Universidade do Estado de Minas Gerais
Fundação Educacional de Ituiutaba
Curso de Engenharia da Computação
Engenharia de Software

APOSTILA
ENGENHARIA DE SOFTWARE

Prof. Walteno Martins Parreira Júnior
www.waltenomartins.com.br
walteno@ituiutaba.uemg.br
waltenomartins@yahoo.com

SUMÁRIO

1 SOFTWARE E ENGENHARIA DE SOFTWARE

1.1 Introdução

No inicio da década de 1980, uma reportagem de primeira pagina da revista Business Week apregoava a seguinte manchete: "Software: A Nova Força Propulsora". O software amadurecera - tornara-se um tema de preocupação administrativa. Em meados da década de 1980, uma reportagem de capa da Fortune lamentava "Uma Crescente Defasagem de Software" e, ao final da década, a Business Week avisava os gerentes sobre a "Armadilha do Software - Automatizar ou Não?" . No começo da década de 1990, uma reportagem especial da Newsweek perguntava: "Podemos Confiar em Nosso Software?" enquanto o Wall Street Journal relacionava as "dores de parto" de uma grande empresa de software com um artigo de primeira página intitulado "Criar Software Novo: Era Uma Tarefa Agonizante..." . Essas manchetes, e muitas outras iguais a elas, eram o anuncio de uma nova compreensão da importância do software de computador - as oportunidades que ele oferece e os perigos que apresenta.

O software agora ultrapassou o hardware como a chave para o sucesso de muitos sistemas baseados em computador. Seja o computador usado para dirigir um negocio, controlar um produto ou capacitar um sistema, o software é um fator que diferencia. A inteireza e a oportunidade. das informações oferecidas pelo software (e bancos de dados relacionados) diferenciam uma empresa de suas concorrentes. O projeto e a capacidade de ser "amigável ao ser humano" (human-friendly) de um produto de software diferenciam-no dos produtos concorrentes que tenham função idêntica em outros aspectos. A inteligência e a função oferecidas pelo software muitas vezes diferenciam dois produtos de consumo ou industrias idênticas. E o software que pode fazer a diferença.

1.2 Software

Há 20 anos, menos de 1% do público poderia descrever de forma inteligível o que significa "software de computador". Hoje, a maioria dos profissionais bem como a maior parte do público, acham que entendem o que é software. Será que entendem?

Uma descrição de software num livro didático poderia assumir a seguinte forma: "Software é: (1) instruções (programas de computador) que, quando executadas, produzem a função e o desempenho desejados; (2) estruturas de dados que possibilitam que os programas manipulem adequadamente a informação; e (3) documentos que descrevem a operação e o uso dos programas". Não há duvida de que outras definições, mais completas, poderiam ser oferecidas. Mas precisamos de algo mais que uma definição formal.

O software é um elemento de sistema lógico, e não físico. Portanto, o software tem características que são consideravelmente diferentes das do hardware:

1. O software é desenvolvido e projetado por engenharia, não manufaturado no sentido clássico.
2. O software não se desgasta.
3. A maioria dos softwares é feita sob medida em vez de ser montada a partir de componentes existentes.

1.3 Problemas associados ao software

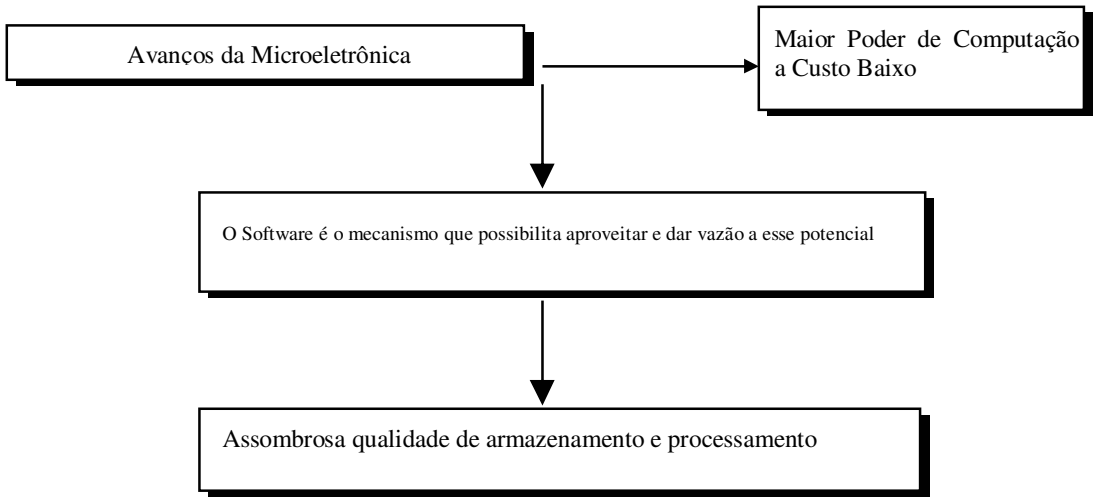
Existem um conjunto de problemas associados ao software que vários autores chamam de "crise do software". Este conjunto de problemas não se limitam ao software que não funciona adequadamente, mas ab range também prob lemas associados a forma de desenvolvimento destes softwares, a forma como é efetuada a manutenção destes softwares, como atender a demanda por novos softwares e como desenvolver novos softwares cada vez mais rapidamente.

Os problemas que atingem o desenvolvimento podem ser descritos como:

- Estimativas de prazos e de custos que são frequentemente imprecisos;
- Produtividade dos profissionais da área que não tem acompanhado a demanda por novos serviços;
- A qualidade do software desenvolvido que é insuficiente.

1.4 A Importância do Software

Durante as três primeiras décadas da era do computador, o principal desafio era desenvolver um hardware que reduzisse o custo de processamento e armazenagem de dados. Ao longo da década de 1980, avanços na microeletrônica resultaram em maior poder de computação a um custo cada vez mais baixo. Hoje, o problema é diferente. O principal desafio durante a década de 1990 é melhorar a qualidade (e reduzir o custo) de soluções baseadas em computador - soluções que são implementadas com software.



O poder de um computador mainframe da década de 1980 agora está a disposição sobre uma mesa. As assombrosas capacidades de processamento e armazenagem do moderno hardware representam um grande potencial de computação. O software é o mecanismo que nos possibilita aproveitar e dar vazão a esse potencial.

1.5 O Papel Evolutivo do Software

O contexto em que o software foi desenvolvido está estreitamente ligado a quase cinco décadas de evolução dos sistemas computadorizados. O melhor desempenho de hardware, menor tamanho e custo mais baixo precipitaram o aparecimento de sistemas baseados em computadores mais sofisticados. Mudamos dos processadores a válvula para os dispositivos

microeletronicos que são capazes de processar 200 milhões de instruções por segundo. Em livros populares sobre a "revolução do computador", Osborne caracterizou uma "nova revolução industrial", Toffler chamou o advento da microeletronica de "a terceira onda de mudança" na historia humana e Naisbitt previu que a transformação de uma sociedade industrial em uma "sociedade de informação" terá um profundo impacto sobre nossas vidas. Feigenbaum e McCorduck sugeriram que a informação e o conhecimento (controlados por computador) serão o foco principal de poder no século XXI, e Stoll argumentou que a "comunidade eletrônica" criada por redes e software e a chave para a troca de conhecimentos em todo o mundo. Quando se iniciava a década de 1990, Toffler descreveu uma "mudança de poder", em que as velhas estruturas de poder (governamental, educacional, industrial, econômico e militar) se desintegrarão enquanto os computadores e o software levarão a uma "democratização do conhecimento" .

Primeiros Anos 1950 a 1960	**Segunda Era 1960 a 1970**	**Terceira Era 1970 a 1980**	**Quarta Era 1980 a 2000**
Orientação Batch	Multiusuário Interativo	Sistemas Distribuídos	Sistemas de Desktop poderosos
Distribuição Limitada	Tempo Real	Hardware de Baixo Custo	Tecnologias Orientadas à Objetos
Software Customizado	Banco de Dados	Microprocessadores	Sistemas Especialistas
Programação Artesanal	Produto de Software	Impacto de Consumo	Computação paralela
Sem Administração Específica	Software House	“inteligência” embutida	Ferramentas CASE
Sem Documentação			Reutilização
			Redes Neurais artificiais

A tabela descreve a evolução do software dentro do contexto das áreas de aplicação de sistemas baseados em computador. Durante os primeiros anos do desenvolvimento de sistemas computadorizados, o hardware sofreu continuas mudanças, enquanto o software era visto por muitos como uma reflexão posterior. A programação de computador era uma arte "secundaria" para a qual havia poucos métodos sistemáticos. O desenvolvimento do software era feito virtualmente sem administração - ate que os prazos começassem a se esgotar e os custos a subir abruptamente. Durante esse período, era usada uma orientação batch (em lote) para a maioria dos sistemas. Notáveis exceções foram os sistemas interativos, tais como o primeiro sistema de reservas da American Airlines e os sistemas de tempo real orientados a defesa, como o SAGE. Na maior parte, entretanto, o hardware dedicava-se a execução de um único programa que, por sua vez, dedicava-se a uma aplicação específica.

Durante os primeiros anos, o hardware de propósito geral tornara-se lugar comum. O software, por outro lado, era projetado sob medida para cada aplicação e tinha uma distribuição relativamente limitada. O produto software (isto e, programas desenvolvidos para serem vendidos a um ou mais clientes) estava em sua infância. A maior parte do software era desenvolvida e, em ultima analise, usada pela própria pessoa ou organização. Você escrevia-o, colocava-o em funcionamento e, se ele falhasse, era você quem o consertava. Uma vez que a rotatividade de empregos era baixa, os gerentes podiam dormir tranqüilos com a certeza de que você estaria lá se defeitos fossem encontrados.

Por causa desse ambiente de software personalizado, o projeto era um processo implícito realizado no cérebro de alguém, e a documentação muitas vezes não existia. Durante os primeiros anos, aprendemos muito sobre a implementação de sistemas baseados em computador, mas relativamente pouco sobre engenharia de sistemas de computador. Por justiça, entretanto, devemos reconhecer os muito surpreendentes sistemas baseados em computador desenvolvidos durante essa época. Alguns deles permanecem em uso ate hoje e constituem feitos que são um marco de referência e que continuam a justificar a admiração.

A segunda era da evolução dos sistemas computadorizados estendeu-se de meados da década de 1960 até o final da década de 1970. A multiprogramação e os sistemas multiusuários introduziram novos conceitos de interação homem-máquina. As técnicas interativas abriram um novo mundo de aplicações e novos níveis de sofisticação de software e hardware. Sistemas de tempo real podiam coletar, analisar e transformar dados de múltiplas fontes, dai controlando processos e produzindo saída em milissegundos, e não em minutos. Os avanços da armazenagem on-line levaram à primeira geração de sistemas de gerenciamento de bancos de dados.

A segunda era também foi caracterizada pelo uso do produto de software e pelo advento das "software houses". O software era desenvolvido para ampla distribuição num mercado interdisciplinar. Programas para mainframes e minicomputadores eram distribuídos para centenas e, às vezes, milhares de usuários. Empresários da industria, governos e universidades puseram-se "desenvolver pacotes de software" e a ganhar muito dinheiro.

A medida que o numero de sistemas baseados em computador crescia, bibliotecas de software de computador começaram a se expandir. Projetos de desenvolvimento internos nas empresas produziram dezenas de milhares de instruções de programa. Os produtos de software comprados no exterior acrescentaram centenas de milhares de novas instruções. Uma nuvem negra apareceu no horizonte. Todos esses programas - todas essas instruções - tinham de ser corrigidos quando eram detectadas falhas, alterados conforme as exigências do usuário se modificavam ou adaptados a um novo hardware que fosse comprado. Essas atividades foram chamadas coletivamente de manutenção de software. O esforço despendido na manutenção de software começou a absorver recursos em índices alarmantes.

E, ainda pior, a natureza personalizada de muitos programas tornava-os virtualmente impossíveis de sofrer manutenção. Uma "crise de software" agigantou-se no horizonte.

A terceira era da evolução dos sistemas computadorizados começou em meados da década de 1970 e continua ate hoje. Os sistemas distribuídos - múltiplos computadores, cada um executando funções concorrentemente e comunicando-se um com o outro - aumentaram intensamente a complexidade dos sistemas baseados em computador. As redes globais e locais, as comunicações digitais de largura de banda (handwidth) elevada e a crescente demanda de acesso "instantâneo" a dados exigem muito dos desenvolvedores de software.

A terceira era também foi caracterizada pelo advento e generalização do uso de microprocessadores, computadores pessoais e poderosas estações de trabalho (workstations) de mesa. O microprocessador gerou um amplo conjunto de produtos inteligentes - de automóveis a fornos de microondas, de robôs industriais a equipamentos para diagnostico de soro sangüíneo. Em muitos casos, a tecnologia de software esta sendo integrada a produtos por equipes técnicas que entendem de hardware mas que freqüentemente são principiantes em desenvolvimento de software.

O computador pessoal foi o catalisador do crescimento de muitas empresas de software. Enquanto as empresas de software da segunda era vendiam centenas ou milhares de copias de seus programas, as empresas da terceira era vendem dezenas e ate mesmo centenas de milhares de cópias. O hardware de computador pessoal está se tornando rapidamente um produto primário, enquanto o software oferece a característica capaz de diferenciar. De fato, quando a taxa de crescimento das vendas de computadores pessoais se estabilizou em meados da década de 1980, as vendas de software continuaram a crescer. Na industria ou em âmbito domestico, muitas pessoas gastaram mais dinheiro cm software do que aquilo que despenderam para comprar o computador no qual o software seria instalado.

A quarta era do software de computador esta apenas começando. As tecnologias orientadas a objetos estão rapidamente ocupando o lugar das abordagens mais convencionais para o desenvolvimento de software em muitas áreas de aplicação. Autores tais como Feigenhaum, McCorduck e Allman prevêem que os computadores de "quinta geração", arquiteturas de computação radicalmente diferentes, e seu software correlato exercerão um profundo impacto sobre o equilíbrio do poder político e industrial em todo o mundo. As técnicas

de "quarta geração" para o desenvolvimento de software já estão mudando a maneira segundo a qual alguns segmentos da comunidade de software constróem programas de computador. Os sistemas especialistas e o software de inteligência artificial finalmente saíram do laboratório para a aplicação pratica em problemas de amplo espectro do mundo real. O software de rede neural artificial abriu excitantes possibilidades para o reconhecimento de padrões e para capacidades de processamento de informações semelhantes as humanas.

Quando nos movimentamos para a quinta era, os problemas associados ao software de computador continuam a se intensificar:

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

Em resposta a esses problemas, estão sendo adotadas práticas de engenharia em toda a industria.

1.6 Aplicações do Software

O software pode ser aplicado a qualquer situação em que um conjunto previamente especificado de passos procedimentais (um algoritmo) tiver sido definido. O conteúdo de informações e a previsibilidade são fatores importantes na determinação da natureza de um aplicativo.

Desenvolver categorias genéricas para as aplicações de softwares é uma tarefa muito difícil. Quanto mais complexo é o sistema, mais difícil é determinar de forma clara os vários componentes do software.

Pode-se dividir as aplicações em:

- Software Básico – que é um conjunto de programas para dar apoio a outros programas. Tem como característica uma forte interação com o hardware, operações concorrentes, compartilhamento de recursos, uso por múltiplos usuários e múltiplas interfaces.
- Software de Tempo Real – são programas que monitora, analisa e controla eventos do mundo real, devendo responder aos estímulos do mundo externo com restrições de tempo pré-determinadas.
- Software Comercial – é a maior área de aplicação de softwares, são aplicações que gerenciam as operações comerciais de modo a facilitar o gerenciamento comercial do negócio da empresa, permitindo também a tomada de decisões.
- Software Cientifico e de Engenharia – são caracterizados por algoritmos de processamento numérico, dependentes da coleta e processamento de dados para as mais variadas áreas do conhecimento.
- Software Embutido – são desenvolvidos para executar atividades muito específicas e inseridos em produtos inteligentes tanto para atividades comerciais como para atividades domesticas.
- Software de Computador Pessoal – são produtos desenvolvidos para o uso pessoal do computador, tais como planilhas eletrônicas, processadores de textos, jogos, etc..
- Software de Inteligência Artificial – faz uso de algoritmos não-numéricos para resolver

problemas complexos que não apresentam facilidades computacionais numéricas ou de análise direta.

1.7 Engenharia de Software: Uma Definição

Urna primeira definição de engenharia de software foi proposta por Fritz Bauer na primeira grande conferencia dedicada ao assunto: "O estabelecimento e uso de sólidos princípios de engenharia para que se possa obter economicamente um software que funcione eficientemente cm máquinas reais."

A engenharia de software é uma derivação da engenharia de sistemas e de hardware. Ela abrange um conjunto de três elementos fundamentais - métodos, ferramentas e procedimentos - que possibilita ao gerente o controle do processo de desenvolvimento do software e oferece ao profissional uma base para a construção de software de alta qualidade produtivamente.

Os métodos de engenharia de software proporcionam os detalhes de "como fazer" para construir o software. Os métodos envolvem um amplo conjunto de tarefas que incluem: planejamento c estimativa de projeto, análise de requisitos de software e de sistemas, projeto da estrutura de dados, arquitetura de programa e algoritmo de processamento, codificação, teste e manutenção. Os métodos da engenharia de software muitas vezes introduzem uma notação gráfica ou orientada a linguagem especial e introduzem um conjunto de critérios para a qualidade do software.

As ferramentas de engenharia de software proporcionam apoio automatizado ou semi-automatizado aos métodos. Atualmente, existem ferramentas para sustentar cada um dos métodos anotados anteriormente. Quando as ferramentas são integradas de forma que a informação criada por uma ferramenta possa ser usada por outra, é estabelecido um sistema de suporte ao desenvolvimento de software chamado engenharia de software auxiliada por computador (CASE - Computer-Aided Software Engineering). O CASE combina software, hardware e um banco de dados de engenharia de software (uma estrutura de dados contendo importantes informações sobre analise, projeto, codificação e teste) para criar um ambiente de engenharia de software que seja análogo ao projeto auxiliado por computador/engenharia auxiliada por computador para o hardware.

Os procedimentos da engenharia de software constituem o elo de ligação que mantém juntos os métodos e as ferramentas e possibilita o desenvolvimento racional e oportuno do software de computador. Os procedimentos definem a seqüência em que os métodos serão aplicados, os produtos (deliverables) que se exige que sejam entregues (documentos, relatórios, formulários etc.), os controles que ajudam a assegurar a qualidade e a coordenar as mudanças, e os marcos de referencia que possibilitam aos gerentes de software avaliar o progresso.

A engenharia de software compreende um conjunto de etapas que envolve métodos, ferramentas e os procedimentos. Essas etapas muitas vezes são citadas como paradigmas da engenharia de software. Um paradigma de engenharia de software e escolhido tendo-se como base a natureza do projeto e da aplicação, os métodos e as ferramentas a serem usados, os controles e os produtos que precisam ser entregues.

1.8 O que é engenharia de Software?

É um conjunto integrado de métodos e ferramentas utilizadas para especificar, projetar, implementar e manter um sistema.

Segundo Ariadne Carvalho & Thelma Chiossi no livro Introdução à Computação, a Engenharia de Software é “Uma disciplina que reúne metodologias, métodos e ferramentas a

ser utilizados, desde a percepção do problema até o momento em que o sistema desenvolvido deixa de ser operacional, visando resolve problemas inerentes ao processo de desenvolvimento e ao produto de software." Pode-se definir como:

- Um **método** é uma prescrição explícita de como chegar a uma atividade requerida por um modelo de ciclo de vida, visando otimizar a execução das atividades que foram especificadas.
- As **ferramentas** proporcionam apoio automatizado ou semi-automatizado aos métodos.

