguns exemplos. Adotaremos "interface". Uma página do portal pode conter uma ou mais interfaces, que podem pertencer a diferentes aplicativos. O framework fornece ainda uma camada de comunicação entre as interfaces, que permite aos aplicativos trocarem informações de uma maneira fácil e genérica. A integração das interfaces dentro de uma página do portal permite ao framework ter um maior controle da identidade visual do portal corporativo. Tecnologias como CSS, XML e XSL são utilizadas pelo framework para controlar e manter a consistência visual das informações exibidas para o usuário final.
- **Gerando Log e Relatórios:** incluem ferramentas para registro e geração de relatórios sobre a utilização do portal. Com isso, desenvolvedores têm uma preocupação a menos ao implementar suas aplicações. Estas informações são essenciais para que os administradores possam controlar e analisar o ROI do portal.
- **Mecanismo de Busca:** a capacidade de centralização e integração da busca nas diferentes aplicações, bem como em documentos corporativos, estão entre as principais funcionalidades oferecidas por um framework. A busca é geralmente integrada com a personalização e o controle de acesso do portal, retornando apenas resultados de áreas às quais os usuários tenham permissão de acesso.
- **Segurança:** aplicações construídas a partir de um framework não precisam preocupar-se com a administração da segurança do portal. Acesso via HTTPS, validação de login, integração com mecanismos de autenticação externos e a tecnologia *single-sign-on* (senha única de acesso a todas aplicações) são tratados pelo próprio portal. Isto não centraliza a autenticação e alivia os técnicos de grandes preocupações no desenvolvimento de uma solução.
- **Personalização, Colaboração e Integração com SGC (ou CMS - Content Management System); (ver adiante)**

Em sua essência, o portal corporativo permite à empresa organizar tanto a distribuição e uso de seus ativos de tecnologia da informação, quanto o acesso ao conhecimento em seu estado mais genuíno, ou seja, face a face, a partir do relacionamento humano. Traz alta capacidade de criação de contexto de uso das informações e facilidade de integração à interface de trabalho de aplicações como ERP, CRM, e-mail, agendas e calendários, ferramentas de escritório, de busca, CGS, LMS, gerenciamento de documentos, ferramentas de comunicação e colaboração e aplicações desenvolvidas internamente.

Como os portais permitem embutir muitas ferramentas, tecnologias e utilidades, além dos benefícios diretos resultantes de sua utilização massiva, grande parte dos investimentos realizados em T.I. passam a ser consolidados e depreciados dentro da empresa. Iniciativas mais amplas de portal podem estender suas funcionalidades a parceiros, fornecedores, clientes, distribuidores, acionistas e outros interessados.

Os portais corporativos parecem responder positivamente às necessidades competitivas das companhias, valorizando simultaneamente as expectativas de ROI, o uso efetivo dos ativos da tecnologia da informação e seus ativos intelectuais.

Gestão de Conteúdo

Um componente importante de qualquer solução de portal corporativo deveria ser a descentralização e delegação de poder para que os funcionários possam facilmente incluir informação e conhecimento no sistema e ter estes “inputs” eficientemente e rapidamente distribuídos para grupos específicos, toda a empresa e até mesmo clientes. Dessa maneira, as implementações Sistema de Gestão de Conteúdo (SGC), associadas com implementações portais corporativos, representam uma oportunidade formidável e um desafio para as organizações e consultorias envolvidas na implementação de portais corporativos.

Os SGCs estão se tornando uma necessidade prática em operações web onde milhares de páginas compartilham de elementos comuns (ex: marca da empresa). É muito difícil manter a integridade de páginas de site e seus vários links, de maneira geral, sem o uso de SGCs avançados. Em um ambiente de publicação web, uma simples página ou conjunto de páginas pode ser formada a partir de centenas de fontes de dados, links, gifs e imagens, que são armazenados em arquivos distintos.

Os SGCs mais avançados permitem a integração fácil e dinâmica de dados muito estruturados (dos sistemas de back-end) e de dados nãoestruturados (informados pelos indivíduos de dentro e de fora da empresa). Os SGCs podem também ser integrados profundamente com aplicações eletrônicas de workflow e outras ferramentas de colaboração e gerenciamento de projeto. Um processo padrão de SGC geralmente envolve os seguintes passos:

1) Criação de Documentos;
2) Revisão de Documentos;
3) Inclusão de Metadata e Controle de Qualidade;
4) Publicação;
5) Revisão Periódica;
6) Arquivamento ou Eliminação dos Documentos.

As implantações de SGC podem desempenhar um papel crítico na implementação dos portais corporativos. Os SGCs fornecem a infra-estrutura técnica e os processos centrais que garantem que o conteúdo correto, atualizado e pontual estará disponível para aqueles que precisarem. Embora engenhosa, em um nível conceitual, as plataformas SGC podem ser bastante diferentes em termos de suas várias características técnicas. Algumas das características encontradas nas soluções SGC mais avançadas estão incluídas, mas não limitadas, na seguinte lista:

- **Funcionalidades de Criação e Design**
    - Existem poucas limitações de layout e projeto;
    - O conteúdo é separado do formato;
    - Inclusão de ferramentas gráficas e intuitivas para se construir um workflow;
    - Permite-se que os usuários não apenas publiquem informação / conteúdo, mas também customizem facilmente a interface de suas publicações;
    - Facilidade para os usuários não-técnicos trabalharem com seus aplicativos de escritório (desktop);
    - Os templates são modificados facilmente;
    - Metadata é incluida automaticamente;
    - Permite-se a criação de documentos baseados em XML por usuários que desconheçam XML;
    - Facilidade para o usuário organizar, classificar e fazer referências cruzadas do conteúdo publicado;
    - Permissão para o usuário associar facilmente termos de busca (palavraschave) com o conteúdo criado;
    - Suporte à publicação de conteúdo em muitos tipos de arquivos (ex.: áudio, vídeo, imagem, apresentação, código HTML, componentes Java, arquivos ZIP para download, etc.);
    - Permissão para os criadores de conteúdo incluírem níveis de prioridade no documento que será publicado ou distribuído para grupos selecionados;
    - Habilidade para customizar dinamicamente a navegação, o conteúdo e aplicativos que serão mostrados a diferentes grupos de usuários;
    - Usuários podem utilizar as ferramentas para criação de conteúdo que lhes sejam mais adequadas.
- **Definição de Regras**
    - Permite-se mudanças fáceis de regras para autoria, edição, aprovação, publicação e remoção de conteúdo;
    - Os usuários comuns podem também definir ou mudar facilmente as regras e fluxo do processo de publicação das páginas;
    - Os empregados e/ou o *webmaster* podem gerenciar facilmente papéis e direitos de acesso;
    - Permite-se a publicação pública e com controle de privacidade (isto é, os usuários controlam quem têm acesso às suas páginas publicadas).
- **Administração e Controle de Versões**

- Muitas opções para controle de versão;
- Permite-se visualizar o "histórico de mudanças" de algum item específico que tenha sido publicado;
- Geração automática de atributos (metadata) associados com cada documento publicado (data da criação, criador, tamanho do documento, indicador de item novo, indicador de item atualizado, etc.);
- Permite-se que papéis sejam facilmente designados: quem pode ler, editar, arquivar e eliminar um documento ou página;
- Permite-se a adição de comentários para documentos revisados;
- Permite-se voltar atrás na publicação de itens caso necessário;
- São fornecidos relatórios e ferramentas para monitorar facilmente o sistema;
- Inclui alertas via e-mail para os administradores;
- Arquivos totalmente passíveis de auditoria são fornecidos.

- **Arquitetura**
  - Inclui servidores separados para desenvolvimento, teste e produção?
  - Inclui um repositório global e robusto? O sistema permite a criação de repositórios agrupados?
  - Qual o nível de granularidade que o conteúdo pode ser armazenado no repositório? Esses níveis são facilmente configurados pelo usuário? São facilmente remontadas e disponibilizadas novamente?
  - O sistema produz soluções em termos de redundância no caso de perda total ou parcial dos repositórios?
  - O sistema simplifica a reclassificação e recombinação de conteúdo para novos usuários e para a entrega através de múltiplos canais ?
  - Quais são as habilidades de integração do sistema? E as ferramentas de desenvolvimento e aplicações?
  - É baseado em XML? Suporta LDAP? Quais outros padrões são suportados?
  - O sistema se integra com as plataformas líderes de PdCC, bancos de dados (Oracle, Sybase, SQL Server, Informix, etc.), aplicações legadas, servidores web, servidores de aplicação, servidores comerciais, etc. ? São fornecidos adaptadores pré-desenvolvidos para essas integrações?
  - Ele suporta a integração de fornecedores e parceiros em um Processo de Gerenciamento de Conteúdo necessário para disponibilizar transações de comércio eletrônico em tempo real?
  - O sistema inclui *caching*?
  - O sistema inclui procedimentos simples para backup?

Os aspectos essencialmente técnicos relacionados aos SGC, embora bastante importantes e não triviais, dizem respeito apenas a uma pequena parte das questões relacionadas a uma efetiva implementação de sistemas de gerenciamento de conteúdo no contexto de portais corporativos. Muito mais importante são os temas estratégicos e organizacionais que necessariamente devem ser transformados durante processos de implementação de SGCs.

Uma efetiva gestão de conteúdo pode transformar como uma empresa pública, compartilha e utiliza informação, conhecimento e poder. Por isso, novas políticas de compartilhamento de informação interna e externa (com clientes) e descrições de cargo (definindo responsabilidades, inclusive da alta gerência) devem acompanhar a implementação das soluções de SGC.

JSR 168 - Java Specification Request 168 Portlet Specification

O JSR 168 da Sun define um padrão API oferecendo interface comum para agregar diversas fontes e aplicações de conteúdo em um único portal. É suportado pelo Sun ONE Portal Server.

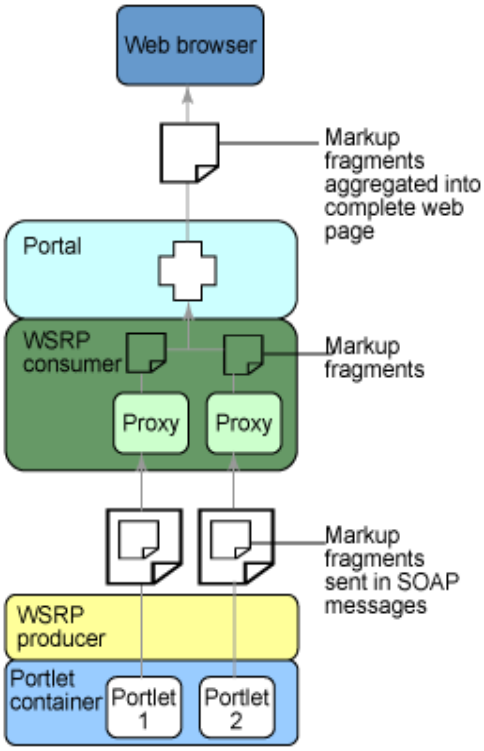
A diferença entre Portal e *Portlet* também é importante:

- **Portal:** pode ser definido como uma aplicação baseada na Web que é customizada pelo usuário final, tanto em sua apresentação quanto no seu conteúdo, inclusive de aplicações contidas neste. É responsável por agregar o conteúdo e outras aplicações em um ponto único de entrada para o usuário (e suas ferramentas).
- **Portlets:** são pequenas aplicações Web que rodam dentro de um portal, ao lado de outras entidades similares. É um componente Web definido pela especificação JSR-168, e gerenciado por um contêiner, podendo processar requisições e gerar conteúdo dinâmico.

WSRP - Web Services for Remote Portlets

O WSRP, do OASIS, é uma forma padronizada simples de integração de aplicações / conteúdos remotos dentro de portais, oferece aos administradores de portais poderem escolher entre uma rica variedade de serviços e permitem a integração deles em seus portais sem esforços de programação. Como resultado, os WSRP se tornaram os meios para os fornecedores de conteúdo e aplicações para fornecer seus serviços para as organizações rodando portais de uma forma fácil e consumível.

Enquanto *Web Services* permitem a reutilização de serviços de *back end*, o WSRP permite a reutilização de toda a interface com o usuário. Em outras palavras, permite aos provedores de soluções integrarem os *Web services* a portais sem a necessidade de escrever códigos de comunicação com a plataforma que roda serviços.



O WSPR define os seguintes atores em uma arquitetura WSPR:

WSRP producer: este é um Web service que oferece um ou mais portlets, e implementa um conjunto de interfaces WSRP, fornecendo um conjunto comum de operações para os consumidores. Dependendo da implementação, um produtor pode oferecer apenas um portlet, ou pode fornecer um container para implantação e gerenciamento de vários portlers. O produtor é um Web service completo (descrito por WSDL).

WSRP portlet: A WSRP portlet is a pluggable user interface component that lives inside of a WSRP producer and is accessed remotely through the interface defined by that producer. A WSRP portlet is not a Web service in its own right (it cannot be accessed directly, but instead must be accessed through its parent producer).

WSRP consumer: This is a Web service client that invokes producer-offered WSRP Web services and provides an environment for users to interact with portlets offered by one or more such producers. The most common example of a WSRP consumer is a portal.

WSRP defines a set of common interfaces that all WSRP Producers are required to implement and which WSRP Consumers must use to interact with remotely-running portlets. Standardizing these interfaces allows a portal to interact with remotely-running portlets generically, since it has a well-defined mechanism for communicating with any WSRP-compliant producer. The WSRP Specification requires that every producer implement two required interfaces, while allowing them to optionally implement two others as well:

Service Description Interface (required): The Service Description Interface allows a WSRP producer to advertise its capabilities to perspective consumers. A WSRP consumer can use this interface to query a producer to discover what portlets the producer offers, as well as additional metadata about the producer itself. This interface can act as a discovery mechanism to determine the set of offered portlets, but also importantly allows consumers to determine additional information about the producer's technical capabilities. The producer's metadata might include information about whether the producer requires registration or cookie initialization before a consumer can interact with any of the portlets.