Os Métodos de Desenvolvimento de Sistema se diferenciam pela maneira como o problema deve ser visualizado e como a solução do problema deve ser modelada. São eles:

1.8.1 Método baseado na Decomposição de Funções:

Abordagem estruturada caracterizada pela decomposição das funções. Os tipos de modelos que representam as funções são:

- DFD (Diagrama de Fluxo de Dados) – se caracteriza pela decomposição hierárquica de processos.
- MHT (Modelo Hierárquico de Tarefas) – se baseia na decomposição hierárquica de tarefas.

1.8.2 Método baseado na Estrutura de Dados:

Abordagem baseada na decomposição de um problema a partir dos dados. Exemplos de tipos de modelos dessa classe:

- MER (Modelagem Entidade-Relacionamento)
- Técnica de Warnier

1.8.3 Método de Análise baseado na Orientação a Objeto.

Os tipos de modelos que representam essa classe são:

- UML (Unified Process) – notação de modelagem, independente de processos de desenvolvimento.
- Cenários

1.9 Paradigmas de Engenharia de Software

Segundo Roger Pressman, "Paradigmas são os modelos de processos que possibilitam:

- ao gerente: o controle do processo de desenvolvimento de sistemas de software,
- ao desenvolvedor: a obter a base para produzir, de maneira eficiente, software que satisfaça os requisitos preestabelecidos."

Os paradigmas especificam algumas atividades que devem ser executadas, assim como a ordem em que devem ser executadas. A função dos paradigmas é diminuir os problemas encontrados no processo de desenvolvimento do software, e deve ser escolhido de acordo com a natureza do projeto e do produto a ser desenvolvido, dos métodos e ferramenta a ser utilizado e dos controles e produtos intermediários desejados.

Os paradigmas serão estudados no em outro capitulo.

1.10 Processos de Software

Segundo Ian Sommerville, “Um processo de software é um conjunto de atividades e resultados associados que levam à produção de um produto de software.” Embora existam muitos processos de software diferentes, há atividades fundamentais comuns a todos eles, tais como:

- Especificação de software;
- Projeto e implementação de software;
- Validação de software;
- Evolução do software.

A melhoria dos processos de software podem ser implementadas de diferentes maneiras, como serão estudados posteriormente.

1.11 Os desafios da Engenharia de Software

Neste século XXI, os principais desafios serão:

- O desafio do legado, que é fazer a manutenção e a atualização dos softwares atuais, sem apresentar grandes custos e ao mesmo tempo prosseguir com a prestação dos serviços corporativos essenciais;
- O desafio da heterogeneidade, que refere-se a desenvolver técnicas para construir softwares confiáveis e que sejam flexíveis para lidar com diferentes tipos de equipamentos e sistemas;
- O desafio do fornecimento, que deve apresentar uma redução do tempo gasto no desenvolvimento e implantação do software sem comprometer a qualidade.

2 TÉCNICAS DE ENTREVISTAS E DE COLETA DE DADOS

2.1 Introdução

Este texto apresenta algumas diretrizes para as entrevistas que você fará durante a fase de análise de sistemas do projeto de desenvolvimento de um sistema. Provavelmente entrevistará usuários, gerentes, auditores, programadores e auxiliares que utilizam sistemas já existentes (informatizados ou manuais) e também várias outras pessoas envolvidas.

Por que fazemos entrevistas durante a análise de sistemas? Porque:

- Precisamos coletar informações sobre o comportamento de um sistema atual ou sobre os requisitos de um novo sistema de pessoas que têm essas informações armazenadas em algum lugar em suas cabeças.
- Precisamos verificar nossa própria compreensão, como analistas de sistemas, do comportamento de um sistema atual ou dos requisitos de um novo sistema. Essa compreensão deve ser adquirida através de entrevistas em combinação com informações coletadas de modo independente.
- Precisamos coletar informações sobre o(s) sistema(s) atual(is) para a execução do estudo de custo/benefício para o novo sistema.

2.2 Tipos de Entrevistas

A forma mais comum de entrevista é uma reunião pessoal e direta entre você (talvez acompanhado por até dois analistas auxiliares do projeto) e um ou mais interlocutores (entrevistados). Normalmente são efetuadas anotações por um dos entrevistadores; menos costumeiramente, a entrevista pode ser gravada ou um(a) secretário(a) tomará notas durante a entrevista.

Pode-se perceber que as informações obtidas em uma entrevista também podem ser obtidas por outros meios, por exemplo, solicitando-se que os entrevistados respondam por escrito a um questionário formal, ou solicitando que descrevam por escrito os requisitos do novo sistema. Também podemos aumentar as entrevistas pela presença de vários especialistas (que podem até conduzir a entrevista enquanto o analista de sistemas participa como assistente), como peritos em indústria/comercio, psicólogos comportamentais e negociadores. E, finalizando, deve-se lembrar que outros meios de obtenção de dados (ex.: videocassete, gravadores, etc...) podem ser utilizados para registrar uma entrevista.

Durante a década de 80, uma forma especializada de entrevista tornou-se popular em algumas empresas; conhecida como JAD (Joint Application Development) ou projeto acelerado, ou análise em equipe, e por vários outros nomes. Ela consiste em uma rápida entrevista e um processo acelerado de coleta de dados em que todos os principais usuários e o pessoal da análise de sistemas agrupam-se em uma única e intensiva reunião (que pode prolongar-se de um dia a uma semana) para documentar os requisitos do usuário. A reunião costuma ser supervisionada por um profissional experiente e bem treinado que atua como mediador para encorajar melhores comunicações entre os analistas de sistemas e os usuários. Freqüentemente, este supervisor é apoiado por algumas pessoas que se dedicam integralmente ao processo.

Embora todas essas variações tenham de fato sido utilizadas, elas são relativamente raras e não serão discutidas em maiores detalhes neste texto. A entrevista mais utilizada ainda é a reunião pessoal entre um analista de sistemas e um usuário final.

2.3 Problemas Fundamentais

Inicialmente pode parecer que o processo de entrevistar um usuário seja uma questão simples e direta. Afinal, o analista e o usuário são pessoas inteligentes e capazes de expressarem. Os dois são pessoas racionais e ambos têm o mesmo objetivo: passar informações relativas a um novo sistema proposto da mente do usuário para a do analista. Qual é o problema?

Na realidade, existem muitos problemas que podem ocorrer. Em muitos projetos de alta tecnologia, tem-se observado que a maioria dos problemas difíceis não envolvem hardware nem software, mas sim o pleopleware. Os problemas de peopleware na análise de sistemas são muitas vezes encontrados nas entrevistas. Os problemas mais comuns a que você deve estar atento são:

- **Entrevistar a pessoa errada no momento errado**. É muito fácil, por causa dos problemas e interesses organizacionais, falar com a pessoa que é o perito oficial na orientação do usuário, que demonstra nada saber a respeito dos verdadeiros requisitos do sistema; também é possível perder a oportunidade de falar com o usuário desconhecido que realmente sabe quais são os requisitos. Mesmo que encontre a pessoa certa, talvez o analista tente entrevistá-la durante um período em que o usuário não está disponível ou esteja mergulhado sob outras pressões e emergências.
- **Fazer perguntas erradas e obter respostas erradas**. A análise de sistemas é, como Tom DeMarco gosta de dizer, uma forma de comunicação entre estranhos. Os usuários e os analistas de sistemas têm vocabulários diferentes, experiências básicas diferentes e muitas vezes um diferente conjunto de pressuposições, percepções, valores e prioridades. Desse modo, é fácil fazer ao usuário uma pergunta racional sobre os requisitos do sistema e o usuário não entender absolutamente a pergunta, sem que nenhum dos dois perceba o fato. E é fácil para o usuário prestar-lhe algumas informações sobre os requisitos e o analista não entender essas informações, novamente sem que nenhum dos dois perceba o fato. As ferramentas de modelagem são uma tentativa de fornecer uma linguagem comum e sem ambigüidades para diminuir esses mal entendidos. Porém as entrevistas costumam ser realizadas em uma língua comum (inglês, espanhol, francês, português etc.),portanto o problema é real. Isso explica porque é tão importante marcar entrevistas subseqüentes para confirmar que ambas as partes entenderam as perguntas e as respostas.
- **Criar ressentimentos recíprocos**. existem algumas razões pelas quais o usuário pode não se sentir à vontade ou mesmo colocar-se em posição antagônica na entrevista com um analista de sistema (ex.: porque ele percebe que o verdadeiro objetivo do novo sistema que o analista está especificando é tomar-lhe o emprego). E o analista pode ressentir-se do modo como o usuário está respondendo as perguntas. Em qualquer caso, o ressentimento pode surgir entre as duas partes, tornando a comunicação muito mais difícil.

Não existe um modo mágico de garantir que esses problemas não ocorrerão; eles são a conseqüência das interações pessoa-a-pessoa, e cada uma dessas interações é única. Contudo, as sugestões dadas a seguir podem ajudar a reduzir a probabilidade desses problemas; fora isso, dependerá de prática para melhorar cada vez mais em cada entrevista subseqüente.

2.4 Diretrizes Para a Realização de Entrevistas

As seguintes diretrizes podem ser de grande auxílio na direção de entrevistas bem-sucedidas com o usuário.

2.4.1 Desenvolva um Plano Geral de Entrevistas

Antes de tudo, é extremamente importante que o analista descubra quem deve ser entrevistado. Caso contrário, desperdiçará o tempo de todos e criará um enorme tumulto, por falar com pessoas erradas sobre coisas erradas.

Isso requer que obtenha um organograma da empresa que mostre as pessoas da organização usuária, bem como a hierarquia entre elas. Se não existir um organograma formal, encontrar alguém que saiba como a organização funciona e pedir ajuda. Se o organograma existir, certificar-se de que esteja correto e atualizado; as empresas muitas vezes modificam-se com muito mais freqüência do que o ciclo editorial anual em que os diagramas são produzidos!

O fato de conhecer o esquema organizacional não diz necessariamente com quem o analista deve entender-se; às vezes, a pessoa que mais sabe a respeito de algum aspecto de um sistema é um funcionário administrativo ou burocrata que sequer aparece no organograma. Muitas vezes existem três níveis de usuários em uma organização grande e complexa (o usuário verdadeiro, o usuário supervisor operativo e o usuário supervisor executivo) e é muitas vezes de grande importância falar com todos os três níveis.

Em muitos casos também é importante conversar com os usuários na seqüência adequada e na combinação certa. Por vezes o analista poderá saber em que seqüência os usuários deverão ser entrevistados face a seu conhecimento geral da organização e por vezes os próprios usuários lhe dirão uma vez que saibam que serão entrevistados.

2.4.2 Certifique-se de que tem Autorização para Falar com os Usuários

Em algumas organizações informais não haverá restrições na escolha dos usuários com quem falar ou sobre como as entrevistas serão marcadas. Porém isso não é comum em empresas grandes; é politicamente perigoso vagar pela organização usuária realizando entrevistas sem prévia autorização.

Na maioria dos casos, a autorização virá ou do encarregado de um setor usuário (um departamento ou divisão) ou do representante da empresa que estará auxiliando o projeto de desenvolvimento do sistema. Em qualquer caso, os usuários têm legítimas razões em querer aprovar, antecipadamente, quem será entrevistado:

- Eles podem saber que alguns usuários não serão capazes de compreender ou exprimir bem os requisitos do sistema.
- Eles podem desconfiar de que alguns dos usuários do nível operativo sejam rebeldes que apresentarão requisitos falsos (ou requisitos que a direção não aprova).
- Eles podem recear que as entrevistas interfiram com as atribuições normais de trabalho que os usuários tenham de executar. Por causa disso, desejarão marcar as entrevistas para os momentos apropriados.
- Eles podem recear que as entrevistas sejam vistas como o início de um esforço de substituição dos usuários humanos por um sistema computadorizado, provocando demissões e coisas desse gênero.
- Eles podem considerar que eles próprios (os diretores) sabem mais a respeito dos requisitos do sistema do que qualquer outro e por isso não desejam que você entreviste qualquer usuário de nível operativo.

- Pode estar havendo uma batalha política em andamento em um nível de chefia muito mais elevado, entre o setor usuário e a organização de desenvolvimento de sistemas. Desse modo, o gerente usuário pode não ter reais objeções a suas entrevistas, porém, impedindo que elas sejam feitas, ele estará enviando uma mensagem política para o chefe do chefe do seu chefe.

Por todos esses motivos, é uma boa medida obter uma prévia autorização. Na maior parte dos casos, basta uma autorização verbal; se a organização for terrivelmente burocrática ou paranóica, pode–se precisar de uma autorização escrita. Isso, a propósito, também significa que o analista deve estar atento à política organizacional se tiver certeza da necessidade de falar com um usuário (normalmente um usuário do nível operativo) com quem tenha sido avisado para não conversar.

2.4.3 Planeje a Entrevista para Fazer Uso Eficiente do Tempo

O principal aspecto desta sugestão é que deve-se compreender que está tomando o tempo do usuário e que ele (ou o chefe dele) pode achar até que o analista esteja desperdiçando o tempo dele. Assim sendo, é importante o planejamento e preparação tão antecipadamente quanto possível para poder fazer uso eficiente da entrevista.

A primeira coisa a fazer é certificar-se de que o usuário conhece o assunto da entrevista. Em alguns casos isso pode ser feito por telefone; em outros, pode ser adequado preparar uma lista das perguntas que serão feitas, ou dos tópicos que serão abordados, ou dos DFD que necessita ser revisados, e remetê-la ao usuário com um dia ou dois de antecipação. Se não puder fazer isso, é um indício de que de fato o analista não está preparado para a entrevista. E se o usuário não tiver lido o material remetido, é sinal de que:

- está muito ocupado,
- está desinteressado,
- opõe-se a toda a idéia da entrevista ou
- é incapaz de entender as perguntas apresentadas.

Um aspecto relacionado: coletar, antes da entrevista, tantos dados pertinentes quanto possível. Se houver formulários ou relatórios que sejam pertinentes à discussão, geralmente poderá obtê-los antecipadamente. Se existirem outros documentos escritos do usuário descrevendo o novo ou o antigo sistema consiga-os e estude-os antes da entrevista.

Se tiver preparado as perguntas antecipadamente, o analista deve ser capaz de realizar a entrevista em uma hora ou menos. Isso é importante; não só o usuário é geralmente incapaz de reservar mais do que uma hora de cada vez, mas também as pessoas normalmente não conseguem se concentrar intencionalmente (principalmente se estiverem examinando diagramas um tanto estranhos) por mais do que uma hora. Isso naturalmente significa que deve-se organizar a entrevista para abranger um escopo relativamente limitado, focalizando normalmente uma parte do sistema. Isso também pode significar que tenha de marcar algumas entrevistas com o mesmo usuário para abranger inteiramente a área em que ele está envolvido.

Finalizando, marcar uma reunião subseqüente para rever o material que foi coletado. Normalmente, após a entrevista, o analista irá para sua mesa com todas as informações colhidas na entrevista, e executará bastante trabalho com os dados brutos. Pode haver DFD a serem desenhados ou itens a serem criados no dicionário de dados; cálculos de custo/benefício podem precisar ser feitos; as informações provenientes da entrevista podem precisar ser correlacionados com dados de outras entrevistas, e assim por diante. Em qualquer caso, os dados dessa entrevista serão manipulados, documentados, analisados e convertidos em uma forma que o usuário pode nunca ter visto antes. Desse modo, pode ser necessário marcar uma entrevista posterior para verificar:

- que o analista não cometeu nenhum engano em seu entendimento do que o usuário lhe disse,
- que o usuário não mudou de opinião nesse ínterim,
- que o usuário entende a notação ou a representação gráfica dessas informações.

2.4.4 Utilize Ferramentas Automatizadas que Sejam Adequadas, Mas Não Abuse

Durante a entrevista, pode-se achar conveniente utilizar ferramentas de prototipação, principalmente se o objetivo da entrevista for discutir a visão que o usuário tem da interface pessoa-máquina. De modo semelhante, se estiver revisando um diagrama de fluxo de dados e discutindo possíveis modificações, poderá achar prático usar uma ferramentas CASE.

Lembre-se, que o objetivo dessas ferramentas é facilitar as discussões e não complicá-las; elas devem permitir que o analista e o usuário examinem alternativas e modificações com rapidez e facilidade; elas podem ajudar a registrar o conhecimento de um requisito do usuário e corrigir imediatamente quaisquer erros que tenha sido cometido.

Se, a tecnologia introduzir-se no assunto, deve-se deixar fora da entrevista. Se o usuário tiver de aventurar-se além de seu ambiente normal de atividade (para outro prédio, na sala do computador) poderá encarar a ferramenta como um aborrecimento. Se o usuário não estiver familiarizado com a tecnologia de computadores e for solicitado a utilizar a ferramenta, poderá rejeitá-la. E se o analista não estiver familiarizado com a ferramenta (ou se a ferramenta for lenta, apresentar erros ou de emprego limitado) isso interferirá sensivelmente na entrevista. Em qualquer desses casos, talvez seja preferível usar a ferramenta off-line depois da entrevista; então, poderá mostrar ao usuário a saída da ferramenta sem causar quaisquer problemas desnecessários.

2.4.5 Tente Descobrir quais Informações o Usuário tem mais Interesse

Se o analista tiver de desenvolver um modelo completo de sistema para alguma parte de um sistema, possivelmente necessitará determinar entradas, saídas, funções, características tempo-dependentes e a memória armazenada do sistema. Porém a ordem em que se obtém essas informações costumam não ter muita importância. Mas pode significar muito para o usuário, e deve deixá-lo começar a entrevista por onde ele preferir. Alguns usuários desejarão começar pelas saídas, isto é, pelos relatórios e valores de dados que eles querem que o sistema produza (na realidade, eles talvez nem saibam que entradas serão necessárias para produzir as saídas desejadas). Outros usuários poderão estar mais interessados nas entradas ou nos detalhes de uma transformação funcional. Outros ainda preferirão falar sobre os detalhes dos dados de um depósito de dados. Deve-se esforçar para enxergar os requisitos do sistema da perspectiva desses usuários, e conservar esta perspectiva em mente quando fizer as perguntas necessárias sua entrevista.

2.4.6 Use um Estilo Adequado de Entrevistar

Como diz William Davis [Davis, 1983]: Sua atitude em relação à entrevista é importante na determinação de seu sucesso ou fracasso. Uma entrevista não é uma competição. Evite ataques; evite o uso excessivo do jargão técnico; conduza uma entrevista, não uma tentativa de persuasão. Fale com as pessoas, não fale muito alto, nem muito baixo, nem indiretamente. Uma entrevista não é um julgamento. Faça perguntas detalhadas, mas não faça perguntas para confirmar outras respostas. Lembre-se que o entrevistado é o perito e que é você que precisa de respostas. Para concluir, de modo algum critique a credibilidade de outras pessoas.

Fazer perguntas detalhadas nem sempre é fácil; dependendo da personalidade do entrevistado e do tema da entrevista, pode-se precisar de um conjunto de estilos para extrair as informações necessárias. Alguns estilos que podem mostrar-se úteis:

- **Relacionamentos.** Pedir ao usuário para explicar o relacionamento entre o que está em discussão e as demais partes do sistema. Se o usuário estiver falando sobre um assunto (p.ex.: um cliente), pedir-lhe que explique seu relacionamento sob outros aspectos; se ele estiver descrevendo uma função (ex.: uma bolha de um DFD), pedir-lhe que explique seu relacionamento com outras funções. Isso não só o auxiliará a descobrir mais detalhes sobre o item em pauta, mas também o ajudará a descobrir interfaces (ex.: fluxos de dados de uma bolha para outra no DFD) e relacionamentos formais.
- **Pontos de vista alternativos.** Solicitar ao usuário que descreva o ponto de vista de outros usuários em relação ao item que esteja sendo discutido. Perguntar ao usuário, por exemplo, o que seu chefe pensa sobre uma bolha do DFD, ou um tipo de objeto do DER; ou pergunte o que pensa seu subordinado.
- **Detalhamento.** Solicitar ao usuário uma informal descrição narrativa do item em que estiver interessado. Fale-me sobre o modo como você calcula o valor das remessas. Ou, se estiver falando com o usuário sobre um tipo de objeto no DER, poderia dizer-lhe: Fale-me a respeito de um cliente. O que você sabe (ou precisa saber) sobre um cliente?
- **Dependências.** Perguntar ao usuário se o item em discussão depende, para sua existência, de alguma outra coisa. Isso é especialmente útil quando se discutem possíveis tipos de objetos e relacionamentos no DER. Em um sistema de controle de pedidos, por exemplo, pode-se perguntar ao usuário se seria possível haver um pedido sem que haja um cliente.
- **Confirmação.** Dizer ao usuário o que acha que ouviu ele dizer; usar suas próprias palavras em lugar das dele e peça confirmação. Pode-se dizer: Deixe-me ver se entendi o que você disse...

2.5 *Possíveis Formas de Resistência na Entrevista*

Deve-se estar preparado para o fato de que alguns usuários serão contrários à idéia de uma entrevista; essa é uma das razões para garantir que o chefe ou alguém com autoridade no setor esteja ciente e tenha permitido a entrevista. Algumas das objeções mais comuns e algumas possíveis respostas a essas objeções:

- **Você está tomando tempo demais de mim**. A resposta a isso é explicar que compreende, e desculpar-se pelo tempo que precisará tomar, mas que já preparou tudo e fará a entrevista no tempo mais curto possível. Isso naturalmente exige que o analista chegue pontualmente na hora marcada para o início da entrevista, mantenha a discussão no rumo previsto, e encerre-a no momento em que tenha dito que o faria.
- **Você está ameaçando meu emprego**. Isso muitas vezes é uma reação emocional que pode ou não ter fundamento. Embora possa pensar em várias maneiras de responder a esse comentário, lembre-se de que o analista não é o patrão dessa pessoa e que não está em posição de garantir que o emprego dela não esteja em perigo, ou de informá-la do contrário. Ele vai considerar o analista como o perito em eficiência cuja tarefa é orientar a direção em como o emprego dele pode ser eliminado pela informatização. A solução para esse problema, caso ele ocorra, é fazer com que seja levado ao conhecimento dos níveis superiores dos usuários e obter o pronunciamento oficial, se possível, pessoalmente ou por escrito.

- **Você não conhece nossa empresa, como você quer dizer-nos como deve ser o novo sistema?** A resposta a essa pergunta é: Você tem razão! E por isso que o estou entrevistando para saber o que você pensa sobre quais devam ser os requisitos!. Por outro lado, deverá sugerir várias maneiras de melhorar as coisas (especialmente se parte ou todo o serviço realizado atualmente pelo usuário for em um antigo e ineficiente sistema); assim, esse tipo de comentário pode ser inevitável. Entretanto, o verdadeiro truque é continuar sendo tão respeitoso quanto possível e reconhecer constantemente a experiência do usuário na sua área, embora continuando a perguntar se ele poderia explicar-lhe porque sua idéia não funcionaria.
- **Você está tentando mudar o modo como as coisas são feitas aqui.** O artifício neste caso é mostrar ao usuário que embora possa estar propondo algumas (radicais) mudanças na implementação do sistema atual, não é pensamento modificar as características essenciais desse sistema, exceto nas áreas onde eles mesmos tenham solicitado uma alteração. Devemos lembrar, que algumas das características da implementação do sistema atual podem ser preservadas, por causa das interfaces do sistema atual com outros sistemas externos que exijam que as entradas ou saídas apresentem formatos prescritos.
- **Não queremos esse sistema.** Esta é uma variação da queixa você está querendo tirar meu emprego. A verdadeira resposta é que o analista está ali, conduzindo a entrevista, porque a direção quer o novo sistema. Não é da sua competência convencer os usuários operativos que eles devem querer o novo sistema; fazer isso é colocar o peso da responsabilidade sobre seus ombros, onde ela não deve ficar.
- **Por que você está desperdiçando nosso tempo com esta entrevista?** Sabemos o que queremos, e se você fosse competente, você saberia imediatamente o que queremos. Por que você não vai em frente e constrói o sistema? Esta é uma reclamação difícil de se lidar, porque relaciona-se com o fato fundamental de que os usuários e os analistas de sistemas estão falando línguas diferentes e estranhas. Lembre-se, que com a disponibilidade das linguagens de quarta geração e dos computadores pessoais, o usuário pode achar que pode construir ele próprio o sistema; sucessos fáceis com projetos simples (planilhas eletrônicas, por exemplo) podem ter-lhe dado a impressão de que todos os sistemas são fáceis de implementar. Isso pode explicar a impaciência em relação ao analista.

2.6 Outros Problemas

As diretrizes acima alertaram sobre os inúmeros problemas políticos com que o analista pode se defrontar em uma entrevista e os muitos motivos pelos quais o usuário pode mostrar-se hostil. Mas ainda existem alguns problemas para os quais deve-se estar atento:

- **Uma discussão que focalize mais os problemas de implementação do que os problemas dos requisitos.** Isso muitas vezes ocorre quando o usuário diz: Eis como eu gostaria que você construísse o sistema... . Isso acontece quase sempre quando o usuário está raciocinando em termos da implementação do sistema atual; e pode acontecer se o usuário conhecer alguma coisa da tecnologia de computadores (ex.: quando ele possui um PC particular ou quando ele é um ex-programador). Lembrar que não é obrigação em uma entrevista de análise descrever características de implementação do sistema a não ser que sejam tão importantes que realmente pertençam ao modelo de implementação do usuário.
- **Confusão entre sintomas e problemas.** Isso ocorre em muitas áreas, não apenas na área do processamento de dados. O que pode acontecer nas entrevistas de análise de sistemas. Entretanto, boa parte dele depende de onde a fronteira foi estabelecida no diagrama de contexto: se a queixa do usuário é um sintoma ou um problema depende

de ela estar associada a alguma coisa dentro dos limites do sistema ou fora deles. Desse modo, deve-se dar especial atenção ao desenvolvimento do modelo ambiental.

- **O usuário pode ser incapaz de explicar o que ele quer que o sistema faça ou pode mudar de opinião.** Esse é um problema comum e o analista de sistemas deve estar preparado para ele. Quanto maior for esse problema mais importante torna-se a prototipação.
- **Desentendimento entre usuários de mesmo nível, subordinados e chefes.** Infelizmente, isso coloca o analista de sistemas no papel de mediador entre as partes em desentendimento. Como analista, não pode abdicar desse papel; não pode dizer: Quando vocês decidirem o que querem e entrarem em um acordo, procurem-me. Em vez disso, deve-se agir como um negociador conduzindo todos os interessados a uma sala e trabalhando com eles na tentativa de chegar a um consenso. Isso, infelizmente, envolve habilidades e procedimentos fora do escopo deste texto.
- **Descobrir disparidades entre o padrão oficial e a pratica do trabalho.** Durante as entrevistas, o analista descobre que podem existir algumas disparidades entre a versão oficial de como o sistema funciona e como realmente ocorre no nível operacional. Neste caso, o analista deve utilizar a diplomacia quando relatar estas discrepâncias para a gerência, pois o pessoal de nível operacional pode relutar em admitir que as operações nem sempre seguem os padrões oficiais de trabalho.

2.7 Formas Alternativas de Coleta de Dados

As entrevistas não são o único modo de coleta de informações sobre os requisitos de um sistema. Na realidade, quanto mais informações puder colher de outras fontes, mais produtivas poderão ser as entrevistas pessoais. Alternativas para as entrevistas:

- **Questionários.** Pode remeter questionários escritos para os usuários dentro da organização, para as pessoas (ou setores) que interagem com o sistema, para os diretores que aprovaram o projeto e para outros.
- **Demonstrações feitas pelos fornecedores.** Os fornecedores de hardware e os fornecedores de software podem já haver desenvolvido sistemas prontos para a aplicação em que você esteja interessado. Solicitando-lhes uma demonstração dos recursos desses sistemas pode não somente auxiliá-lo a decidir se o produto é uma boa solução, mas também revelar funções e dados armazenados que você pode não ter percebido.
- **Visitas a outras instalações.** Procure outras empresas que estejam no mesmo ramo de atividades ou que tenham sistemas semelhantes aquele em que você esteja trabalhando. Combine uma visita à instalação para obter informações diretas sobre as características e aptidões do sistema.
- **Coleta de dados.** Procure formulários, relatórios, manuais, procedimentos escritos, registros, imagens de tela de terminais e listagens de programas que já existam na organização usuária. Lembre-se, todavia, que esses recursos normalmente estão relacionados com a implementação atual do sistema; devemos lembrar que isto costuma incluir informações redundantes e/ou contraditórias e/ou obsoletas. Não obstante, isto muitas vezes é um bom ponto de partida para você familiarizar-se com o terreno antes de iniciar as entrevistas pessoais com o usuário.
- **Pesquisa externa.** Se você estiver construindo um sistema para uma nova aplicação, para a qual o usuário não dispõe de qualquer experiência para descrever os requisitos, talvez seja necessário tentar obter informações em sociedades profissionais, ou em periódicos e livros técnicos e em relatórios de pesquisas.

2.7.1 Questionário de Pesquisa

Essa ferramenta é muito conveniente quando há um grande número de usuários ou vários grupos de usuários em locais diferentes. Nestas situações não é prático entrevistar todas as pessoas, mas tão logo as informações obtidas através dos questionários tenham sido analisadas, podem ser realizadas entrevistas com usuários selecionados.

Podem ser usados diversos formatos para o questionário de pesquisa, tais como: de múltipla escolha, lista de verificação (checklist), questões abertas (descritivas) e combinações das anteriores. Os passos para a utilização do questionário de pesquisa são:

a) Preparação do questionário:

- Identifique o tipo de informação que deseja obter, como problemas experimentados ou oportunidades a explorar;
- Definidos os requisitos, escolha um formato apropriado para o questionário;
- Monte questões de forma simples, clara e concisa;
- Se incluir questões abertas, observar o espaço necessário para a resposta;
- Agrupar as questões por tópicos ou áreas afins;
- Se possível, enviar junto com o questionário uma carta da direção da empresa explicando os motivos e a importância da pesquisa para a organização.

b) Identificação dos respondentes

- O questionário deve ser personalizado com o nome, função e localização do respondente;
- Elaborar um controle de identificação das pessoas que receberão os questionários. Utilizar o controle para monitorar a situação dos questionários.

c) Distribuição do questionário

- Distribua o questionário junto com as instruções de preenchimento;
- Indique claramente o prazo para a devolução do questionário e a forma de devolução.

d) Análise das respostas

- Consolide e análise as informações fornecidas pelos questionários devolvidos;
- Documente as principais descobertas;
- Envie uma cópia do relatório com as principais descobertas para cada respondente como uma cortesia pelo tempo dedicado à pesquisa.

2.7.2 Observações no ambiente

A análise de observação é uma técnica de observação de fatos muito efetiva. Ela pode ser usada para diversas finalidades, como processamento e confirmação de resultados de uma entrevista, identificação de documentos que devem ser coletados para análise, esclarecimento do que está sendo feito no ambiente atual.

Esta técnica é simples. Durante algum tempo, o analista fica observando os usuários em seu ambiente enquanto eles executam suas tarefas diárias. Freqüentemente o analista fará perguntas para conhecer o funcionamento e as atividades desenvolvidas. É importante explicar para as pessoas que serão observadas o que será observado e porque. Os passos para a observação são:

a) Antes:

- Identifique as áreas a serem observadas;
- Obter a aprovação da gerencia apropriada para executar a observação;
- Obter os nomes e funções das pessoas-chaves que serão envolvidas no estudo de observação;
- Explicar a finalidade do estudo;

b) Durante:

- Familiarizar-se com o local de trabalho a ser observado;
- Observar os agrupamentos organizacionais atuais;
- Observar as facilidades manuais e automatizadas em uso;
- Coletar amostras de documentos e procedimentos escritos que são utilizados nos processos observados;
- Acumular informações estatísticas das tarefas observadas: freqüência que ocorrem, estimativas de volumes, tempo de duração, etc...
- Enquanto interage com os usuários, tente ser objetivo e não comentar as formas de trabalho de maneira não construtiva;
- Observar também as exceções;
- Terminando as observações, agradeça às pessoas o apoio e a colaboração dispensada ao seu trabalho.

c) Após:

- Documente as descobertas resultantes das observações;
- Consolide os resultados;
- Reveja os resultados consolidados com as pessoas observadas e/ou com seus superiores.

3 OS PARADIGMAS DA ENGENHARIA DE SOFTWARE

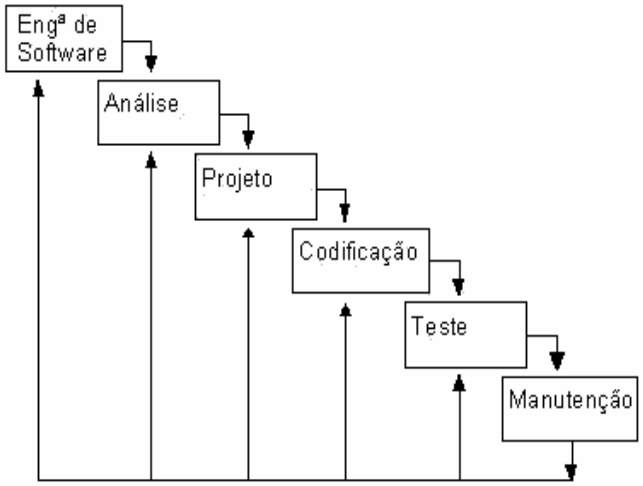
Um paradigma de engenharia de software é escolhido tendo como base: a natureza do projeto e da aplicação, os métodos e as ferramentas a serem usadas, os controles e os produtos que precisam ser entregues ao usuário.

3.1 O Ciclo de Vida Clássico

São utilizados conceitos de Engenharia de Software, a qual prevê atividade de verificação (estamos fazendo o produto de forma correta?), validação (estamos fazendo o produto certo?) e de controle de qualidade. É um paradigma que utiliza um método sistemático e seqüêncial em que o resultado de uma fase se constitui na entrada de outra.

As vezes chamado de modelo cascata, este paradigma abrange atividades (ou ciclos) de:

- **Engenharia de Software:** quanto mais dados forem coletados em nível de sistema, menor será a probabilidade de haver "bugs" no sistema, consequentemente diminuirá os futuros reparos no mesmo. Também conhecido como estudo de viabilidade.
- **Análise e Especificação de Requisitos de software:** é importante saber o que o cliente quer que o software tenha, com relação à recursos. Os requisitos devem ser documentados e revistos com o cliente antes de começar a execução do projeto.



- **Projeto:** envolve muitos passos que se concentram em 4 atributos distintos do programa: estrutura de dados, arquitetura de software, detalhes procedimentais e caracterização de interface. O processo de feitura do projeto traduz quanto à qualidade antes de iniciar a codificação.
- **Codificação:** o projeto deve ser transformado em programa, usando uma linguagem de programação, isto é, traduzido numa linguagem de máquina.
- **Testes e Integração do sistema:** deve-se testar todas os programas a procura de erros. A Integração consiste das junções das várias unidades e programas desenvolvidos. O resultado real deve concordar com o projeto ou resultado exigido. Depois de testado, o software é entregue ao usuário.
- **Manutenção e Operação:** o sistema é instalado e colocado em uso, indubitavelmente o software sofrerá mudanças depois que foi entregue ao cliente, e, mesmo software embutido sofre upgrade de tempos em tempos. A manutenção ocorre basicamente com a correção de erros não detectados (manutenção corretiva), com a adaptação da aplicação às mudanças do ambiente (manutenção adaptativa) e na adição de novas características e qualidades do software (manutenção evolutiva).

O ciclo de vida clássico é o paradigma mais antigo e o mais amplamente usado em Engenharia de Software, mesmo assim surgirão, ou surgem problemas como:

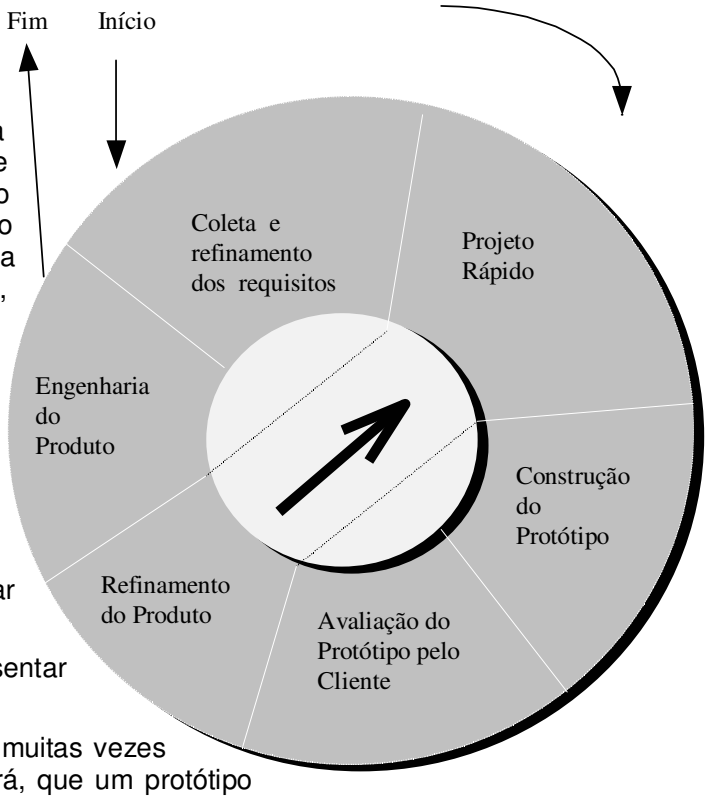
- Os projetos reais raramente seguem o fluxo sequencial que o modelo propõe.
- O cliente, amiúde, não saberá declarar todas as suas exigências.
- O cliente deve ter paciência, pois qualquer erro detectado após a revisão do programa de trabalho pode ser desastroso.

Os problemas são reais, mas, mesmo assim o ciclo de vida clássico continua sendo o mais usado no desenvolvimento de software. Este modelo é apropriado para sistemas transacionais onde as rotinas e procedimentos a serem automatizados são altamente estruturados. A principal desvantagem desta abordagem é o alto custo de correção das especificações quando nas fases de Teste e Implantação..

3.2 Prototipação

Muitas vezes, as dúvidas do cliente e do programador, em relação ao software, levam ao processo de prototipação. A prototipação capacita o desenvolvedor a criar um modelo de software que será implementado. O objetivo é antecipar ao usuário final um modelo de sistema para que ele possa avaliar sua finalidade, identificar erros e omissões, quando em utilização, efetuando de imediato correções e ajustes. O modelo pode assumir uma das três formas citadas:

- Um protótipo em papel ou em PC, que retrata a interação homem máquina, podendo ver quantas interação haverá.
- Um protótipo que implemente funções específicas, da função exigida pelo software.
- Um programa existente que executa parte ou toda a função desejada, mas que tem características que deverão ser melhoradas.



Como todas as abordagens ao desenvolvimento de software, a prototipação inicia-se com a coleta de requisitos. Faz-se então um projeto rápido contendo os aspectos que serão visíveis ao cliente. O projeto rápido leva à construção de um protótipo, que, será avaliado pelo cliente/usuário. Esta avaliação será usada para refinar requisitos para o software desenvolvido.

Idealmente, o protótipo serve como um mecanismo para identificar os requisitos de software. Muitas vezes, é preciso descartar um protótipo e, partir do início para evitar perda de tempo com correções.

Esse paradigma pode apresentar problemas pelas seguintes razões:

- O cliente quer resultados, e, muitas vezes não saberá, ou não entenderá, que um protótipo pode estar longe do software ideal, que ele nem sequer imagina como é. Mesmo

assim, a gerência de desenvolvimento cede às reclamações e tenta encurtar o prazo de entrega, o qual já estava prolongado.