Mark-up Interface (required): The Markup Interface allows a WSRP consumer to interact with a remotely running portlet on a WSRP producer. For example, a consumer would use this interface to perform some interaction when an end-user submits a form from the portal page. Additionally, a portal might need to simply obtain the latest mark-up based on the current state of the portlet (for example when the user clicks refresh or interaction with another portlet on the same page takes place).

Registration Interface (optional): The Registration Interface allows a WSRP producer to require that WSRP consumers perform some sort of registration before they can interact with the service through the Service Description and Mark-up interfaces. Through this mechanism a producer can customize its behavior to a specific type of consumer. For example, a producer might filter the set of offered portlets based on a particular consumer. In addition, the Registration Interface serves as a mechanism to allow the producer and consumer to open a dialogue so that they can exchange information about each others' technical capabilities.

Portlet Management Interface (optional): The Portlet Management Interface gives the WSRP consumer access to the life cycle of the remotely-running portlet. A consumer would have the ability to customize a portlet's behavior or even destroy an instance of a remotely-running portlet using this interface.

Acessibilidade

A importância dessa temática na área da tecnologia da informação, tem se tornado um foco de atenção de gestores públicos e privados e equipes desenvolvedoras de sistemas. Acessibilidade na internet se refere à flexibilidade de acesso à informação e interação de usuários que possuem algum tipo de deficiência, quando os mesmos interagem com mecanismos de navegação, manuseio de páginas e telas e com operação de software.

Decreto nº 5296, de 02/12/2004

Segundo o Decreto n. 5.296, de 02/12/2004, acessibilidade significa: condição para utilização, com segurança e autonomia, total ou assistida, dos espaços, mobiliários e equipamentos, das edificações, dos serviços de transporte e dos dispositivos, sistemas e meios de comunicação e informação, por pessoa portadora de deficiência ou com mobilidade reduzida.

Algumas situações e características que o usuário pode estar sujeito e que podem impactar sua navegação:

- Incapacidade de ver, ouvir ou deslocar-se;
- Impossibilidade de interpretar certos tipos de informação;
- Dificuldade visual para ler ou compreender textos;
- Falta de percepção de texto e de elementos gráficos quando vistos sem cores;
- Dificuldade para falar ou compreender a língua em que o documento foi escrito.

Essas diferentes situações precisam ser consideradas pelos gestores de intranets e portais corporativos, para que possam oferecer respostas simultâneas a vários grupos de deficientes.

Um sítio que respeita a navegação dos deficientes visuais, deve obedecer a regras simples que possibilitam uma boa interpretação das páginas por aplicativos de leitura de tela, proporcionando aos deficientes visuais entendimento do conteúdo sem que tenham que "adivinhar" os caminhos que levam à informação.

Deficientes visuais utilizam o teclado como ferramenta principal de navegação na web, feita através de tabulações e reconhecimento de telas apresentadas no browser. O aplicativo de leitura de tela lê a parte textual da página e reconhece a existência de imagens e animações em flash, contudo essas imagens e animações sempre aparecem como algo sem sentido, pois a mensagem que passa não é assimilada.

Por isso, toda informação deve, também, ser apresentada na forma de texto. Se uma informação for expressa de outra forma, por exemplo, imagem, animação, filme ou som, ela deverá ser repetida na forma de descrição textual. Esta descrição textual deve ser “equivalente”, ou seja, deve preservar o sentido da informação. No HTML, os atributos “alt” e “longdesc” foram criados para textos equivalentes. Outras técnicas para tornar a navegação mais amigável para portadores de deficiência:

- Use rótulo de identificação e descrição para imagens e logomarcas;
- Use rótulo de identificação de links em textos e imagens indicando o destino do mesmo;
- Use rótulo de identificação em formulários;
- Use "list menus" na navegação por tabulação - essa é uma das boas opções para navegação;
- Os scripts geralmente não são muito bem resolvidos pelos aplicativos de leitura de tela. Se usá-los, tenha sempre uma segunda opção de navegação;
- *Sites* totalmente construídos em flash não são reconhecidos pelos aplicativos de leitura de tela, impossibilitando a navegação. As animações flash podem conter descrições por meio do TAG do atributo TITLE, colocado no TAG OBJECT, que chama a animação flash;
- Forneça títulos de *links* que faça sentido quando lidos fora do contexto;
- Não confie apenas na cor para transmitir as informações sobre *links*. Ex: “Clique nos *links* vermelhos para editar...”.
- Utilize sempre os atributos de TAG com o parâmetro alt="descrição..." ou title="descrição...". Eles são reconhecidos pelo aplicativo de leitura de tela e ajudam a tornar os caminhos mais claros.

Modelo de acessibilidade

A história dos modelos de acessibilidade começou em 1998 com a "Section 508", que é uma lei que define: "a tecnologia inacessível interfere na capacidade individual de adquirir e usar a informação de maneira rápida e fácil".

No que se refere a acesso ao computador, ocorrem quatro tipos de situações por parte de usuários com necessidades especiais:

- **Acesso ao computador sem mouse:** pessoas com cegueira, dificuldade de controle dos movimentos, paralisia ou amputação de um membro superior. Tais pessoas sentem várias dificuldades na utilização do mouse;
- **Acesso ao computador sem teclado:** pessoas com amputações, grandes limitações de movimentos ou falta de força nos membros superiores têm sérias dificuldades para utilizar o teclado tradicional. Nesses casos, a interação poderá ser feita através de um periférico especial de reconhecimento da fala ou de um emulador de teclado na tela;
- **Acesso ao computador sem monitor:** a verdade é que a informação processada por um computador não é de natureza visual. Para obterem a informação que é projetada na tela, os cegos recorrem a um software (programa leitor de tela) que capta essa informação e a envia para um sintetizador de voz ou para um terminal Braille;
- **Acesso ao computador sem áudio:** encontram-se relacionadas neste caso pessoas com baixa audição e pessoas com surdez completa. Há dificuldade em acessar informações que se encontram disponíveis somente através de áudio.

Dentro deste contexto, o Departamento de Governo Eletrônico tem o compromisso de elaborar um Modelo de Acessibilidade para o desenvolvimento e a adaptação de conteúdos do governo na internet, gerando um conjunto de recomendações que devem ser consideradas. Uma das principais atribuições do Governo Federal é a **inclusão social** (diminuir desigualdades).

O Modelo de Acessibilidade proposto é baseado nas regras do W3C e outras normas, com as seguintes visões:

- **Visão Técnica:** cartilha de recomendações práticas para a construção e / ou adaptação de sítios eletrônicos. A Visão Técnica é voltada ao desenvolvedor, à pessoa que fará as alterações nos códigos dos sítios eletrônicos;
- **Visão do Cidadão:** arquitetura de segmentação da Visão Técnica. A Visão do Cidadão do Modelo de Acessibilidade proporciona uma orientação e compreensão mais lógica e intuitiva do modelo propriamente dito e da Visão Técnica. Abaixo então relacionamos as Áreas de Acessibilização compreendidas na Visão do Cidadão:
    - *Área da Percepção:* trata de benefícios relacionados à apresentação do conteúdo, da informação. Ela preocupa-se com a percepção de elementos como gráficos, sons, imagens, multimídia e equivalentes.
    - *Área da Operação:* preocupa-se com a manipulação da informação, do conteúdo. Deve garantir formas alternativas ao acesso às informações através de maneiras diferenciadas de navegação ou técnica similar.
    - *Área do Entendimento:* trata de questões relacionadas à compreensão do conteúdo publicado. Ela deve garantir que todo o conteúdo apresentado seja de fácil compreensão para qualquer tipo de usuário.
    - *Área da Compatibilidade:* aborda questões como a necessidade de utilizarmo-nos sempre de tecnologias acessíveis e compatíveis com o modelo aqui proposto.

Além disso, podemos classificar os conteúdos em três grandes Níveis de Acessibilidade, por prioridades, onde o Nível 1 são as exigências **básicas** de acessibilidade (pontos obrigatórios); o Nível 2 facilita o **acesso** às informações do documento, e o Nível 3 facilita o acesso aos **documentos** armazenados na Web. O nível AAA é aquele que está de acordo com todos níveis.

Os cinco passos do processo de acessibilização são:

1. Verificação da necessidade de acessibilização do conteúdo;
2. Acessibilização do conteúdo;
3. Validação da acessibilidade do conteúdo;
4. Promoção da acessibilidade conquistada;
5. Garantia contínua da acessibilidade.

Cartilha técnica

A cartilha técnica de recomendações foi desenvolvida buscando atender e propiciar a acessibilização dos sítios governamentais, como proposto em "eMAG, Acessibilidade de Governo Eletrônico - Modelo".

- **Diretriz 1.** Forneça alternativas equivalentes para o conteúdo gráfico e sonoro.
- **Diretriz 2.** Assegure-se de que seu sítio seja legível e compreensível mesmo sem o uso de formatações.
- **Diretriz 3.** Dê preferência às tecnologias de marcação e formatação.
- **Diretriz 4.** Assegure que toda a informação seja interpretada corretamente, com clareza e simplicidade.

- **Diretriz 5.** Assegure que as tecnologias utilizadas funcionem - de maneira acessível - independente de programas, versões e futuras mudanças.
- **Diretriz 6.** Assegure sempre o controle do usuário sobre a navegação no sítio.
- **Diretriz 7.** Identifique claramente quais são os mecanismos de navegação.
- **Diretriz 8.** Em casos não contemplados pelas diretrizes anteriores, utilize sempre recursos reconhecidos - por instituições com propriedade no assunto - como tecnologias acessíveis.

Os recursos técnicos para implementação da acessibilidade em HTML (a seguir) sumariza as recomendações da cartilha.

Recursos técnicos para implementação da acessibilidade em HTML

Visando tornar a Web acessível a um número cada vez maior de pessoas e com o objetivo de levá-la ao potencial máximo de interoperabilidade, o W3C, comitê formado por grandes empresas, criou o WAI (*Web Accessibility Initiative*). Entre outras atribuições, o WAI mantém grupos de trabalho elaborando conjuntos de diretrizes para garantir a acessibilidade do conteúdo da Web às pessoas com necessidades especiais, ou que acessam a Web em condições especiais.

As práticas recomendadas incluem **Conteúdo Web** (HTML, CSS, imagens, sons, etc), **Authoring Tool** (ferramentas que criam páginas *Web*), **User Agent** (navegadores e mídia), e as **tecnologias** (XML, SVG, SMIL, plataformas, etc).

- É uma boa prática utilizar <HTML lang="fr"> para identificar o idioma da página, e <SPAN lang="it"> para indicar uma mudança na língua no meio do texto. Isso facilita *softwares* de reconhecimento de caracteres especiais.
- Utilizar folhas de estilos (CSS) para controlar o *layout* e a apresentação do documento.
- Abreviações e citações devem ser indicadas pelos elementos <ACRONYM> e <BLOCKQUOTE>.
- Em fórmulas matemáticas, garantir que as variáveis são bem definidas. Por exemplo, F = Força, m = massa, etc.
- Elementos CITE (citações), DFN, CODE (código), SAMP, KBD, VAR, INS (texto inserido), DEL (texto deletado).
- Nas tabelas as informações das linhas e colunas devem ser definidas, e um sumário da tabela incluído.
- Fornecer um equivalente texto para imagens utilizadas como *links*, através do elemento "alt", "longdesc", etc.
- Criar uma ordem lógica de "tabs" entre os *links* e figuras (elemento TABINDEX).
- Incluir atalhos de teclas (elemento ACCESSKEY).
- Incluir descrições longas de imagens, e evitar arte ASCII, ou utilizar um *link* para pular esses blocos ASCII.
- Deve existir contraste entre as cores das figuras, fundo e texto, inclusive para facilitar a visualização monocromática.
- Prover o texto equivalente à áudios e vídeos. Isso também vale para gráficos, botões gráficos, etc.
- Escrever para navegadores que não suportam FRAMES, e caso sejam utilizados, descrever o seu relacionamento.
- Informar previamente ao usuário o destino e resultado da ação, quando houver campos e elementos do formulário.
- Assegurar a acessibilidade de objetos programados (*applets*, *scripts*, programas interpretados, *objects*).
- Evite comandos que caíram em desuso: "blink", "marquee", "center", "font", "menu", "u", entre outros.
- Não provocar o aparecimento de janelas de sobreposição, janelas *popup* ou mudanças dinâmicas na página atual.
- Sempre especificar fontes genéricas como opções em folhas de estilos.
- A linguagem deve ser clara e acessível (evite a voz passiva, linguagens idiomáticas, gírias, frases longas, etc).

Exemplo 1: Para uma imagem simples, decorativa ou acidental, fazer apenas uma breve descrição indicando que a imagem é de um carro amarelo, o código pode ser:

```
<img src="carro.jpg" alt="Foto de um carro amarelo.">
```

Exemplo 2: O atributo "longdesc" foi criado para situações em que a descrição a ser feita deve ser mais longa do que a permitida pelo atributo "alt".

```
<img src="mont_rio.jpg" longdesc="montanhas.htm" alt="Foto de montanhas no Rio de Janeiro.">
```

Exemplo 3: Se você produziu um script (ou um programa) que causa um aviso sonoro, para ser tocado se o visitante da sua página tentar enviar um formulário antes dos campos requeridos estarem preenchidos, coloque no seu programa (ou script) a capacidade de escrever uma mensagem na tela, que diga algo como:

Alerta: "Você tentou submeter um formulário incompleto. Por favor, preencha os campos necessários".