- O desenvolvedor, na pressa de colocar um protótipo em funcionamento, é levado a usar um SO ou linguagem de programação imprópria por simplesmente estar a disposição ou estar mais familiarizado. Essa atitude poderá levar à um algoritmo ineficiente.

A filosofia de protótipos possui as seguintes vantagens:

- Maior garantia de sucesso técnico e psicológico;
- Redução no fator tempo: "O usuário gosta de ver o sistema funcionando";
- Ideal para sistemas gerenciais e de apoio a decisão.

Como desvantagens tem-se:

- Exige elevada capacitação gerencial por parte da equipe do projeto;
- Aparentemente, mais dispendioso (a longo prazo esta desvantagem tende a desaparecer);
- Exige uma ferramenta apropriada de prototipação.

O protótipo é baseado no feedback dos usuários, devido a isso mudanças são feitas no protótipo. Dependendo do comentário dos usuários, estas mudanças podem ser menores, de modo que alterem formatos, ou podem ser maiores, de modo que são requeridas mudanças estruturais no protótipo.

Ainda que possam ocorrer problemas, o uso da prototipação é um paradigma eficiente na Engenharia de Software. O segredo é o entendimento entre o desenvolvedor e o cliente.

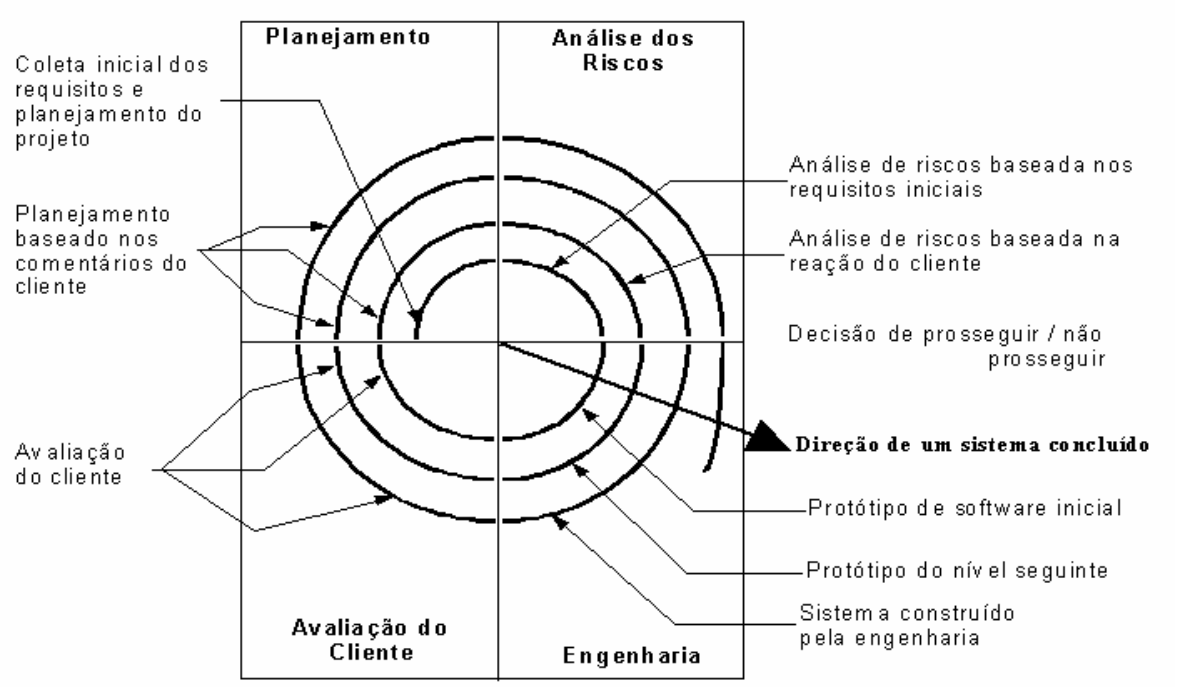
3.3 O Modelo Espiral

Foi desenvolvido para abranger as melhores características do ciclo de vida clássico e da prototipação, ao mesmo tempo que adiciona um novo elemento, que é a análise de risco. Este modelo de ciclo de vida se utiliza de protótipos por se adequar muito bem com esta filosofia de desenvolvimento. Cada passo através do ciclo inclui: planejamento, análise e projeto, prototipação e avaliação. Os passos vão sendo repetidos até que um produto seja obtido. Sendo assim definiu-se 4 importantes atividades:

- Planejamento, determinação de objetivos, alternativas e restrições do ciclo, considerando os resultados do ciclo anterior ou da análise dos requisitos.
- Análise dos riscos, análise das alternativas e identificação (resolução) dos riscos.
- Engenharia, Desenvolvimento e verificação da solução escolhida.
- Avaliação do Cliente, avaliação dos resultados e planejamento do ciclo seguinte.

Os riscos são circunstancias adversas que podem atrapalhar o processo de desenvolvimento e a qualidade do produto a ser desenvolvido, assim é possível prever e eliminar os problemas de alto risco através de um planejamento e projetos cuidadosos. Este é um modelo que atende os seguintes casos:

- o problema a ser resolvido não está totalmente entendido;
- a realidade pode mudar enquanto o sistema está sendo desenvolvido;
- a própria solução adotada pode ter algum efeito colateral desconhecido;
- a preocupação está centrada mais na qualidade e funcionalidade do que se produz.



Com base na experiência adquirida com a primeira versão, estabelecem-se novos requisitos para o sistema, e uma nova versão é concebida e implementada. A prototipação no ciclo de vida espiral tem como objetivos:

- estabelecer um diálogo intensivo entre usuários e analistas/projetistas;
- encurtar ao máximo o ciclo "concepção-implementação-utilização-avaliação" do sistema;
- possibilitar a evolução do sistema através de vários ciclos ou refinamentos sucessivos;
- avaliar constantemente o sistema.

Baseado principalmente em decisões de prosseguir / não prosseguir, de acordo com a avaliação, seja do cliente ou do desenvolvedor, o modelo espiral tende à uma trajetória que ruma para o modelo mais completo do sistema.

O paradigma de modelo espiral, é atualmente a abordagem mais realista para o desenvolvimento de softwares e sistemas em grande escala. Ele usa uma abordagem "evolucionária", capacitando o desenvolvedor e o cliente a entender e reagir aos riscos, em cada etapa evolutiva da espiral. Usa a prototipação como um mecanismo de redução de riscos, e a mesma pode ser aplicada em qualquer ponto evolutivo. Porém, pode ser difícil convencer grandes clientes de que a abordagem evolutiva é controlável. Se um grande risco não for descoberto, com certeza ocorrerão problemas.

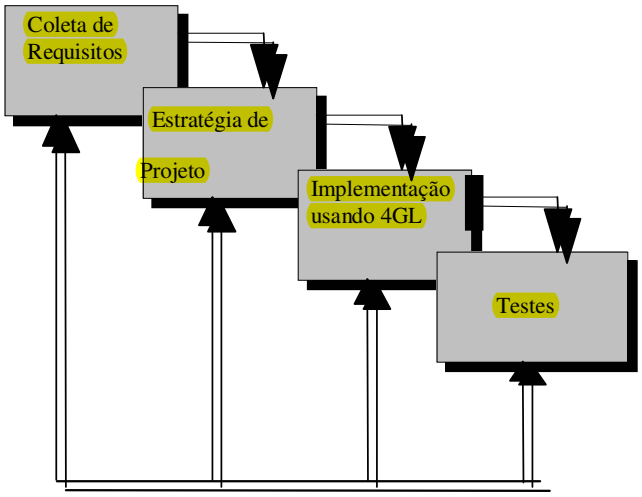
Esse paradigma é relativamente novo, e não tem sido amplamente usado como os 2 paradigmas anteriormente explicados. Somente o tempo, que gera a experiência, poderá comprovar a eficácia do modelo espiral.

3.4 Técnicas de 4a Geração (4GT)

Abrange um amplo conjunto de ferramentas para o desenvolvimento de software que tem uma coisa em comum: cada uma delas possibilita que o desenvolvedor especifique alguma característica do software num nível elevado. O código fonte é gerado automaticamente, tendo por base a especificação do desenvolvedor. Praticamente pode-se dizer à máquina, em

linguagem natural, as especificações que se quer para o software.

O paradigma 4GT possibilita resultados em um pequeno período de tempo. A parte de codificação, geração de telas, relatórios, consultas, em fim a programação propriamente dita, se torna automatizada. A parte considerada difícil, ou mesmo rotineira, seria a coleta de dados com o cliente. Saber o que o cliente quer, ainda é o principal problema de todos os paradigmas. Com a utilização de 4GL, talvez seja possível passar da coleta diretamente para a implementação. Porém, mesmo usando uma 4GL, é preciso fazer um planejamento do sistema, para evitar problemas de má qualidade, manutebilidade ruim e má aceitação do cliente.



Ferramentas como o Access, da Microsoft, podem fazer "milagres" pelo programador. Porém, para transformar uma 4GT em um produto, o desenvolvedor deve desenvolver testes cuidadosos e uma documentação significativa, além de executar todas as demais atividades de "transição", que os paradigmas anteriores executam. O sistema deverá possibilitar rápida manutenção.

Os opositores do paradigma 4GT, afirmam que tais ferramentas são insuficientes e que a manutebilidade de grandes sistemas de software desenvolvidos pelas 4GT estão abertas a questionamentos.

A prós e contras na discussão sobre este paradigma:

- Com raras exceções as 4GT's se limitam a aplicações de sistemas de informação comercial. Mas, utilizando ferramentas CASE, que agora suportam o uso das 4GT's, para a geração automática de esqueleto de código para as aplicações de engenharia em tempo real.
- Os dados preliminares, coletados em empresas que estão usando 4GT parecem indicar redução na quantidade de planejamento e análise para aplicações pequenas e intermediárias.
- Entretanto, o uso das 4GTs ainda exige tanto ou mais análise, planejamento e teste.

A projeção é que a demanda de software continue em ascensão, mas, que os paradigmas convencionais perderão espaço no mercado de desenvolvimento, para as 4GTs.

3.5 Modelo por incremento

Nos modelos por incremento, também chamado de modelo evolutivo, um único conjunto de componentes é desenvolvido simultaneamente, num primeiro tempo o núcleo do software e depois os incrementos vão sendo agregados.

As atividades de desenvolvimento e validação são desempenhadas paralelamente, com um rápido feedback entre elas. O processo de desenvolvimento de cada incremento é realizado usando um processo clássico dos já estudados anteriormente.

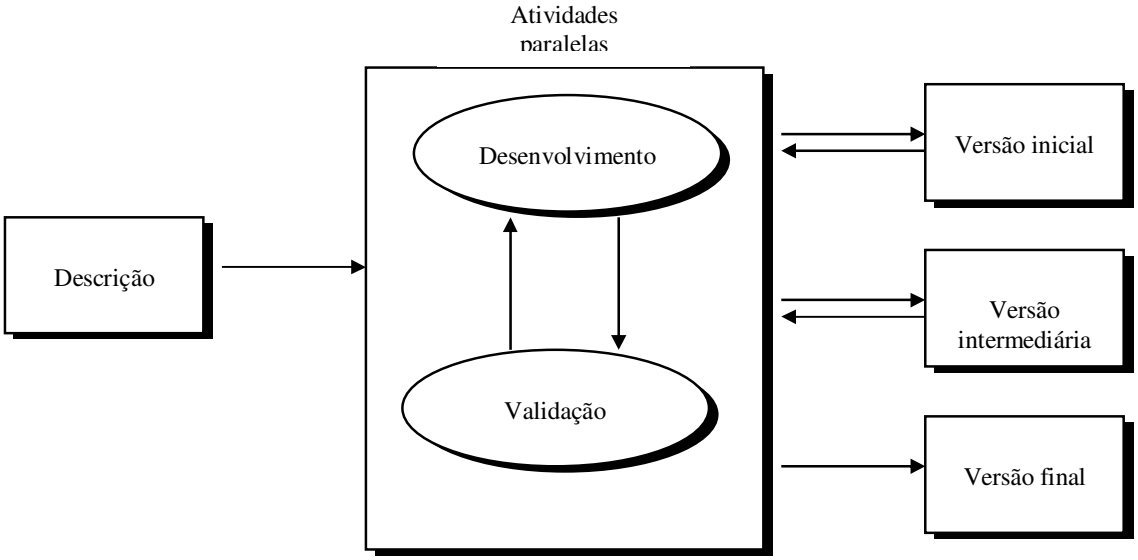
As vantagens são:

- Cada desenvolvimento é menos complexo.

- As integrações são progressivas.
- A cada interação, uma nova versão do produto pode ser distribuída.

Os riscos são:

- Os problemas no núcleo do software, inviabilizando novos incrementos, e determinando o fim do ciclo de vida do software.
- Incapacidade de integrar novos incrementos

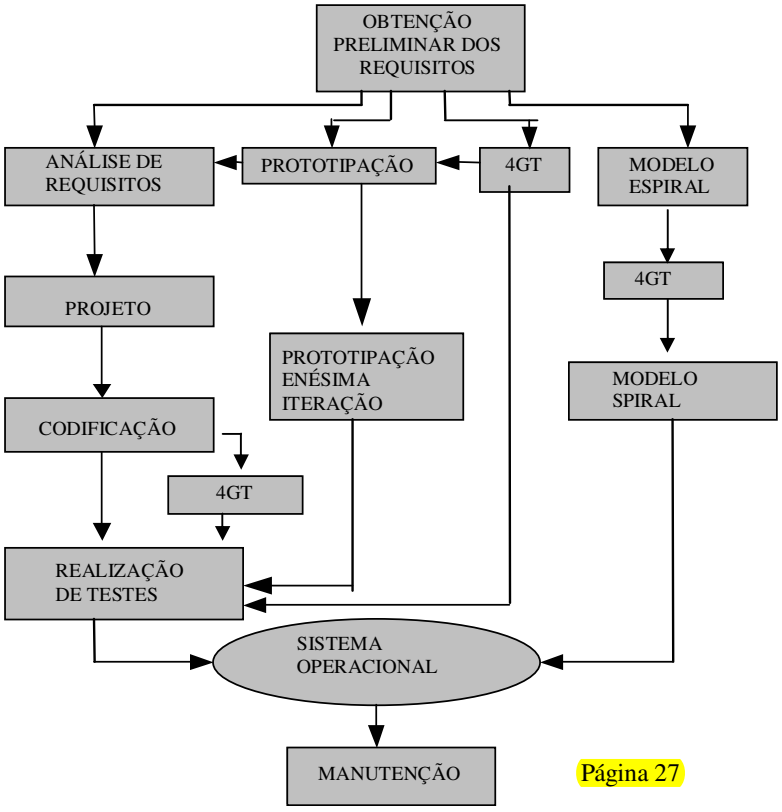


São usados em grandes projetos e também em softwares desenvolvidos para distribuição em grande escala, por permitir evolução e melhorias sem descaracterizar o software.

3.6 Combinando Paradigmas

Os paradigmas podem e devem ser combinados de forma que as potencialidades de cada um, possam ser utilizados em um único projeto. Qualquer paradigma pode constituir a base a qual os outros poderão ser integrados.

O processo começa com a determinação dos objetivos, alternativas e restrições e depois disso, qualquer caminho pode ser tomado. A natureza da aplicação é que vai determinar o paradigma a ser utilizado, e a combinação de paradigmas só tende a beneficiar o processo como um todo.



4 OS PROCESSOS DE SOFTWARE

Um processo de software é um conjunto de atividades e resultados associados que levam à produção de um produto de software. Esse processo pode envolver o desenvolvimento de software desde o início, embora, cada vez mais, ocorra o caso de um software novo ser desenvolvido mediante a expansão e a modificação de sistemas já existentes.

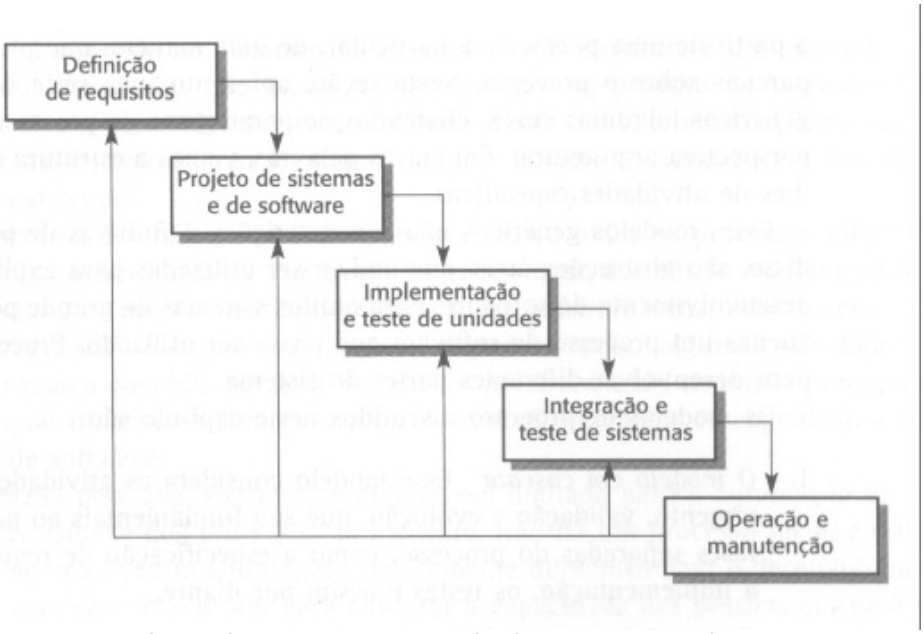
4.1 Modelos de processos de software

Um modelo de processo representa um processo a partir de uma perspectiva particular, de uma maneira que proporciona apenas informações parciais sobre o processo. Esses modelos genéricos não são descrições definitivas de processos de software, mas são abstrações úteis, que podem ser utilizadas para explicar diferentes abordagens do desenvolvimento do software.

4.2 Modelo em Cascata

É o primeiro modelo publicado do processo de desenvolvimento de software, já estudado no capitulo 2.

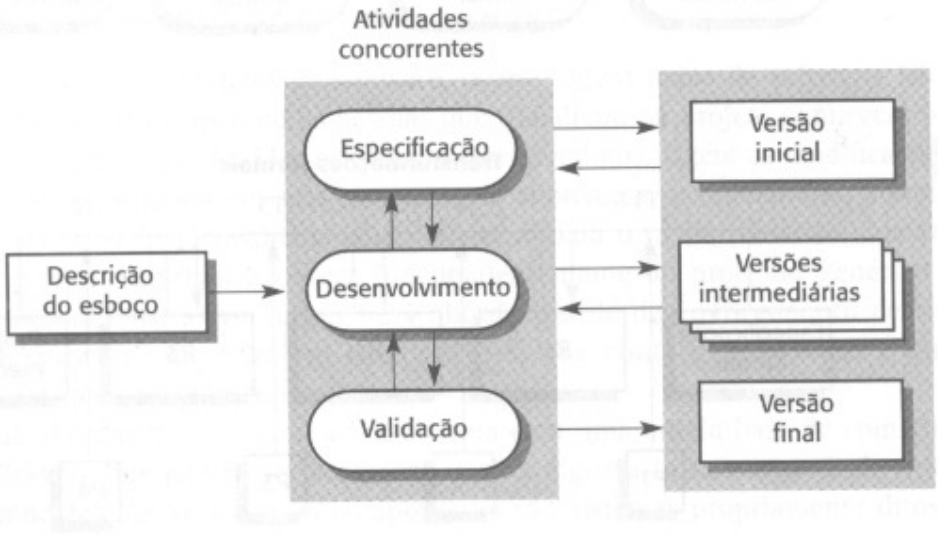
O resultado de cada fase envolve um ou mais documentos que são aprovados e assinados. A fase seguinte só é iniciada após a conclusão da fase precedente, mas na prática eles se sobrepõem e trocam informações. Durante o projeto, são identificados problemas com os requisitos; durante a codificação são verificados problemas do projeto, e assim por diante. O processo não é um modelo linear simples, mas envolve uma sequencia de iterações das atividades de desenvolvimento.



Devido aos custos de produção e aprovação de documentos, as iterações são onerosas e envolvem muito retrabalho. Depois de algumas iterações é normal suspender partes do desenvolvimento da fase e passar para o próximo estágio, postergando soluções ainda não encontradas. Essa suspensão prematura pode resultar em problemas futuros, quando da finalização do software.

4.3 Desenvolvimento Evolucionário

Tem como idéia o desenvolvimento da versão inicial que é exposta aos comentários do usuário e a partir destes é desenvolvido uma versão intermediária que é exposta aos comentários do usuário e assim sucessivamente (gera várias versões) até que um sistema adequado tenha sido desenvolvido.



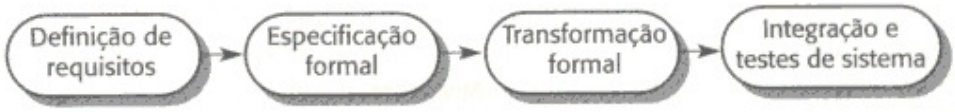
Há duas formas de desenvolvimento:

- Desenvolvimento exploratório – tem como objetivo trabalhar com o cliente e explorar os seus requisitos e entregar um sistema final;
- Prototipos descartáveis – tem como objetivo compreender os requisitos através dos prototipos e depois desenvolver uma definição dos requisitos para o sistema.

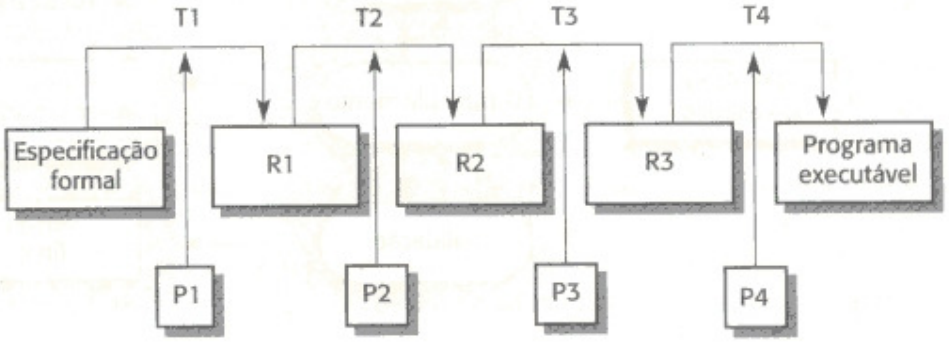
4.4 Desenvolvimento formal de sistemas

É uma abordagem que tem pontos em comum com o modelo em cascata, mas cujo processo de desenvolvimento tem como base a transformação matemática formal de uma especificação de sistemas em um programa executável

O desenvolvimento formal de sistema:



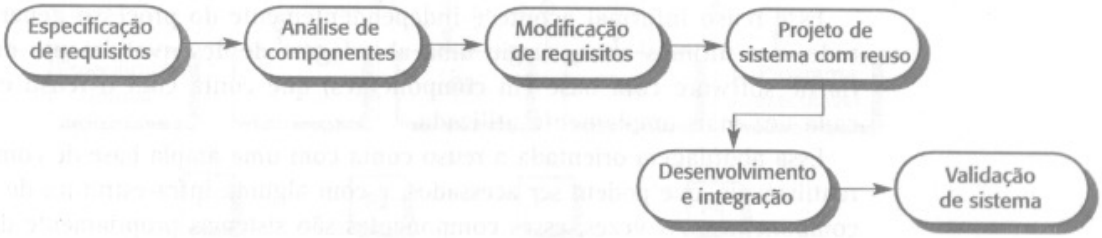
A transformação formal:



Cada etapa acrescenta mais detalhes, mas ainda é matematicamente correta, até que a especificação formal seja convertida em um programa equivalente.

4.5 Desenvolvimento Orientado a Reuso

Na maioria dos projetos de software já ocorre de modo informal algum reuso de software. A abordagem aqui descrita propõe a utilização de componentes previamente desenvolvidos e que podem ser utilizados por vários sistemas da forma que foram confeccionados. Existe uma ampla base de componentes de software reutilizáveis, que podem ser acessados, e com alguma infra-estrutura de integração para esses componentes.



Este modelo tem a vantagem de reduzir a quantidade de software a ser desenvolvido, portanto de reduzir custos e riscos, permitindo desta forma a entrega mais rápida do software. Contudo, as adequações nos requisitos podem ser inevitáveis e pode resultar em um sistema que não atenda as necessidades do usuário, além de que novas versões dos componentes reutilizáveis podem não estar sob controle da equipe de desenvolvimento.

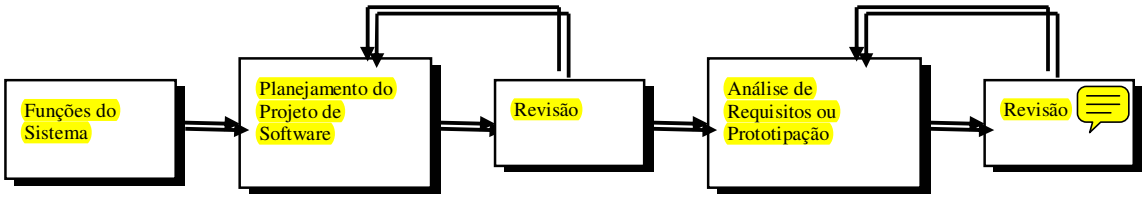
5 O DESENVOLVIMENTO DE SISTEMAS E AS SUAS ETAPAS

5.1 O Desenvolvimento na visão Pressman,

O desenvolvimento, segundo Presman é realizado em três fases:

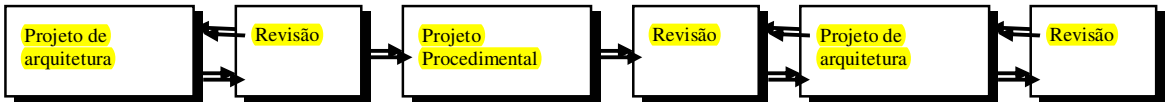
5.1.1 Fase de Definição

Planejamento do software: descrição do escopo, análise do esforço, análise de riscos, levantamento dos recursos exigidos, estimativas de custos e de prazos. O objetivo é fornecer uma indicação da viabilidade do software; fase de análise e requisitos do software: a análise forma do domínio da informação é utilizada para estabelecer modelos de fluxo de dados e da estrutura da informação. Alternativamente pode ser feito um protótipo. Estes modelos são detalhados para se tornar uma **especificação do software**, que é o **documento** produzido com resultado desta fase.



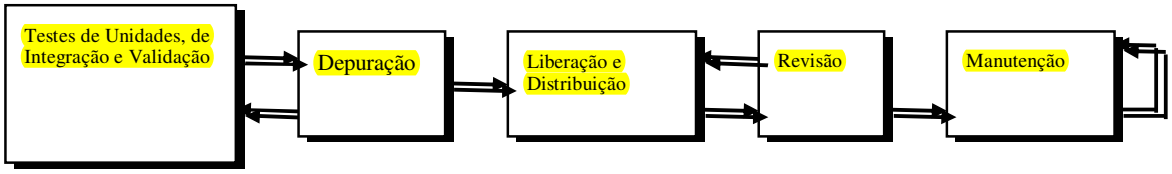
5.1.2 Fase de Desenvolvimento

Descrição de estrutura modular, definição de interfaces, uma estrutura de dados é estabelecida. Uma especificação de projeto é produzida. E a codificação é realizada.



5.1.3 Fase de Verificação, Liberação e Manutenção

Realização de testes para descobrir o máximo de erros. Faz-se a manutenção do software ao longo da sua vida útil.



5.1.4 Conceitos utilizados no desenvolvimento:

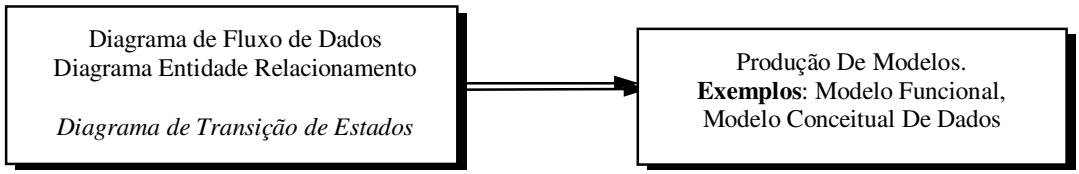
Metodologias de Desenvolvimento - maneira de se utilizar um conjunto coerente e coordenado de métodos para atingir um objetivo. Em outras palavras, a metodologia deve definir quais as fases de trabalho previstas no desenvolvimento de sistemas.

Método - é um procedimento a ser adotado para se atingir um objetivo.

Técnica - é um modo apropriado de se investigar sistematicamente um determinado universo de interesse ou domínio de um problema. Ex: análise estruturada, análise essencial e projeto estruturado.

Notação - é um conjunto de caracteres, símbolos e sinais formando um sistema convencionado de representação.

Ferramentas



A metodologia deve estabelecer quais os pontos de controle e padrões de qualidade

5.1.5 Técnicas utilizadas no desenvolvimento de sistemas

Técnicas	Abordagens	Ferramentas
Análise Tradicional	Funcional	Textos e Fluxogramas
Análise Estruturada	Funcional Dados	Diagrama de fluxo de dados Diagrama de estrutura de dados Normalização Dicionário de dados
Analise Essencial	Funcional Dados Controle	Tabela Diagrama de fluxo de dados Diagrama de entidade-relacionamento Diagrama de transição de estados
Analise Orientada à Objetos	Funcional Dados Componentes	Diagrama de classes de objetos Cenários e maquinas de estados Diagrama de fluxo de dados

6 TÉCNICA ESTRUTURADA

6.1 Introdução

Neste capitulo serão apresentadas as técnicas estruturadas utilizadas no desenvolvimento de sistemas, e que podem também ser chamadas de: técnicas de melhoria de produtividade, técnicas de produtividade na programação ou ainda técnicas de engenharia de software. Elas incluem os seguintes tópicos:

- Análise estruturada;
- Projeto estruturado;
- Programação estruturada;
- Desenvolvimento top-down;
- Equipes de programação;
- Revisões estruturadas.

6.2 Análise Estruturada

Como o próprio nome sugere, ela refere-se ao extremo inicial de um projeto de desenvolvimento de sistemas, durante o tempo em que os requisitos do usuário são definidos e documentados.

Por que a análise estruturada é tão importante? Simplesmente porque a análise clássica é não atendeu as necessidades. O problema é muito simples de ser expressado: em qualquer coisa diferente de um projeto de desenvolvimento de sistemas trivial, o usuário não tem um bom entendimento do que está sendo feito para ele. Os problemas da análise clássica podem ser resumidos como:

- As especificações funcionais clássicas são monolíticas;
- As especificações funcionais clássicas são redundantes;
- As especificações funcionais clássicas são difíceis de se modificar ou manter;
- As especificações funcionais clássicas fazem muitas suposições sobre os detalhes de implementação.

Todos esses fatores contribuem para tornar a fase de análise dos sistemas convencionais, na maioria dos grandes projetos, uma atividade dolorosa e demorada. Em muitos casos, todos estão desesperados para passar para pela aprovação. Tendo acabado a fase de análise de sistemas, poucos pensam em voltar atrás e reexaminar ou revisar as especificações formais.

O que então a análise estruturada pode oferecer? A análise estruturada introduz o uso de ferramentas de documentação gráfica para produzir um tipo diferente de especificação funcional, uma especificação estruturada, em contraste à novela vitoriana clássica e monolítica. As ferramentas de documentação da análise estruturada são:

- Lista de Eventos
- Diagrama de Contexto
- Diagrama de fluxo de dados (DFD);

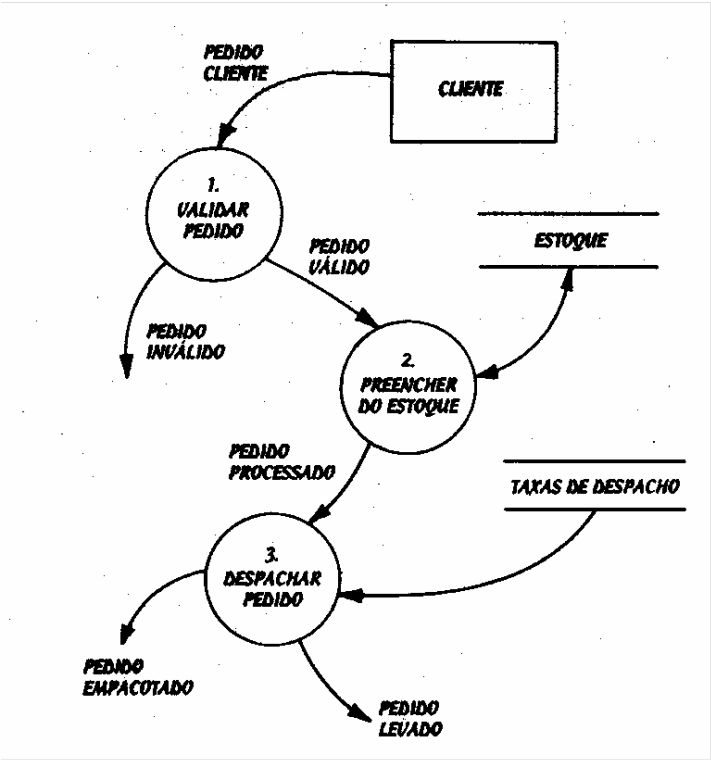
- Dicionário de dados (DD);
- Diagrama de entidade relacionamento;
- Diagramas de transição de estado (DTE);
- Especificação de processo.

6.2.1 Diagrama de Contexto

Documenta o âmbito do estudo, apresentando os fluxos de dados que entram e saem do sistema e sua interação com as entidades externas. Tem como função delinear a área de observação.

6.2.2 Diagrama de fluxo de dados

Fornece um meio fácil e gráfico de modelar o fluxo de dados pelo sistema. É uma representação em rede dos processos (funções) do sistema e dos dados que ligam esses processos. Ele mostra o que o sistema faz e não como é feito. A figura mostra os elementos básicos de um DFD: os terminadores ou Entidades (fontes ou destinos), Fluxos de dados, processos e Depósitos ou armazenagem de dados.



Entidade – é na maioria das vezes uma categoria de coisas ou pessoas que representam uma origem ou destino das transações.

Fluxo de dados – podemos associar cada fluxo de dados com um tubo por onde passam pacotes de dados. É identificado por uma descrição do seu conteúdo.

Processo – É a descrição da função a ser executada, o seu nome é sempre no imperativo e o mais objetivo possível.

Depósito de Dados – é o local onde serão armazenadas as informações (dados) processadas e de onde serão recuperadas quando necessárias.

6.2.3 Dicionário de dados

Fornece a informação de texto de suporte para complementar a informação gráfica mostrada no DFD, é simplesmente um grupo organizado de definições de todos os elementos de dados no sistema sendo modelado. Deverá conter a definição dos elementos que tornam o Modelo de Dados e o Diagrama de Fluxo de Dados precisos, quais sejam: Fluxos de dados, Depósitos de dados e Atributos dos depósitos.

Por exemplo, o elemento de dados **pedido-cliente** mostrado no DFD da figura acima poderia ser definido da seguinte forma:

PEDIDO-CLIENTE = NOME-CLIENTE + NRO-CONTA + ENDEREÇO + 1{ITEM-PEDIDO}6

ITEM-PEDIDO = COD-PRODUTO + DESC-PRODUTO + QTD-PROD + PRECO

ENDERECO = TIPO-LOGRADO + LOGRADOURO + NRO-LOGRADO + (COMPLEMENTO) + BAIRRO + CIDADE + UF + CEP + (TELEFONE)

TIPO-LOGRADO = [RUA | AVENIDA | PRACA | TRAVESSA]

Regras para a formação de nomes dos fluxos de dados:

- O nome deve ser formado por palavras separadas por sublinha com até 32 caracteres.
- Preferencialmente os nomes devem ser efetuados de acordo com o usuário.
- Devem ser eliminados as proposições e conjunções.
- Quando houver a necessidade de abreviar uma palavra, que seja uma abreviatura clara

Simbologia utilizada no Dicionário de Dados:

Símbolo	**Significado**
=	É composto de
+	E
[ ]	Escolha uma das opções alternativas
{ }	Interações de
( )	Opcional (pode estar presente ou ausente)
|	Separa as opções alternativas na construção [ ]
* *	Comentário
@	Identificador (campo chave) de um depósito de dados

6.2.4 Diagrama de Entidade-Relacionamento (DER)

Também conhecido como Modelo de Dados. Ele enfatiza os principais objetos, ou entidades com o que o sistema lida, bem como a relação entre os objetos. Os objetos normalmente correspondem, um a um, aos locais de armazenagem de dados mostrados no DFD, mas o DFD não informa sobre as relações entre os objetos. Mostra uma visão estática das informações (entidades) de interesse e dos vínculos (relacionamentos) existentes entre elas.

Os componentes do DER são: Entidade, Atributo, Chave de identificação, Lista de entidades, Domínio e Ocorrência.

Entidade é algo real ou abstrato, percebido no ambiente e sobre o qual nos interessa armazenar dados.

Atributo é um dos itens de dados que armazenamos sobre uma entidade, caracteriza ou qualifica uma determinada propriedade de uma entidade.

Chave de Identificação de uma entidade é definida por um atributo, ou conjunto de atributos, cujos valores individualizam uma única ocorrência dessa entidade. Tipos de chaves:

- chaves candidatas – são as possíveis chaves de identificação de uma única ocorrência na entidade.

- Chave primária – é uma das chaves candidatas, selecionada por melhor conveniência (facilidade de uso, menor possibilidade de erro, etc...)
- Chave estrangeira é um conjunto de um ou mais atributos de uma entidade que são chave primária em outra entidade.

Exemplo:

EMPREGADO(MATRICULA, NOME_EMPR, CPF_EMPR, COD_DEPTO)

DEPARTAMENTO(COD_DEPTO, NOME_DEPTO)

Lista de Entidades é uma relação de entidades com seus respectivos atributos, utilizada para documentar os trabalhos de análise de dados. É formada pelo nome da entidade seguida da relação de atributos que compõem entre parêntesis e seguindo a seguinte convenção:

- Cada atributo é separado do outro pelo sinal de adição (+).
- O(s) atributo(s) que identificam a entidade devem estar no início da relação e sublinhados.
- O(s) atributo(s) que ocorrem mais de uma vez (repetitivos) são identificados por uma inclusão entre parêntesis.

Exemplo:

CLIENTE(COD_CLIENTE + NOME_CLIENTE + END_CLIENTE + (FONE_CLIENTE) + CPF_CLIENTE)

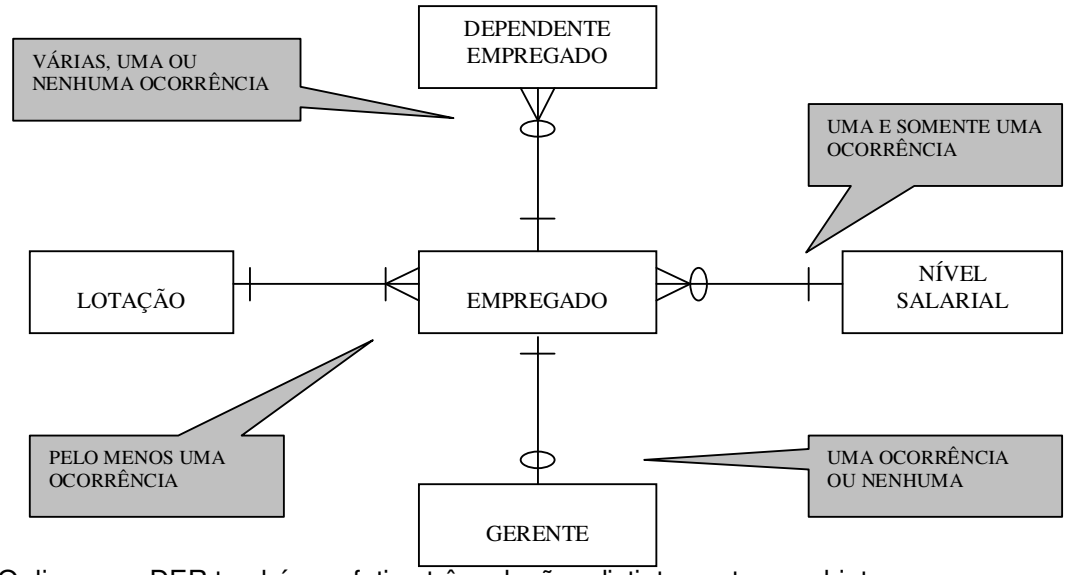
Domínio são os possíveis valores que um atributo pode assumir.