Segurança da Informação

Política de segurança

Os objetivos de uma política de segurança são, de acordo com a ICP-Brasil:

- Definir o escopo da segurança das entidades;
- Orientar, por meio de suas diretrizes, todas as ações de segurança das entidades
- Reduzir riscos e garantir a integridade, sigilo e disponibilidade das informações dos sistemas e recursos
- Permitir a adoção de soluções de segurança integradas
- Servir de referência para auditoria, apuração e avaliação de responsabilidades

Regras gerais da Política de Segurança

À política de segurança das entidades são aplicados os seguintes conceitos:

- **Ativo de Informação:** é o patrimônio composto por todos os dados e informações geradas e manipuladas durante a execução dos sistemas e processos das entidades;
- **Ativo de Processamento:** é o patrimônio composto por todos os elementos de hardware e software necessários para a execução dos sistemas e processos das entidades, tanto os produzidos internamente quanto os adquiridos;
- **Controle de Acesso:** são restrições ao acesso às informações de um sistema exercido pela gerência de Segurança da Informação das entidades;
- **Custódia:** consiste na responsabilidade de se guardar um ativo para terceiros. Entretanto, a custódia não permite automaticamente o acesso ao ativo, nem o direito de conceder acesso a outros;
- **Direito de Acesso:** é o privilégio associado a um cargo, pessoa ou processo para ter acesso a um ativo;
- **Ferramentas:** é um conjunto de equipamentos, programas, procedimentos, normas e demais recursos através dos quais se aplica a Política de Segurança da Informação das entidades;
- **Incidente de Segurança:** é qualquer evento ou ocorrência que promova uma ou mais ações que comprometa ou que seja uma ameaça à integridade, autenticidade, ou disponibilidade de qualquer ativo das entidades;
- **Política de Segurança:** é um conjunto de diretrizes destinadas a definir a proteção adequada dos ativos produzidos pelos Sistemas de Informação das entidades;
- **Proteção dos Ativos:** é o processo pelo qual os ativos recebem classificação quanto ao grau de sensibilidade. O meio de registro de um ativo de informação recebe a mesma classificação de proteção dada ao ativo que o contém;
- **Responsabilidade:** é definida como as obrigações e os deveres da pessoa que ocupa determinada função em relação ao acervo de informações;
- **Senha Fraca ou Óbvia:** é aquela onde se utilizam caracteres de fácil associação com o dono da senha, ou que seja muito simples ou pequenas, tais como: datas de aniversário, casamento, nascimento, o próprio nome, o nome de familiares, seqüências numéricas simples, palavras com significado, dentre outras.

A abrangência da política de segurança de uma entidade deve levar em conta os requisitos de segurança humana, segurança física, segurança lógica (recursos criptográficos, etc). Algumas de suas regras gerais são:

- A Política de Segurança Geral se aplica a todos os recursos humanos, administrativos e tecnológicos pertencentes às entidades que a compõem. A abrangência dos recursos citados refere-se tanto àqueles ligados às entidades em caráter permanente quanto temporário;
- Esta política deve ser comunicada para todo o pessoal envolvido e largamente divulgada através das entidades, garantindo que todos tenham consciência da mesma e a pratiquem na organização;
- Todo o pessoal deve receber as informações necessárias para cumprir adequadamente o que está determinado na política de segurança;

- Um programa de conscientização sobre segurança da informação deverá ser implementado para assegurar que todo o pessoal seja informado sobre os potenciais riscos de segurança e exposição a que estão submetidos os sistemas e operações das entidades. Especificamente, o pessoal envolvido ou que se relaciona com os usuários deve estar informado sobre ataques típicos de engenharia social e como se proteger deles;
- Os procedimentos deverão ser documentados e implementados para garantir que quando o pessoal contratado ou prestadores de serviços sejam transferidos, remanejados, promovidos ou demitidos, todos os privilégios de acesso aos sistemas, informações e recursos sejam devidamente revistos, modificados ou revogados;

Ameaças, ataques e análise de vulnerabilidade

A análise de risco é fundamental para a segurança de qualquer organização, pois garante que os controles e gastos sejam comensuráveis com os riscos expostos por ela. Definem-se os seguintes conceitos:

- **Ameaça:** um evento que causa dano a empresa.
- **Vulnerabilidades:** são aberturas, fraquezas ou falta de controles da empresa às ameaças
- **Ativos:** são os bens corporativos
- **Risco:** é definido como um contexto que inclui as ameaças, vulnerabilidades e o valor a proteger (ativo)

A análise de risco é usada para analisar, estudar, avaliar as vulnerabilidades e a falta de controle que pode gerar ameaças para a empresa. E essa analise não deve ser baseada somente nas tecnologias mas também em todos os processos e pessoas que fazem parte da instituição. O gerenciamento de risco é dividido em 4 (quatro) etapas básicas:

- **Identificação dos Riscos:** como o próprio nome já diz, nessa etapa são identificados os riscos a que o negócio (o foco sempre deve ser este) está sujeito. O primeiro passo é a realização de uma Análise de Riscos, que pode ser tanto *quantitativa* – baseada em estatísticas, numa análise histórica dos registros de incidentes – quanto *qualitativa* – baseada em know-how, e realizada por especialistas, que têm profundos conhecimentos sobre o assunto.
- **Quantificação dos Riscos:** nessa etapa é mensurado o impacto que um determinado risco pode causar ao negócio. Como é praticamente impossível oferecer proteção total contra todas as ameaças existentes, é preciso identificar os ativos e as vulnerabilidades mais críticas, possibilitando a priorização dos esforços e os gastos com segurança. Uma das ferramentas existentes no mercado é o BIA (*Business Impact Analysys*). Esta técnica consiste, basicamente, da estimativa de prejuízos financeiros decorrentes da paralisação de um serviço. Responde a questões do tipo "quanto sua empresa deixaria de arrecadar caso um sistema estivesse indisponível durante 2 horas?"
- **Tratamento dos Riscos:** uma vez que os riscos foram identificados e a organização definiu quais serão tratados, as medidas de segurança devem ser de fato implementadas. Alguns riscos podem ser eliminados, outros reduzidos ou até mesmo aceitos pela empresa, tendo sempre a situação escolhida documentada. Só não é permitido ignorá-los. Nessa etapa podem ser definidas medidas adicionais de segurança, como os *Planos de Continuidade dos Negócios*.
- **Monitoração dos Riscos:** através de uma monitoração constante, é possível identificar quais áreas foram bem sucedidas e quais precisam de revisões e ajustes. O ideal é que este trabalho seja norteado por um modelo de Gestão de Segurança, que defina atribuições, responsabilidades e fluxos de comunicação interdepartamentais.

Auditoria de Sistemas e Soluções baseadas em Tecnologia da Informação

Auditoria de segurança possui o objetivo de assegurar o cumprimento dos padrões definidos e, conseqüentemente, medir a eficácia da estratégia de segurança adotada. Ela divide-se em **interna** e **externa**.

A auditoria interna é realizada com recursos materiais e pessoas que pertencem à empresa auditada. A auditoria interna existe por expressa decisão da empresa, que pode optar por sua dissolução em qualquer momento.

Por outro lado, a auditoria externa é realizada por pessoas afins à empresa auditada; Se pressupõe uma maior objetividade que na auditoria interna, devido ao maior distanciamento entre auditores e auditados

Um sistema completo de segurança deve compreender:

- Elementos administrativos
- Definição de política de segurança
- Organização e divisão de responsabilidades

- Segurança física e contra-catástrofes (incêndio, terremotos, etc.)
- Práticas de segurança do pessoal
- Elementos técnicos e procedimentos
- Sistemas de segurança (de equipes e de sistemas, incluindo todos os elementos, tanto redes como terminais).
- Aplicação dos sistemas de segurança, incluindo datas e arquivos
- O papel dos auditores, tanto internos como externos
- Planos de continuidade de negócios

Algumas ferramentas e técnicas para a auditoria de sistemas são:

- **Questionários:** enviado para pessoas de uma área específica, para recolher informações sobre o seu funcionamento, não precisando ser os responsáveis oficiais pela área.
- **Entrevistas:** uma das atividades pessoais mais importantes do auditor, que pode obter as informações mais sensíveis sobre o auditado. Uma entrevista bem preparada e sistematizada é fundamental para a eficácia desta etapa.
- ***Checklist:*** o auditor deve aplicar um *checklist* de modo que o auditado responda ou forneça as informações entre um *range* de respostas (por exemplo, 1-5, ou Muito Deficiente, Deficiente, Razoável, Aceitável e Excelente), ou em respostas binárias (Verdadeiro / Falso ou Sim / Não).
- **Software de Auditoria:** com freqüência o auditor necessita verificar se os programas e sistemas realizam as funções previstas, além de medir o nível de segurança e outros requisitos. Para isso existem *softwares* específicos, que podem medir a vulnerabilidade da rede, gerar *logs*, gráficos e planilhas.
    - *IDS (intrusion detection system)*: um dos mecanismos utilizados por esses sistemas é a detecção por assinatura, em que a assinatura típica de um tráfego malicioso permite identificá-lo como um ataque.
    - *Nessus:* uma ferramenta largamente empregada na Internet para análise de vulnerabilidades em sistemas operacionais é o NESSUS, que permite realizar diagnóstico de vários tipos de dispositivos instalados em redes que utilizam o protocolo TCP/IP. Ele permite que se faça isso de uma forma segura, não permitindo que usuários não autorizados possam escanear sua rede com ele. Ele é composto por duas partes, sendo um cliente e um servidor. Pode-se usá-lo a partir do Windows, Linux e Unix.

Requisitos de Segurança

Os princípios básicos de segurança em sistemas são (CIDA):

- **Confidencialidade:** proteção da informação compartilhada contra acessos não autorizados; obtém-se pelo controle de acesso (senhas) e controle das operações individuais de cada usuário (*log*).
- **Integridade:** garantia da veracidade da informação, que não pode ser corrompida por alterações acidentais ou não autorizadas, no caso de adulterações após a assinatura de um documento, por exemplo.
- **Disponibilidade:** prevenção de interrupções na operação de todo o sistema (hardware + software); uma quebra do sistema não deve impedir o acesso aos dados
- **Autenticidade:** garantia da identidade dos usuários. Quando aplicados à documentos digitais, consiste também de:
    - **Irrefutabilidade:** é o requisito de segurança que garante a impossibilidade de que uma autoria de documento eletrônico seja negada posteriormente.
    - **Tempestividade:** a qual nos permite saber com total segurança se determinado documento foi ou não produzido naquela ocasião.

Certificação digital

As técnicas de certificação digital oferecem seis tipos de serviços básicos. Sem estes predicados não é possível realizar o comércio eletrônico seguro na Internet:

- **Disponibilidade:** garante que uma informação estará disponível para acesso no momento desejado.
- **Integridade:** garante que o conteúdo da mensagem não foi alterado.

- **Confidencialidade (privacidade ou sigilo):** impede que pessoas não autorizadas tenham acesso ao conteúdo da mensagem, garantindo que apenas a origem e o destino tenham conhecimento.
- **Autenticidade da origem:** garante a identidade de quem está enviando a mensagem.
- **Não-repudiação:** previne que alguém negue o envio e/ou recebimento de uma mensagem.
- **Controle de acesso:** garante que o conteúdo da mensagem somente será acessado por pessoas autorizadas.

Criptografia

O ciframento de uma mensagem baseia-se em dois componentes: um **algoritmo** e uma **chave**.

- **Algoritmo:** é uma transformação matemática. Ele converte uma mensagem em claro em uma mensagem cifrada e vice-versa. Quando o remetente (origem) cifra uma mensagem, ela utiliza um algoritmo de ciframento para transformar o conteúdo em claro da mensagem em texto cifrado. Quando o destinatário decifra uma mensagem, ele utiliza o algoritmo de deciframento correspondente para converter o texto cifrado de novo em uma mensagem clara.
- **Chave:** é uma cadeia aleatória de bits utilizada em conjunto com um algoritmo. Cada chave distinta faz com que o algoritmo trabalhe de forma ligeiramente diferente.

Embora existam algoritmos que dispensem o uso de chaves, sua utilização oferece duas importantes vantagens. A primeira é permitir a utilização do mesmo algoritmo criptográfico para a comunicação com diferentes receptores, apenas trocando a chave. A segunda vantagem é permitir trocar facilmente a chave no caso de uma violação, mantendo o mesmo algoritmo.

O número de chaves possíveis depende do tamanho (número de bits) da chave. Por exemplo, uma chave de 8 bits permite uma combinação de no máximo 256 chaves (2^8). Quanto maior o tamanho da chave, mais difícil quebrá-la, pois estamos aumentando o número de combinações. A segurança passa a residir na chave empregada, e não no algoritmo utilizado.

Criptografia Simétrica

A criptografia simétrica ocorre quando a chave de ciframento é a mesma utilizada para deciframento ou esta última pode facilmente ser obtida a partir do conhecimento da primeira, e ambas precisam ser compartilhadas previamente entre origem e destinatário, antes de se estabelecer o canal criptográfico desejado.