Exemplo:

SEXO = [M | F]

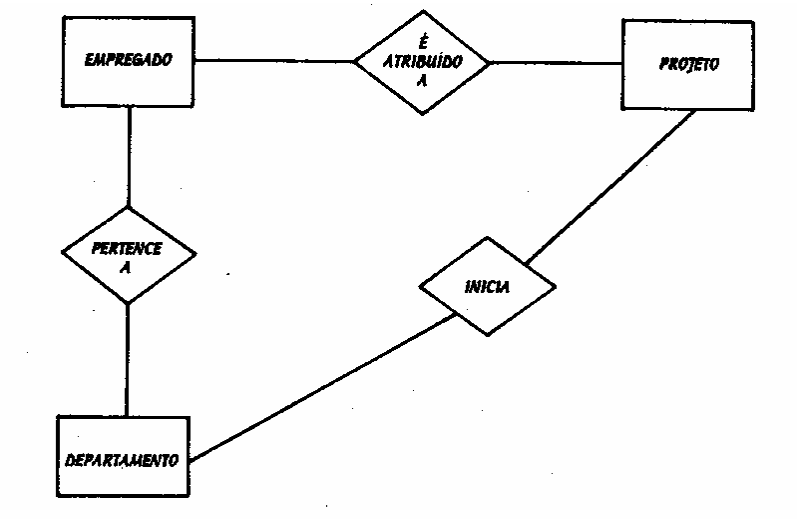
Ocorrência representa o número de vezes que um determinado atributo aparece em outra entidade.

Símbolos especiais colocados nas extremidades de uma linha que representa o relacionamento. Exemplo:



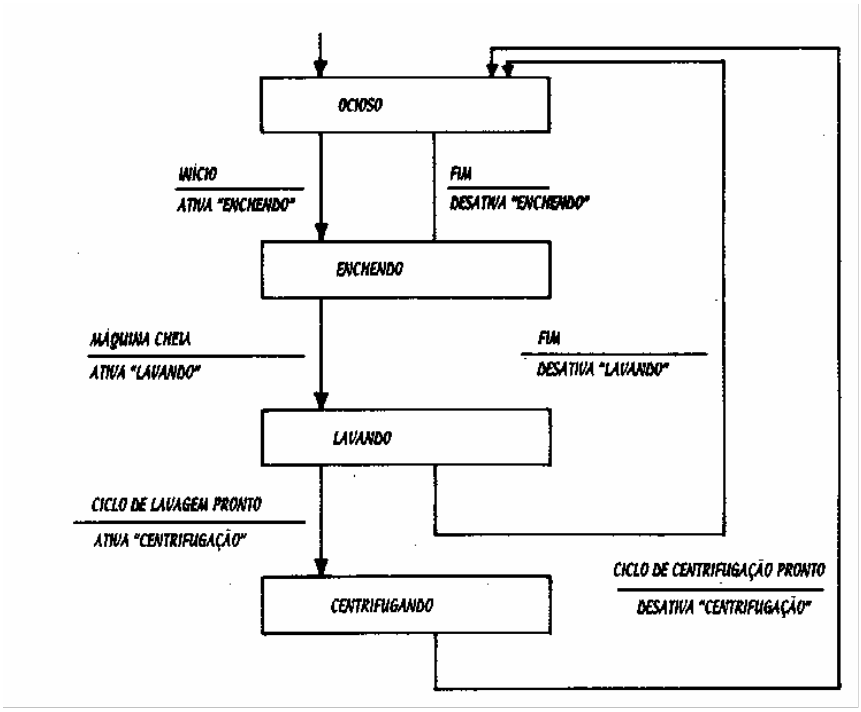
O diagrama DER também enfatiza três relações distintas entre os objetos:

- Empregados pertencem a departamentos;
- Empregados são atribuídos a projetos;
- Departamentos iniciam projetos.



6.2.5 Diagrama de Transição de Estado (DTE)

É usado para modelar o comportamento de transição de estado, é uma ferramenta de modelagem para descrever o comportamento de sistemas de tempo-real e a parte de interface humana de muitos sistemas on-line. Cada estado representa o período de tempo durante o qual o sistema tem algum comportamento observável; as setas conectando cada quadro retangular mostra a mudança de estado, ou transições de um estado para outro. Associadas a cada mudança de estado estão uma ou mais condições (os eventos ou circunstancias que causam a mudança de estado), e zero ou mais ações (a resposta, ou saída, ou atividade que ocupa parte da mudança de estado).



As etapas de construção de um DTE, podem ser seguidas dos seguintes modos:

- Começar pela identificação de todos os estados possíveis do sistema, representando

cada um deles como um retângulo. Em seguida, achar todas as conexões significativas (mudança de estado) entre os retângulos.

- Começar pelo estado inicial, continuando metodicamente seu caminho para o(s) estado(s) seguinte(s). depois do(s) estado(s) secundário(s) para o(s) terciário(s).
- Após ter elaborado o DTE, devemos verificar a consistência do mesmo, respondendo:
- Foram definidos todos os estados?
- Todos os estados foram atingidos? Algum estado foi definido sem que haja caminhos que levem a ele?
- Todos os estados tem saída?
- Em cada estado, o sistema reage adequadamente a todas as condições possíveis? Este é o erro mais comum, é esquecido de especificar o comportamento do sistema em condições inesperadas.

6.2.6 Especificação de Processos

Sua finalidade é permitir que o analista de sistemas descreva, rigorosa e precisamente, a política representada por cada um dos processos atômicos de baixo nível nos diagramas de fluxo de dados de baixo-nível.

Uma das maneiras de descrever a especificação de processos é utilizar o português estruturado, mas existem outras como Tabela de decisão e Árvore de decisão.

O português estruturado, utilizado na especificação de processos consiste apenas no seguinte:

- Um grupo limitado de verbos de ação, como calcular e validar – um grupo de mais ou menos 40 verbos;
- Estruturas de controle tiradas da programação estruturada para descrever ações alternativas e repetitivas, como: Se-então-caso contrario , Faça-enquanto e Repita-enquanto;
- Nomes que foram definidos no dicionário de dados ou que estão definidos dentro da própria especificação de processo.

Exemplo, especificação para um processo:

Processo 3.3 – Executa Cobrança

1. Se o valor da fatura vezes o número de semana em atraso for maior que R$ 1000,00 Faça:
   a) Chamar o Cliente;
   b) Examinar o Cliente em duas semanas;
   c) Incrementar Contador de notas em atraso em um;
2. Caso contrário Se menos de quatro notas de atraso foram enviadas, Faça:
   a) Chamar o Cliente;
   b) Examinar a fatura em uma semana;
   c) Incrementar Contador de notas em atraso em um;
3. Caso contrário :
   a) Enviar ao Cliente uma cópia da fatura contendo o número de semanas e atraso;
   b) Examinar a fatura em uma semana;

Para desenvolver, pode-se seguir:

- Escolha uma ou mais implementações lingüísticas para cada uma das construções básicas. Voce deveria ficar com a fraseologia usada nesta seção ou desenvolver uma própria.
- Use as palavras e frases-chave de suas implementações para compor uma lista de palavras reservadas.
- Acrescente à sua lista palavras novas que são necessárias à descrição de condições, relacionamentos, etc..., quando for detectada a necessidade delas.
- Ao escrever descrições de programas de ação, restrinja-se ao uso das palavras em sua lista reservada, termos do Dicionário de dados, e verbos no imperativo.

Utilizando estes recursos para a descrição do programa de ação, o processo será escrito em português estruturado.

6.3 Projeto Estruturado

Talvez o ponto mais importante a entender seja que o projeto estruturado não é apenas programação modular, ou projeto modular, como foi chamado inicialmente, e nem apenas um projeto top-down. O projeto estruturado pode ser definido como a determinação de quais módulos, interconectados de que forma, melhor solucionarão algum problema bem definido.

O projeto estruturado insiste numa definição de módulo simples, como sendo um grupo de comandos de programa contíguos e ligados, possuindo um único identificador (nome) pelo qual ele pode ser referenciado como uma unidade. Os componentes do projeto estruturado são:

- Técnicas de documentação;
- Critérios de avaliação de projeto;
- Heurísticas de projeto;
- Estratégias de projeto.

6.4 Programação Estruturada

A programação estruturada foi a primeira das técnicas estruturadas a ser discutida e praticada amplamente. Para muitas pessoas, o termo programação estruturada é genérico e inclui filosofias de projeto, estratégias de implementação, conceitos de organização de programas, e as virtudes básicas de prosperidade, lealdade e bravura. Para as finalidades desta discussão, no entanto, a programação estruturada deve ser pensada como uma técnica de codificação e só deve ser discutida no contexto da programação.

A teoria por trás da programação estruturada é relativamente simples: ela diz que qualquer lógica de programa pode ser construída pelas combinações das três estruturas: seqüência, decisão e repetição. Na maioria das linguagens de programação de alto nível isto significa que os programas de computador podem ser construídos com as seguintes combinações:

- Comandos imperativos simples, como: leia, calcule, escreva e comandos algébricos;
- Comandos para a tomada de decisão, como: se e caso;
- Comandos de repetição ou iteração, como: repita até que ou faça enquanto.

O ponto principal é que este conjunto de construções é suficiente para formar todo e qualquer tipo de lógica de programa; e em particular, não é necessário usar os comandos de desvio em programação estruturada.

6.5 Desenvolvimento Top-down

Em muitos casos, por motivos históricos, o desenvolvimento top-down é confundido com projeto top-down, mas estamos pensando em implementação top-down. Três aspectos top-down estão relacionados, mas distintos, da seguinte forma:

- **Projeto top-down** é uma estratégia de projeto que divide problemas grandes e complexos em problemas menores e mais simples, facilmente solucionáveis;
- **Codificação top-down** é uma estratégia para se codificar módulos de alto-nível, antes que os módulos detalhados de nível mais baixo;
- **Implementação top-down** é uma estratégia para testar os módulos de alto-nível de um sistema antes que os módulos de baixo nível tenham sido codificados e possivelmente, até antes que tenham sido projetados.

O desenvolvimento top-down consiste em desenvolver inicialmente uma versão inicial com todos os módulos de alto-nível do sistema. As versões subsequentes simplesmente envolvem o acréscimo de módulos de nível mais baixo ao esqueleto já existente dos módulos de alto-nível, até que uma versão final produz a saída satisfatória ao usuário.

6.6 Equipes de Programação

Outro aspecto do método de desenvolvimento de sistemas estruturados é o conceito de equipes de programação, em particular, as equipes de programador-chefe e equipes abertas.

O conceito de equipe de programador-chefe é baseada na liderança, na capacidade técnica e na experiência de uma pessoa para comandar o desenvolvimento de um sistema por uma equipe.

O conceito de equipe aberta é baseada na teoria da participação de toda a equipe na solução dos problemas sem a supervisão direta de qualquer um.

6.7 Revisões Estruturadas

Uma revisão estruturada é um procedimento organizado para que um grupo de examinadores (analistas de sistemas, programadores, etc) revisem um produto técnico para fins de correção e garantia de qualidade.

As revisões podem ocorrer em uma série de estágios num projeto típico de desenvolvimento de sistemas. A ênfase tem sido na revisão do código, mas o conceito se aplica também às revisões de projetos e especificações.

O método de revisões estruturadas é conduzida por membros de uma equipe, são quase sempre caracterizadas por um ambiente informal e aberto; permite revisões rápidas e eficientes da precisão técnica da especificação, projeto e código para o sistema.

6.8 As Ferramentas da Análise Estruturada

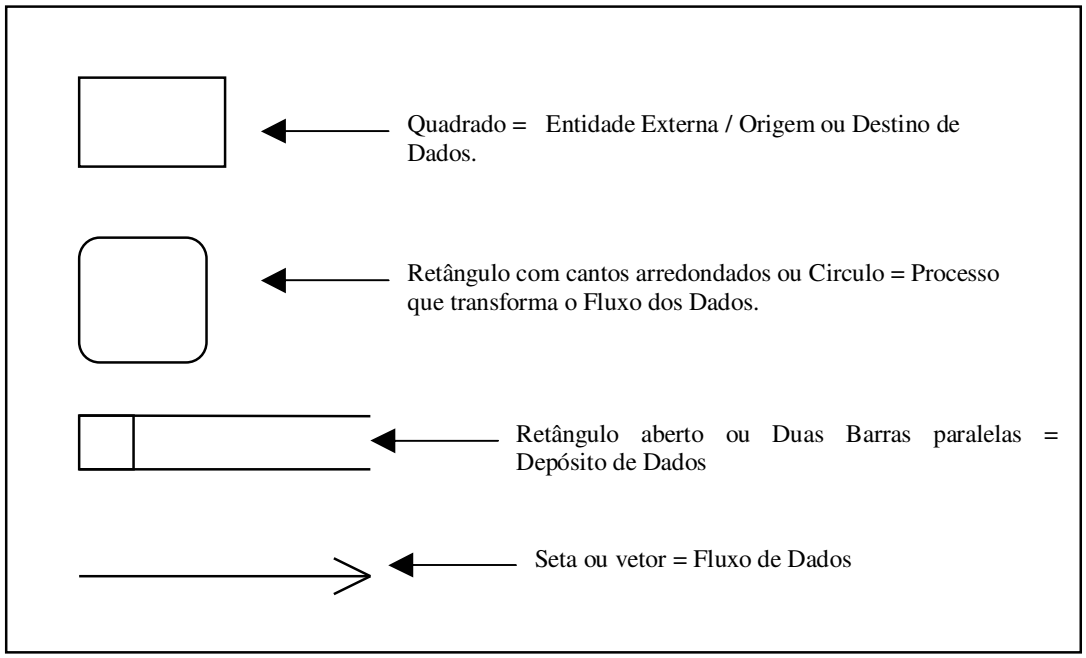
6.8.1 Diagrama de Fluxo de Dados

DFD é uma representação em rede dos processos (funções) do sistema e dos dados que ligam esses processos. Ele mostra o que o sistema faz e não como é feito. É a ferramenta de demonstração central da análise estruturada.

Um DFD apresenta as partes componentes de um sistema e as interfaces entre elas. É um conjunto integrado de procedimentos, sendo que as partes do computados poderão estar inseridos ou não.

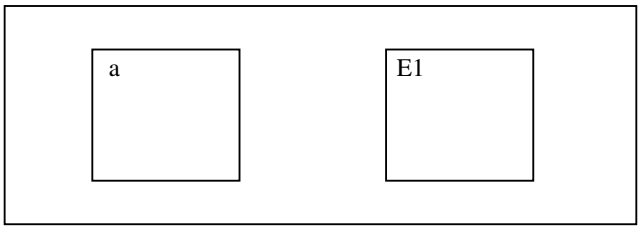
Na elaboração de um DFD, utilizaremos quatro símbolos que nos permitirão, debater e apresentar ao usuário todo o processo, sem assumir nenhum compromisso com implementações e demostrar a sua fluência, sem a preocupação com a hierarquização e tomadas de decisão.

São os seguintes símbolos utilizados na elaboração de um DFD:



a) Entidade

Identificar como entidade, na maioria das vezes, categorias lógicas de coisas ou pessoas que representam uma origem ou destino de transações (Clientes, Fornecedores, Empregados, Etc.). Também pode-se identificar como Entidades fontes ou destinos específicos tais como Departamentos da empresa, Receita Federal, Almoxarifado. É comum adotarmos a terminologia Entidade Externa. Quando um sistema recebe dados resultantes de outro, ou gera informações que servirão como dados de entrada para outro, esse outro sistema também é identificado como uma Entidade Externa.



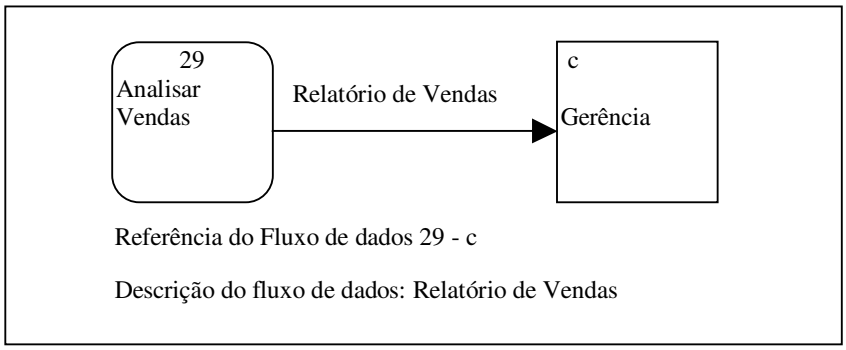
O símbolo utilizado para representar já foi apresentado anteriormente.

Por convenção, a fim de simplificar as referências e o processo de elaboração do Dicionário de dados, adicionamos como identificador de uma entidade uma letra minúscula no canto superior esquerdo do desenho ou a letra E maiúscula e um número, conforme a figura acima:

b) Fluxo de Dados

Pode-se associar cada fluxo de dados com um tubo por onde passam pacotes de dados. Faremos referência ao Fluxo de Dados identificando os processos, entidades ou depósitos de dados das suas extremidades, anotando uma descrição do seu conteúdo ao longo de sua extensão. Lembre-se que a descrição deve ser mais clara possível, de modo a simplificar o trabalho do usuário que irá realizar a revisão do DFD.

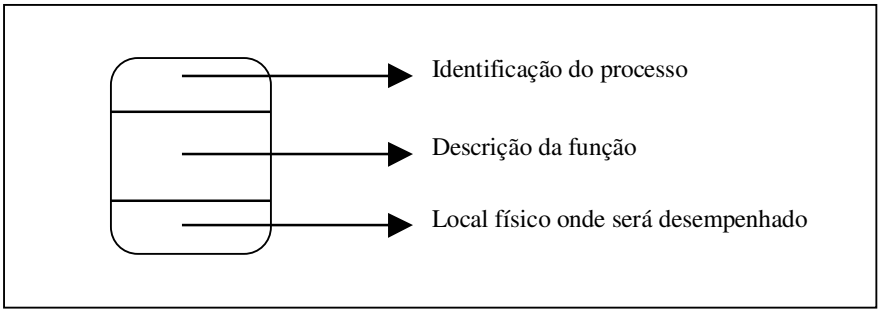
Observe um exemplo de referência e descrição de Fluxo de Dados:



c) Processo

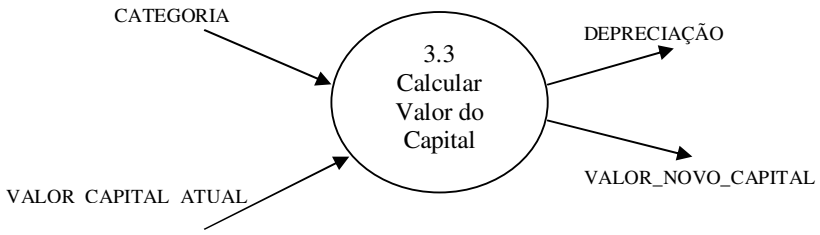
Logicamente, é necessário descrever a função de cada processo, e, para facilitar, atribuir uma identificação única para cada um, buscando, na medida do possível, associá-lo a um sistema físico.