- **DES (*Data Encryption Standard*):** é o algoritmo simétrico mais disseminado no mundo. Foi criado pela IBM em 1977 e, apesar de permitir cerca de 72 quadrilhões de combinações (2^{56}), seu tamanho de chave é considerado pequeno, tendo sido quebrado por "força bruta" em 1997.
- **Triple DES (112 ou 168 bits):** é uma simples variação do DES, utilizando-o em três cifamentos sucessivos, emprega uma versão com duas ou três chaves diferentes. É seguro, porém muito lento para ser um algoritmo padrão.
- **IDEA (*International Data Encryption Algorithm*):** de 128 bits, é estruturado seguindo as mesmas linhas gerais do DES. Mas na maioria dos microprocessadores, uma implementação por software do IDEA é mais rápida do que uma implementação por software do DES. O IDEA é utilizado principalmente no mercado financeiro e no PGP (e-mail).
- ***Blowfish* (32 a 448 bits):** oferece a escolha entre maior segurança ou desempenho através de chaves de tamanho variável. O autor aperfeiçoou-o no Twofish (proposta para um novo padrão do DES, chamado de AES).
- **RC2, RC4, RC5 (8 a 1024 bits):** é voltado para criptografia de e-mail corporativo. Também possui chave de tamanho variável. Também é utilizado no protocolo S/MIME.
- **Rijndael:** o NIST lançou um concurso para selecionar um novo algoritmo para ser adotado como o AES - Advanced Encryption Standard, em substituição do DES. O vencedor foi o Rijndael, criado por belgas e que pode utilizar chaves de 128, 192 e 256 bits.

Apesar de sua simplicidade, existem alguns problemas na criptografia simétrica:

- Como cada par necessita de uma chave para se comunicar de forma segura, para um uma rede de n usuários precisaríamos de algo da ordem de n^2 chaves, quantidade esta que dificulta a gerência das chaves;
- A chave deve ser trocada entre as partes e armazenada de forma segura, o que nem sempre é fácil de ser garantido;
- Não garante a identidade de quem enviou ou recebeu a mensagem (autenticidade e não-repudiação).

Criptografia Assimétrica

Também conhecida como criptografia de chave pública, a criptografia assimétrica está baseada no conceito de par de chaves: uma chave privada e uma chave pública. Qualquer uma das chaves é utilizada para cifrar uma mensagem e a outra para decifrá-la. As mensagens cifradas com uma das chaves do par só podem ser decifradas com a outra chave correspondente. A chave privada deve ser mantida secreta, enquanto a chave pública disponível livremente para qualquer interessado.

A grande vantagem deste sistema é permitir que qualquer um possa enviar uma mensagem secreta, apenas utilizando a chave pública de quem irá recebê-la. Como a chave pública está amplamente disponível, não há necessidade do envio de chaves como é feito no modelo simétrico. A confidencialidade da mensagem é garantida, enquanto a chave privada estiver segura.

- **RSA (Rivest, Shamir, Adleman):** é o algoritmo de chave pública mais amplamente utilizado, além de ser uma das mais poderosas formas de criptografia de chave pública conhecidas até o momento. O RSA utiliza números primos. A premissa por trás do RSA é que é fácil multiplicar dois números primos para obter um terceiro número, mas muito difícil recuperar os dois primos a partir daquele terceiro número. Isto é conhecido como **fatoração**.
- **El Gamal:** também é um sistema comutativo, como o RSA. O algoritmo envolve a manipulação matemática de grandes quantidades numéricas. Sua segurança advém de algo denominado problema do logaritmo discreto. Assim, o ElGamal obtém sua segurança da dificuldade de se calcular logaritmos discretos em um corpo finito
- **Diffie-Hellman:** também baseado no problema do logaritmo discreto, é o criptosistema de chave pública mais antigo ainda em uso. Contudo, ele não permite nem ciframento nem assinatura digital. O sistema foi projetado para permitir a dois indivíduos entrarem em um acordo ao compartilharem um segredo tal como uma chave.
- **Curvas Elípticas:** consiste em modificações de outros sistemas (o ElGamal, por exemplo), que passam a trabalhar no domínio das curvas elípticas, em vez de trabalharem no domínio dos corpos finitos. Eles possuem o potencial de proverem sistemas criptográficos de chave pública mais seguros, com chaves de menor tamanho.

Hashing

A assinatura digital obtida através do uso da criptografia assimétrica (ou de chave pública) infelizmente não pode ser empregada, na prática, de forma isolada, devido à lentidão dos algoritmos assimétricos, em geral cerca de 1.000 vezes mais lentos do que os simétricos. Assim, na prática é inviável e contraproducente utilizar puramente algoritmos de chave pública para assinaturas digitais, principalmente quando se deseja assinar grandes mensagens, que podem levar minutos ou até horas.

Ao invés disso, é empregada uma função **Hashing**, que gera um valor pequeno, de tamanho fixo, derivado da mensagem que se pretende assinar, de qualquer tamanho. Assim, a função *Hashing* oferece agilidade nas assinaturas digitais, além de integridade confiável.

A função *Hashing* funciona como uma impressão digital de uma mensagem gerando, a partir de uma entrada de tamanho variável, um valor fixo pequeno: o *digest* ou *valor hash*. Funciona como um *check sum* de uma mensagem.

- **MD5 (*Message Digest 5*):** é uma função de espalhamento unidirecional, que produz um valor hash de 128 bits, para uma mensagem de entrada de tamanho arbitrário. Foi projetado para ser rápido, simples e seguro. Seus detalhes são públicos, e têm sido analisados pela comunidade de criptografia. Existiram outras versões (MD4, MD2, ...)
- **SHA-1 (*Secure Hash Algorithm*):** é uma função de espalhamento unidirecional inventada pela NSA, gera um valor *hash* de 160 bits, a partir de um tamanho arbitrário de mensagem. Atualmente, não há nenhum ataque de criptoanálise conhecido contra o SHA-1. Mesmo o ataque da força bruta torna-se impraticável, devido ao seu valor *hash*.

Protocolos Criptográficos

Utilizam um esquema híbrido entre a criptografia simétrica (rápida, criptografa a mensagem) e a assimétrica (lenta, distribuição de chaves e assinatura digital).

- **IPSec:** padrão de protocolos criptográficos desenvolvidos para o IPv6. Realiza também o tunelamento sobre IP. É composto de três mecanismos criptográficos (assinatura digital, ciframento simétrico e troca de chaves). Criptografia e tunelamento são independentes. Permite *Virtual Private Network* fim-a-fim. Futuro padrão para as VPNs.
- **SSL e TLS:** oferecem suporte de segurança criptográfica para os protocolos NTTP, HTTP, SMTP e Telnet. Permitem utilizar diferentes algoritmos simétricos, *hashing* e métodos de autenticação e gerência de chaves (assimétricos).
- **PGP:** é um programa criptográfico famoso e bastante difundido na Internet, destinado a criptografia de e-mail pessoal. Algoritmos suportados: *hashing*: MD5, SHA-1, CAST-128, IDEA e 3DES, RSA e Diffie-Hellman/DSS.