A identificação pode ser um número, inicialmente posicionado na posição média superior da figura, não tendo nenhum outro significado além de identificar o processo. Não há porquê vincularmos a identificação com a descrição do processo, pois alguns deles serão subdivididos em dois ou mais nas fases de expansão - o que implicará no surgimento de novos números. Entretanto, a partir do instante que um processo recebe uma identificação, está não deve mais ser modificada, sob a pena de comprometer o trabalho de elaboração do dicionário de dados, exceto nos casos de desmembramentos e agrupamentos. Para simplificar o entendimento da figura, podemos adicionar linhas divisórias, marcando claramente o espaço destinado à identificação do processo, sua descrição e o local físico onde será desempenhado.



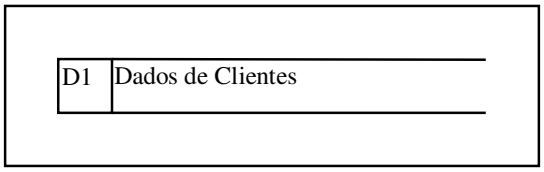
Vale ressaltar que a descrição da função deve ser sempre imperativa, composta por um verbo ativo (verificar, extrair, recuperar, comparar), seguida de uma cláusula, simples e objetiva.

A identificação do local físico onde a função será executa, opcional nos diagramas de nível mais abrangente, é extremamente útil a partir do instante em que a análise foi concluída e o projeto físico do sistema está sendo desenvolvido, pois denota o departamento ou programa que o desempenhará



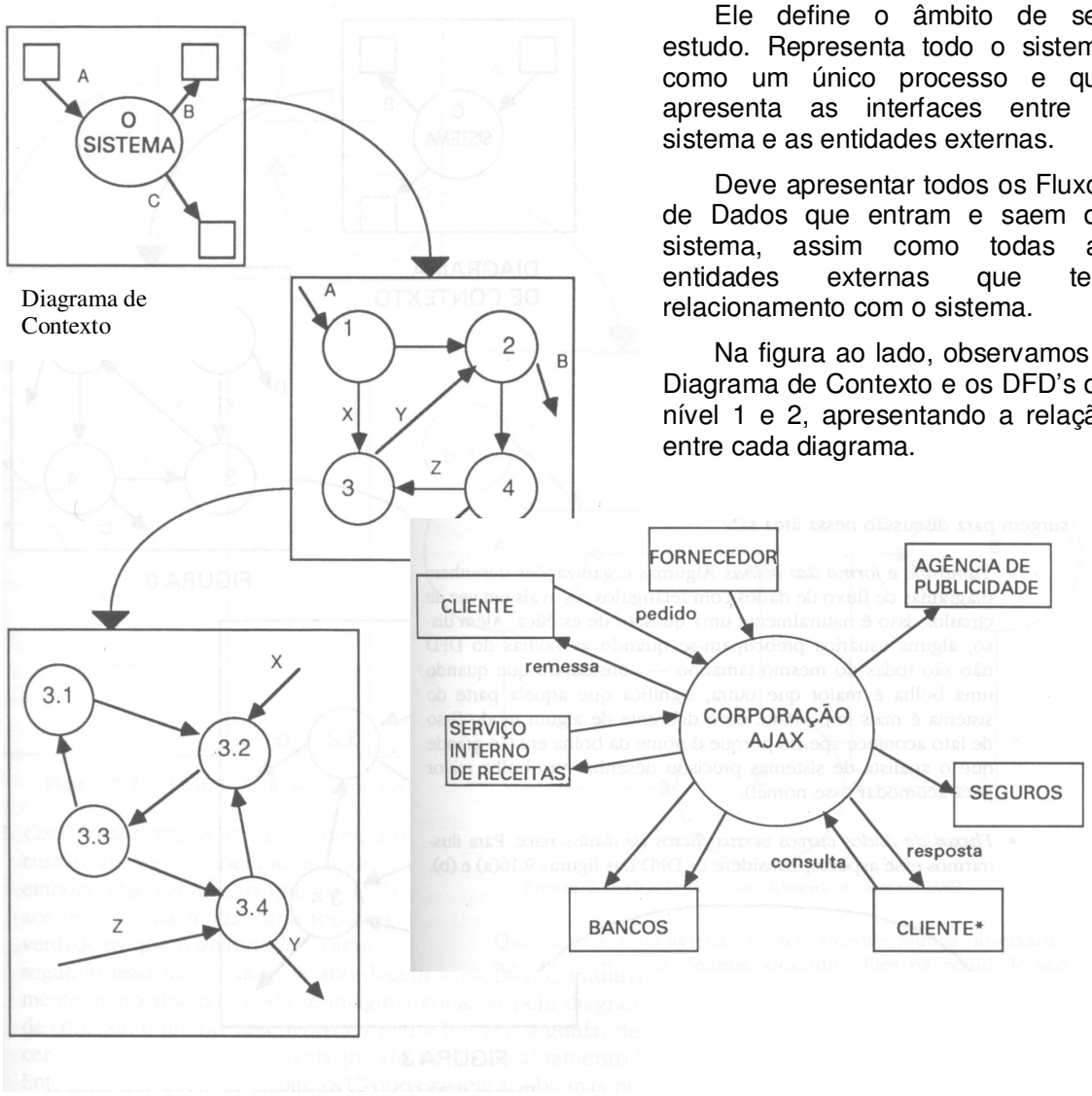
d) Depósito de Dados

Convencionamos a identificação de um depósito de dados pela colocação de uma letra "D" maiúscula seguida de um número, na esquerda do desenho, separada da descrição por uma linha vertical.



e) Diagrama de Contexto

Diagrama de Contexto

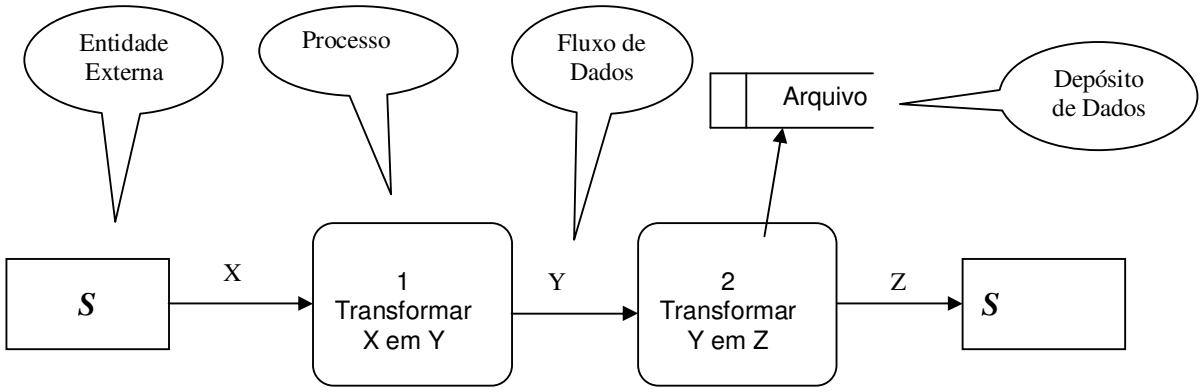


Ele define o âmbito de seu estudo. Representa todo o sistema como um único processo e que apresenta as interfaces entre o sistema e as entidades externas.

Deve apresentar todos os Fluxos de Dados que entram e saem do sistema, assim como todas as entidades externas que tem relacionamento com o sistema.

Na figura ao lado, observamos o Diagrama de Contexto e os DFD's de nível 1 e 2, apresentando a relação entre cada diagrama.

Exemplo:



6.8.2 Dicionários de Dados

O dicionário de dados é um repositório de dados sobre os dados do software. Ele deverá conter a definição dos elementos que tornam o Modelo de Dados e o Diagrama de Fluxo de Dados precisos, quais sejam:

- Fluxos de dados;
- Depósitos de Dados/Entidades;
- Atributos.

Regras para Formação de Nomes:

- O nome deve ser formado por palavras separadas por sublinha até o máximo de 32 caracteres;
- Preferencialmente a nomeação deve ser feita de acordo com o usuário;
- Devem ser eliminados proposições e conjunções;
- Quando houver necessidade de abreviar uma palavra, observar que a abreviatura seja clara, ou inclui-la no dicionário.

Notação:

Símbolo	Significado
=	É composto de
+	E
[ ]	Escolha uma das opções alternativas
{ }	interações de
( )	Opcional (pode estar presente ou ausente)
|	separa opções alternativas na construção [ ]
* *	Comentário
@	Identificador (campo chave) de um depósito.

a) Dado Elementar (Atributo):

i) Com seleção de valores:

NOME_ITEM = [“valor 1”| “valor n”] ft,d onde:

f = formato (A - Alfabético, N - Numérico, X - Caracter Válido);

t = tamanho (em caracteres ou inteiros);

d = decimais.

Exemplo: SEXO=[“M”|“F”] A1 (Alfabético de tamanho 1, podendo assumir valores M ou F).

ii) Sem seleção de valores:

NOME_ITEM = ft,d

Exemplo: VALOR_REPRESENTACAO = N7,2 (Numérico com 7 inteiros e 2 decimais)

iii) Sendo os limites de interação conhecidos:

NOME_ITEM_GRUPO = Y (item_elementar_1 + item_elementar_n) X

onde : X = limite superior, Y = limite inferior.

Exemplo: REFERENCIAS = 2 (NOME + ENDERECO) 10

b) Fluxo de Dados

DETALHE_CUSTO = DESCRICAO_COMPLETA + MATRIZ_CUSTO

A definição demonstra que o fluxo de dados é composto de outros fluxos de dados e/ou dados elementares.

DESCRICAO_COMPLETA = NOME_ITEM + TIPO_ITEM + [PESO | VOLUME]

A definição demonstra que o fluxo e composto por dados elementares e um dos dados deve ser selecionado entre o peso ou o volume.

c) Depósito de Dados

ARQUIVO_DE_CUSTO = 1 (@NUMERO_ITEM + DETALHE_CUSTO) 1000

Os registros de arquivos vão de 1 a 1000 e o atributo NUMERO_ITEM é campo chave.

i) Dados Elementares/Atributos:

TIPO_ITEM = [“SOLIDO” | “LIQUIDO” | “GASOSO”] A7

6.8.3 Descrição de Procedimentos ou Especificação de Processos

A especificação de um processo começa pela sua nomeação, porém é na descrição do processo que o analista deve atentar para atingir o objetivo de fazer uma especificação de processo completa, não-ambígua e não-redundante.

O DFD declara a existência dos procedimentos e das interfaces entre elas, mas e o seu conteúdo? Como alternativas para a descrição de procedimentos podemos ter:

- TEXTO NARRATIVO;
- PORTUGUÊS ESTRUTURADO;
- TABELAS DE DECISÃO;
- ÁRVORES DE DECISÃO;

- FÓRMULAS MATEMÁTICAS;
- ALGUMAS COMBINAÇÕES DOS ACIMA DESCRITAS.

a) Texto Narrativo

É um texto que descreve o que acontece no processo. A dificuldade do texto narrativo é que inconsistências lógicas podem se esconder mais facilmente na descrição. Sempre na terceira pessoa. Exemplo:

> O cálculo da depreciação deverá ser efetuado da seguinte forma:
>
> - Quando o VALOR_CAPITAL_ATUAL for inferior a R$ 1.000,00 e a CATEGORIA do bem sobre o qual vai ser calculada a depreciação for igual a "X", o valor da DEPRECIACAO deverá ser igual ao VALOR_CAPITAL_ATUAL e o VALOR_NOVO_CAPITAL deverá ser reduzido a zeros.
>
> - Quando o VALOR_CAPITAL_ATUAL for inferior a R$ 1.000,00 e a CATEGORIA do bem sobre o qual vai ser calculada a depreciação for igual a "Y", o valor da DEPRECIACAO deverá ser 20% do VALOR_CAPITAL_ATUAL e o VALOR_NOVO_CAPITAL deverá ser 80% do VALOR_CAPITAL_ATUAL.
>
> - Quando o VALOR_CAPITAL_ATUAL for superior a R$ 1.000,00 independentemente do valor da CATEGORIA, se "X" ou "Y", o valor da DEPRECIACAO deverá ser igual 35% do VALOR_CAPITAL_ATUAL e o VALOR_NOVO_CAPITAL deverá ser 65% do VALOR_CAPITAL_ATUAL.
>
> - Quando o VALOR_CAPITAL_ATUAL for igual a R$ 1.000,00 preceder de conformidade com o item anterior.

b) Português Estruturado

i) Linguagem Estruturada

É uma subconjunto da linguagem normal com algumas restrições quanto ao tipo de sentenças que podem ser utilizados e a maneira como essas sentenças podem ser reunidas. O português estruturado usa a sintaxe do português diário, mas restringe a construção de sentenças simples e a poucas e necessárias construções lógicas.

ii) Vocabulário. Consiste de:

- verbos no imperativo
- termos definidos no dicionário de dados
- algumas palavras reservadas para formulação lógica.

iii) Sintaxe. A sintaxe de uma afirmativa está limitada a estas possibilidades:

- sentença declarativa simples
- construção de decisão
- construção de repetição
- combinação das anteriores

Portanto, tem-se:

- **Construção de seqüência** – consiste em uma ou mais programas de ação subordinados que são aplicados um após o outro sem interrupção.
- **Construção de repetição** – um programa de ação subordinada que é feito repetidas

vezes dentro de algum limite.

- **Construção de decisão** - consiste em dois ou mais programas de ação subordinados, apenas um deles se aplica eu qualquer caso determinado.

As três construções tem um único ponto de entrada e um único ponto de saída.

Exemplos:

```
Programas de ação: Peça renovação de estoque.
    Para cada: Requisição–renovação, faça o seguinte:
        1- Encontrar papel–autorização de forma que nro-referencia igual ao nro- requisição da
           requisição – renovação.
        2- Se não combinam rejeitar requisição–renovação
           caso contrário:
               Escrever: ordem – compra para item-pedido.
               Escolher fornecedor para o qual item–pedido aparece no catalogo–fornecedor.
               Copiar nome-e-endereço-fornecedor na ordem-compra.
               copiar nro-ordem-compra na requisição-renovação
               Arquivar requisição-renovação com papel-autorização.
```

```
Processo 3.3 – Calcular valor do Capital
    Se VALOR_CAPITAL_ATUAL <= 1.000
                        Se CATEGORIA = X
                                Faça DEPRECIACAO = VALOR_CAPITAL_ATUAL
                                Faça VALOR_NOVO_CAPITAL = 0
                        Caso Contrario (Categoria = Y)
                                Faça DEPRECIACAO = VALOR_CAPITAL_ATUAL * 0,20
                                Faça VALOR_NOVO_CAPITAL =
    VALOR_CAPITAL_ATUAL * 0,80
    Caso  Contrário (VALOR_CAPITAL_ATUAL >= 1.000)
                        Faça DEPRECIACAO = VALOR_CAPITAL_ATUAL * 0,35
                        Faça VALOR_NOVO_CAPITAL = VALOR_CAPITAL_ATUAL * 0,65
```

c) Tabelas de Decisão

Existem situações em que nem a linguagem estruturada e nem o texto narrativo são adequadas para se elaborar uma especificação de processo. Isso é especialmente verdadeiro quando o processo deve produzir alguma saída ou executar ações com base em decisões complexas. Se as decisões forem baseados em diversas variáveis, e se essas variáveis puderam assumir muitos valores diferentes, então a lógica expressa pela linguagem estruturada será complexa e o usuário provavelmente não entenderá. Uma tabela de decisão é criada relacionando-se todas as entradas (ou condições) e toda as ações relevantes no lado esquerdo da tabela. Cada combinação possível de valores das variáveis é listada em uma coluna. denominada norma. Uma norma descreve a ação (ou ações) que deve(m) ser tomada(s) para uma específica combinação de valores das variáveis. Pelo menos uma ação deve ser específica para cada norma.

Se houver N variáveis com valores binários (falso / verdadeiro), então haverá 2N normas distintas. Assim se houver 3 condições, haverá 8 normas.

Essa técnica é útil para descrever múltiplas condições inter-relacionadas e para detectar especificações incompletas.

Deve-se seguir as seguintes etapa para a criação de uma tabela de decisões para uma especificação de processos:

- Identifique todas as condições ou variáveis na especificação. Identifique todas os valores que cada variável pode assumir.
- Calcule o número de combinações de condições
- Identifique cada ação possível na especificação.
- Crie uma tabela de decisões vazia, relacionando todas as condições e ações no lado esquerdo e numerando as combinações de condições no alto da tabela.
- Relacione todas as combinações de condições, uma para cada coluna vertical da tabela.
- Examine cada coluna vertical (norma) e identifique as ações adequadas a serem empreendidas.
- Identifique todas as omissões, contradições e ambigüidades da especificação.
- Discuta as omissões, contradições e ambigüidades com o usuário.

Exemplo:

i) Processo 3.3 – Calcular valor do Capital

Condições	**1**	**2**	**3**	**4**
1. Categoria (X,Y)	X	Y	X	Y
2. VALOR_CAPITAL_ATUAL = R$ 1.000,00 (<, >=)	<	<	≥	≥
Ações				
1. Faça Depreciação igual a (% do VALOR_CAPITAL_ATUAL)	100	20	35	35
2. Faça VALOR_NOVO_CAPITAL igual a (% do VALOR_CAPITAL_ATUAL)	0	80	65	65

ii) Processo 4.3 – Distribuição da medicação

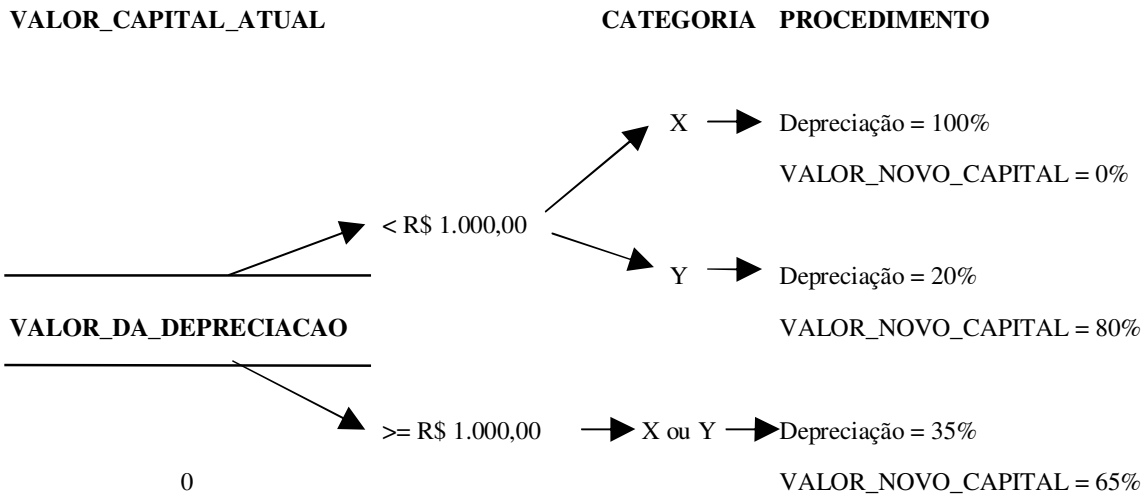
	1	2	3	4	5	6	7	8
Idade > 21	S	S	S	S	N	N	N	N
Sexo	M	M	F	F	M	M	F	F
Peso > 150	S	N	S	N	S	N	S	N
Medicação 1	X				X			X
Medicação 2		X			X			
Medicação 3			X			X		X
Nenhuma Medicação				X			X	

d) Árvores de Decisão

Tem as mesmas utilizações da tabela de decisão, diferindo somente na construção.