- **S/MIME:** consiste em um esforço de um consórcio de empresas, liderado pela RSADSI e pela Microsoft, para adicionar segurança a mensagens eletrônicas no formato MIME. Apesar do S/MIME e PGP serem ambos padrões Internet, o S/MIME deverá se estabelecer no mercado corporativo, enquanto o PGP no mundo do mail pessoal.
- **SET:** é um conjunto de padrões e protocolos, para realizar transações financeira seguras, como as realizadas com cartão de crédito na Internet. Oferece um canal de comunicação seguro entre todos os envolvidos na transação. Garante autenticidade X.509v3 e privacidade entre as partes.
- **X.509:** define o relacionamento entre as autoridades de certificação. Faz parte das séries X.500 de recomendações para uma estrutura de diretório global, baseada em nomes distintos para localização. Utilizado pelo S/MIME, IPSec, SSL/TLS e SET. Baseado em criptografia com chave pública (RSA) e assinatura digital (com hashing).

Assinatura Digital

Outro benefício da criptografia com chave pública é a assinatura digital, que permite garantir a autenticidade de quem envia a mensagem, associada à integridade do seu conteúdo. Isso é feito usando a chave privada para CIFRAR a mensagem, e a chave pública para DECIFRÁ-LA. Assim, qualquer um pode ler a mensagem, e quem possui a chave secreta pode escrevê-la.

É importante perceber que a assinatura digital não garante a confidencialidade da mensagem. Qualquer um poderá acessá-la e verificá-la, mesmo um intruso, apenas utilizando a chave pública. Para obter confidencialidade com assinatura digital, basta combinar os dois métodos (i.e., assinatura digital com chave pública + criptografia assimétrica).

Os algoritmos utilizados são o **RSA**, o **El Gamal** (já descritos) e o **DSA** (*Digital Signature Algorithm*), unicamente destinado a assinaturas digitais, trata-se de uma variação dos algoritmos de assinatura ElGamal e Schnorr, e foi inventado pela NSA.

Além disso, existem os **Certificados Digitais**, que se propõe a impedir que a chave pública fornecida por uma entidade seja proveniente dela mesma, evitando que cada lado pense que está se comunicando com o outro, quando na verdade estão sendo interceptados pelo intruso. Tais certificados consistem em chaves públicas assinadas por uma pessoa de confiança, geralmente no formato padrão ITU X.509v3. Servem para evitar tentativas de substituição de uma chave pública por outra.

- **CA (*Certification Authority*):** validam outros certificados; são auto-assinados ou assinados por outra CA.
- **Servidor:** utilizados para identificar um servidor seguro; contém o nome da organização e o nome DNS do servidor.
- **Pessoais:** contém nome do portador e, eventualmente, informações como endereço eletrônico, endereço postal, etc.
- **Desenvolvedores de software:** utilizados para validar assinaturas associadas a programas.

Autoridades de certificação, como *Verisign*, *Cybertrust* e *Nortel*, assinam certificados digitais garantindo sua validade. Uma CA também tem a responsabilidade de manter uma lista com os certificados revogados (*Certificate Revocation List - CRL*)

Discursivas possíveis:

Apresente os principais operadores da Álgebra booleana. Qual a importância da representação binária em sistemas computacionais?

Fale sobre a numeração binária e hexadecimal, e dê exemplos das diferentes informações que um número binário pode representar.

Explique como um programa pode detectar se houve "overflow" ou "underflow" ao adicionar dois números em numeração binária.

Compare e explique as arquiteturas CISC e RISC, explicando os argumentos a favor de cada uma.

O que é DMA, e como o computador o utiliza em conjunto com o sistema operacional? Quais são os dispositivos periféricos que o utilizam, e exemplifique a sua utilização.

Conceitue e dê exemplos onde pode ocorrer uma interrupção de software e uma de hardware.

Cite três estruturas de endereçamento das instruções de máquina, a sua interação com os registradores e explique o seu funcionamento.

Compare as vantagens e desvantagens de se utilizar um Compilador ou Interpretador para a execução de um programa de computador.

Compare e os sistemas multiprogramados e monoprogramados. Quais as suas características?

Explique se existe alguma diferença conceitual entre mainframes, mini e microcomputadores?

Conceitue sistema operacional. Qual é a sua importância para um sistema computacional?

Quais são os passos que o sistema operacional toma ao escalonar um processo? Explique.

Descreva o escalonamento de processos Round Robin, para os seguintes processos...

Quais as vantagens e desvantagens da paginação, segmentação e paginação com segmentação?

Descreva um algoritmo de alocação de páginas na memória. Use-o para as seguintes páginas...

Descreva os sistemas de arquivos utilizados pelos sistemas Windows e Unix. Faça uma comparação.

De que formas um sistema operacional pode fazer a comunicação com um dispositivo de hardware?

Dê um exemplo onde pode ocorrer *deadlock*, e dê uma solução para o problema de concorrência.

Quais são os principais objetivos e benefícios de uma rede de computadores?

Explique a diferença entre transmissão ponto-a-ponto e redes de difusão, e entre redes com transmissão síncrona e assíncrona.

Descreva a arquitetura OSI ou TCP/IP e explique as funções das camadas.

Descreva um algoritmo de recepção / transmissão de pacotes IP.

Explique o que significa DNS, e qual a sua importância para a Internet.

Compare os métodos funcionais / de dados com os métodos OO, como é feito a modelagem, aspectos como a facilidade de compreensão do domínio, a sua robustez e facilidade de manutenção.

Faça uma modelagem DFD, dado um sistema

Modele um sistema utilizando UML: os seus requisitos (casos de uso), um modelo de domínio (classes) com classes, relações entre classes, atributos e restrições, e o ciclo de vida de um objeto (diagrama de estados).

Faça uma modelagem MER (Entidade-Relacionamento), dado um sistema

Coloque na xFN (Forma Normal) a(s) seguinte(s) tabela(s)

Descreva um exemplo de estratégia de implementação de gestão de qualidade, as atividades e os seus objetivos.

Cite pontos fracos e fortes de cada um dos modelos de ciclo de vida da Engenharia de Software

Caracterize cada uma das principais atividades do processo de Engenharia de Requisitos.

Em que consistem inspeções de software? Qual a sua relevância em processos de validação e verificação de software?

MATERIAL COMPLEMENTAR

SISTEMAS DE COMPUTAÇÃO

Organização de Computadores

http://www.organizacaodecomputadores.kit.net/

ENGENHARIA DE SOFTWARE

Pressman Software Engineering Resources

http://www.rspa.com/spi/

JAVA E WEB

Java e Web para Concursos

Rafael B. Pereira

http://www.rbper.com/java-e-web-para-concursos

http://www.agbook.com.br/book/26055--Java_e_Web_para_Concursos

JAVA OO

Introdução à Programação Orientada a Objetos usando Java

Rafael Santos

http://www.elsevier.com.br/site/produtos/Detalhe-produto.aspx?tid=1439&seg=3&isbn=9788535212068&cat=8&origem=Busca

BANCO DE DADOS

Sistemas de Bancos de Dados

http://www.submarino.com.br/produto/1/191303/sistemas+de+banco+de+dados?menuId=1324

MINERAÇÃO DE DADOS

Inteligência Artificial

Stuart Russell e Peter Norvig.

Editora Campus, 2004.