Exemplo:

Processo 3.3 – Calcular valor do Capital



e) Condições Pré-Pós

É um modo prático de descrevermos a função que deve ser executada por um processo, sem que seja necessário a descrição de um algoritmo ou sobre o procedimento que será empregado. Uma especificação tem duas partes principais: pré condições e pós condições.

i) PRÉ CONDIÇÕES - descrevem tudo que dever ser verdadeiro (se houver) antes que o processo inicie. São descritas como :

- que entradas devem estar disponíveis
- que relacionamentos devem existir entre as entradas ou no interior delas.
- que relacionamentos devem existir entre entradas e depósitos de dados.
- que relacionamentos devem existir entre diferentes depósitos ou no interior de um depósito.

ii) PÓS CONDIÇÕES – descrevem o que deve ser verdadeiro quando o processo terminar a sua tarefa. São descritas como:

- As saídas que serão geradas ou produzidas pelo processo.
- os relacionamentos que existirão entre os valores de saída e os valores originais de entrada
- os relacionamentos que existirão entre os valores de saída e os valores em um ou mais depósitos.
- as alterações que deverão ser feitas nos depósitos.

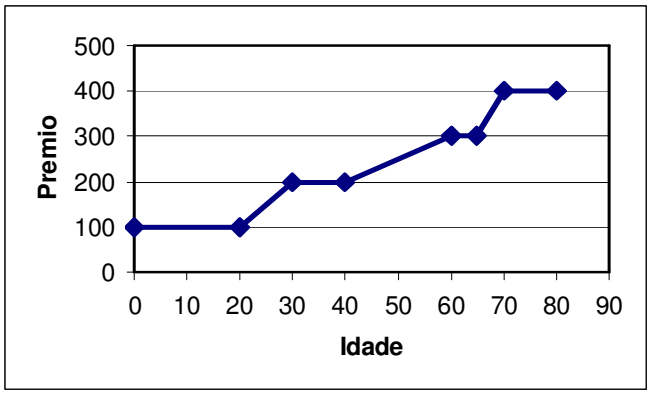
Processo 2.6: **Crédito ao cliente**
Pré Condição 1:
O cliente se apresenta com um nro-de–conta coincidente com um número de conta em contas, cujo cod-de-status esteja ajustado em "válido".
Pós condição 1:
A fatura é emitida contendo nro-de-conta e valor-de-venda.
Pré condição 2:
A pré condição 1 falha por algum motivo (o nro-de-conta não é encantado em contas ou o cod-de-status não é igual a "válido").
Pré condição 2:
É emitida uma mensagem de erro.

f) Grafos e Diagramas

Em alguns casos pode ser apropriado expressar uma especificação de processos por meio de um grafo ou de um diagrama. O usuário pode dispor de um grafo ou diagrama que seja usado para executar aquela parte da aplicação. Não há necessidade de traduzir um grafo para linguagem estruturada.

Exemplo:

Processo 5.4 – Calcular o custo do seguro



7 PROJETO DE TEMPO REAL

7.1 Introdução

- Neste capitulo será apresentada algumas técnicas utilizadas no desenvolvimento de sistemas de tempo real, e que é atualmente uma das atividades mais desafiadoras e complexas que o engenheiro de software tem.
- O software de tempo real é altamente acoplado ao mundo externo, pois deve responder ao domínio do problema (o mundo real) numa escala de tempo por ele ditada. Uma vez que o software deve operar sob rigorosas restrições de desempenho, o projeto é na maioria das vezes baseado na arquitetura de hardware e de software.

Robert Glass escreveu, "O computador digital está se tornando ainda mais presente na vida diária de todos nós. Os computadores permitem que os nossos relógios executem jogos eletrônicos e nos digam a hora, otimizem o consumo de combustível de nossos carros de última geração e coloquem os nossos aparelhos num encadeamento lógico... [Na indústria, os computadores controlam máquinas, coordenam processos e, cada vez mais, substituem as habilidades manuais e o reconhecimento humano por sistemas automatizados e inteligência artificial.]. Todas essas interações computadorizadas – sejam elas úteis ou intrusivas – são exemplo de computação em tempo real. O computador está controlando algo que interage com a realidade numa base oportuna quanto ao tempo. De fato, o timing é a essência da interação... Um sistema de tempo real indiferente pode ser pior do que nenhum sistema em absoluto."

Os sistemas de tempo real geram ação em resposta a eventos externos. Para cumprir essa função eles realizam controle e aquisição de dados em alta velocidade sob severas restrições de tempo e confiabilidade. Como estas restrições são muito severas, eles freqüentemente são dedicados a uma só aplicação.

Existem aplicações de tempo real para controle de processos, automação industrial, pesquisa médica e científica, computação gráfica, comunicações, sistemas aeroespacial, instrumentação industrial, etc.

7.2 Integração e Desempenho

Entre as muitas preocupações relacionadas ao projeto de tempo real, encontram-se a coordenação entre as tarefas de tempo real, processamento das interrupções de sistemas, manipulações de entrada/saída para garantir que nenhum dado seja perdido, especificação das restrições de timing internos e externos ao sistema e garantia da precisão de seu banco de dados.

O desempenho do sistema é medido como uma das características relacionadas ao tempo, mas outras medidas, tais como, a tolerância a falhas também são usadas. Certos sistemas são projetados para aplicações em que somente o tempo de resposta ou a taxa de transferência de dados é crítico, outras aplicações exigem otimização de ambos os parâmetros sob condições de carga máxima.

Tempo de resposta do sistema é o tempo dentro do qual um sistema deve detectar um evento interno ou externo e responder com uma ação. Muitas vezes, a detecção do evento e a geração da resposta são simples, o processamento das informações sobre o evento para determinar a resposta apropriada é que pode envolver algoritmos complexos e consumidores de tempo.

Taxa de transferência de dados é a taxa que indica quão rapidamente dados seriais ou paralelos, bem como analógicos ou digitais, precisam ser movimentados para dentro ou para fora do sistema. Os fornecedores de hardware freqüentemente citam valores de capacidade e de timing para características de desempenho. Mas o importante é analisar as condições de desempenho global do hardware e não de cada componente isoladamente. deve-se analisar o desempenho do dispositivo de E/S, latencia de barramento, tamanho do buffer, desempenho de disco e etc.

Freqüentemente exige-se que os sistemas de tempo real processem uma seqüência contínua de dados que chegam. O projeto deve garantir que dados não sejam perdidos, e o sistema deve reagir a eventos assíncronos.

Toda aplicação deve ser confiável, mas os sistemas de tempo real fazem exigências especiais de confiabilidade, de reinicialização e de recuperação após a ocorrência de falhas. Como o mundo real está sendo monitorado e controlado, a perda de monitoração ou controle (ou ambos) é intolerável em muitas circunstancias. Deste modo, os sistemas contem mecanismos de reinicialização e de reparação de falhas e freqüentemente tem uma redundância interna para assegurar o backup.

7.3 Tratamento de Interrupções

Uma característica que serve para distinguir os sistemas de tempo real de qualquer outro tipo é o tratamento de interrupções. Ele deve responder a estímulos externos – interrupções – num tempo ditado pelo mundo externo. Uma vez que múltiplos estímulos estão presentes, as prioridades e interrupções prioritárias devem ser estabelecidas. Portanto, a tarefa mais importante deve ser atendida dentro das restrições de tempo definida, independentemente de outros eventos.

Um evento é uma ocorrência que exige imediato atendimento, e pode ser gerado ou pelo hardware ou pelo software. O estado do programa interrompido é salvo e o controle é passado a uma rotina de atendimento a interrupções que se ramifica para o software apropriado a fim de manipular a interrupção. Após a conclusão do atendimento à interrupção, o estado da máquina é restabelecido e o fluxo de processamento normal continua.

Para manipular interrupções e ainda atender às restrições de tempo do sistema, muitos sistemas operacionais de tempo real fazem cálculos dinâmicos para determinar se as metas do sistema podem ser cumpridas. Se os cálculos demonstrarem que é impossível manusear os eventos que podem ocorrer no sistema e ainda assim atender às restrições de tempo, o sistema deve decidir por um esquema de ação.

7.4 Linguagens de Tempo Real

Uma combinação de características torna uma linguagem de tempo real diferente de uma linguagem de uso geral. Entre estas características incluem-se as capacidades de multitarefa, construções para implementar diretamente funções de tempo real e características de programação modernas que ajudam a garantir a exatidão do programa.

Por causa dos requisitos reais de desempenho e confiabilidade exigidos dos sistemas de tempo real, a escolha de uma linguagem de programação é importante. Muitas linguagens de programação de propósito geral (Fortran, C, Modula2) podem ser usadas efetivamente. Entretanto, uma classe de “linguagens de tempo real” (Ada, Jovial, HAL/S, Chill) é freqüentemente usada em aplicações militares e de comunicações.

7.5 Sincronização e Comunicação de Tarefas

Um sistema de multitarefas deve fornecer um mecanismo para que as tarefas passem informações umas às outras de forma a garantir a sua sincronização. Para essas funções. sistemas operacionais e linguagens com suporte de run-time comumente usam semáforos de fila, mailboxes ou sistemas de mensagens. Os semáforos fornecem sincronização e sinalização, mas não contem nenhuma informação. As mensagens são semelhantes a semáforos, exceto que carregam as informações associadas. As mailboxes não sinalizam, mas ao contrário, contém as informações.

Semáforos de fila são primitivas de software que ajudam a controlar o tráfego, eles proporcionam um modo de direcionar diversas filas, por exemplo, filas de tarefas que aguardam recursos, filas de acesso a banco de dados e dispositivos, filas de recursos e dispositivos. Os semáforos coordenam (e sincronizam) as tarefas em posição de espera com qualquer coisa que elas estejam esperando, sem deixar que tarefas ou recursos interfiram uns nos outros. Os semáforos são comumente usados para implementar e gerenciar mailboxes.

As Mailboxes são locais de armazenagem temporária para mensagens enviadas de um processo a outro. Um processo produz uma peça de informação, coloca-a na mailbox e depois sinaliza a um processo consumidor que há uma peça de informação na mailbox para que ele a use.

Sistema de mensagens é a terceira abordagem de comunicação e sincronização entre processos. Um processo envia uma mensagem a outro processo, este último é ativado pelo sistema de suporte de run-time ou pelo sistema operacional para processar a mensagem.

7.6 Análise e Simulação de Sistemas de Tempo Real

Vê-se um conjunto de atributos dinâmicos, que não podem estar divorciado das exigências funcionais de um sistema de tempo real, tais como:

- Manuseio de interrupções e troca contextual;
- Tempo de resposta;
- Taxa de transferência de dados e throughput;
- Alocação de recursos e manipulação de prioridades;
- Sincronização de tarefas e comunicação inter-tarefas.

Cada um destes atributos de desempenho podem ser especificados, mas é difícil verificar se os elementos do sistema obterão a resposta desejada, se os recursos do sistema serão suficientes para satisfazer os requisitos computacionais ou se os algoritmos de processamento executarão com velocidade suficiente. A análise de sistemas de tempo real exige uma modelagem e uma simulação que capacitem o engenheiro de sistemas a avaliar as questões de timing e de sizing.

7.7 Métodos de Projeto

O projeto de software de tempo real deve incorporar todos os conceitos fundamentais associados ao software de alta qualidade. Além disso, ele apresenta um conjunto de problemas únicos ao projetista:

- Representação de interrupções e troca contextual;
- Concorrência, enquanto manifestada por multitarefas e multiprocessamento;

- Comunicação e sincronização intertarefas;
- Amplas variações nas taxas de dados e de comunicações;
- Representação das restrições de timing;
- Processamento assíncrono;
- Acoplamento inevitável e necessário a sistemas operacionais, hardware e outros elementos externos do sistema.

Nas últimas décadas vários métodos foram propostos, alguns como extensão dos métodos tradicionais e outros especialmente para software de tempo real.

7.8 Um método de Projeto Orientado para o Fluxo de Dados

Os métodos de projetos orientados para o fluxo de dados são os mais amplamente usados na indústria.

Hassan Gomaa desenvolveu extensões às representações de fluxo de dados que oferecem os mecanismos para o projeto de software de tempo real. A abordagem é denominada de Método de Projeto para Sistemas de Tempo Real (Design Method for Real-Time Systems – DARTS), que permite que os projetistas de sistemas de tempo real adaptem técnicas de fluxo de dados a necessidades especiais de aplicações de tempo real.

7.8.1 Requisitos de um método de projeto de Sistemas de Tempo Real

O método de projeto de software de tempo real DARTS baseia-se na notação e abordagem aplicadas ao projeto de software convencional orientado para o fluxo de dados. Para apoiar o projeto de tempo real, os métodos de fluxo de dados devem ser ampliados, oferecendo:

- Um mecanismo para representar a comunicação e a sincronização de tarefas;
- Uma notação para representar a dependência de estado;
- Uma abordagem que “ligue” os métodos de fluxo de dados convencionais ao mundo do tempo real.

7.8.2 Projeto DARTS

Hassan Gomaa descreve a abordagem do DARTS como: “O método de projeto DARTS pode ser imaginado como uma ampliação do método de Projeto Estruturado/Análise Estruturada ao oferecer uma abordagem para estruturar o sistema em tarefas, bem como um mecanismo para definir as interfaces entre tarefas. Nesse sentido, ele lança mão da experiência obtida em processamentos concorrentes. Como acontece com outros métodos de projeto, o DARTS pretende ser iterativo.”

Logo após o modelo de fluxo e um dicionário de dados tiverem sido criados, o sistema deve ser examinado a partir de um ponto de vista diferente. Os DFD’s não descrevem as tarefas assíncronas e concorrentes que implementam o fluxo de dados. Agora, precisamos de uma abordagem que identifique as tarefas do sistema de tempo real no contexto das funções (transformações) do sistema desenhadas num DFD. As transformações são agrupadas em tarefas de tempo real. O fluxo de dados entre tarefas recém-definidas determina os requisitos de comunicação intertarefas. Os critérios para determinar as transformações do DFD devem ser definidas como tarefas separadas ou agrupadas com outras transformações numa única tarefa:

- Dependência de E/S – Dependendo da entrada ou da saída, uma transformação é

limitada a funcionar na velocidade do dispositivo de E/S com o qual está interagindo. Necessita assim de ser uma tarefa separada.

- Funções críticas quanto ao tempo – É uma função que precisa ser executada com prioridade elevada e deve ser uma tarefa separada.
- Requisitos Computacionais – Uma função de uso computacional intensivo, pode ser executada como uma tarefa de prioridade mais baixa, executada em CPU extra.
- Coesão Funcional – Transformações que executem um conjunto de funções estreitamente relacionadas podem ser agrupadas em uma tarefa, implementando as funções como módulos de uma mesma tarefa garantirá a coesão funcional, o que eliminará o tráfego de dados entre tarefas distintas, diminuindo o overhead do sistema.
- Coesão Temporal – Algumas transformações realizam funções que são executadas ao mesmo tempo, devem portanto serem agrupadas em uma tarefa, para serem executadas cada vez que a tarefa receber um estímulo.
- Execução Periódica – Uma transformação que precisa ser executada periodicamente pode ser estruturada como uma tarefa separada que é ativada em intervalos regulares.

Com as tarefas definidas, DARTS proporciona um mecanismo para o manuseio de informações entre as tarefas, ao defini duas classes de “módulos de interface com as tarefas”: os módulos de comunicação com as tarefas (TCM) e os módulos de sincronização de tarefas (TSM).

Um TCM é gerado por uma tarefa comunicante e usa primitivas de sincronização de sistema operacional para garantir o acesso adequado aos dados. Os TCM’s são divididos em duas categorias:

- módulos de comunicação de mensagens (MCM),
- módulos de ocultação de informações.

Quando o controle, em vez de dados, deve ser passado entre as tarefas, o módulo de sincronização de tarefas entra em cena. Uma tarefa pode sinalizar a outra tarefa que um evento ocorreu, ou uma tarefa pode ficar à espera de tal sinal. o TSM pode ser visto como um módulo supervisor que gerencia o controle e a coordenação quando ocorrem eventos.

7.8.3 Projeto de Tarefas

Uma tarefa de sistema de tempo real é um programa que pode ser projetado usando-se métodos convencionais, ou seja, uma tarefa pode ser vista como um processo seqüencial implementado com uma estrutura de programa hierarquicamente organizada, usando a filosofia de programação estruturada.

O DFD que representa o fluxo de dados dentro da fronteira de uma tarefa pode levar a uma estrutura de programa, porém, para sistemas de tempo real, o controle depende tanto da entrada oferecida à tarefa como do estado atual do sistema, por isto, o controle é dependente do estado. Para isto, sugere-se um mecanismo para manejar esta situação que é denominado de gerenciador de transição de estados (STM) que é uma implementação da tabela de ativação de processos. o STM tem conhecimento do estado corrente do sistema e usa uma tabela de transição de estados para guiar o processamento.

8 UML

Os conceitos da orientação a objetos são bem difundidos, desde o lançamento da primeira linguagem orientada a objetos, a linguagem SIMULA. O desenvolvimento de sistemas de software suportados por métodos de análise e projeto que modelam esse sistema de modo a fornecer para toda a equipe envolvida uma compreensão completa do projeto. A UML (Unified Modeling Language) é a sucessora de um conjunto de métodos de análise e projeto orientados a objeto.

A UML é um modelo de linguagem, não um método. Um método pressupõe um modelo de linguagem e um processo. O modelo de linguagem é a notação que o método usa para descrever o projeto. O processo são os passos que devem ser seguidos para se construir o projeto.

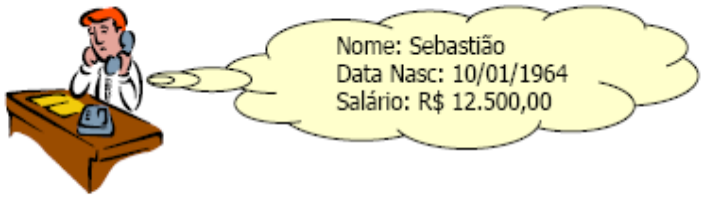
A UML define uma notação e um meta-modelo. A notação representa a sintaxe da linguagem composta por elementos gráficos. Um meta-modelo é um diagrama de classe. A UML foi desenvolvida por Grady Booch, James Rumbaugh, e Ivar Jacobson que são conhecidos como "os três amigos". A UML é a junção do que havia de melhor nas três metodologias e que foi adicionado novos conceitos e visões da linguagem.

8.1 Conceitos

Paradigma - "É uma forma de pensar e perceber o mundo real e determina o que escolhemos como significativo e o que descartamos ao compreender ou descrever o que existe ou ocorre no mundo em torno de nós. A mudança de paradigma é uma oportunidade de encontrar novas interpretações para antigas questões, bem como, para rever soluções tidas como definitivas. Dizemos que a O.O. constitui um novo paradigma computacional pois representa uma mudança na forma de pensar e conceber sistemas e programas de computador. A estratégia de O.O. para modelagem de sistemas baseia-se na identificação dos objetos (que desempenham ou sofrem ações no domínio do problema) e dos padrões de cooperação e interação entre estes objetos."

Abstração - Permite ignorar os aspectos de um assunto não relevante para o propósito. Diminui a complexidade.

Objeto - É alguma coisa que pode ser identificada distintamente. Qualquer coisa, real ou abstrata, à respeito da qual armazenamos dados e os métodos que os manipulam. (Martin/Odell). Uma entidade capaz de armazenar um estado (informação) e que oferece um número de operações (comportamento) para ou consultar ou alterar o estado. (Jacobson) É algo que possui estado, comportamento e identidade. (Booch)



Classe - Pode ser vista como uma fábrica de objetos idênticos. Possui atributos e métodos. Uma descrição de um conjunto de objetos que compartilham os mesmos atributos, operações, relacionamentos e semântica. (Rumbaugh) Um gabarito para diversos objetos que

descreve como estes objetos são estruturados internamente (Jacobson). Um conjunto de objetos que compartilham uma estrutura comum e um comportamento comum (Booch)

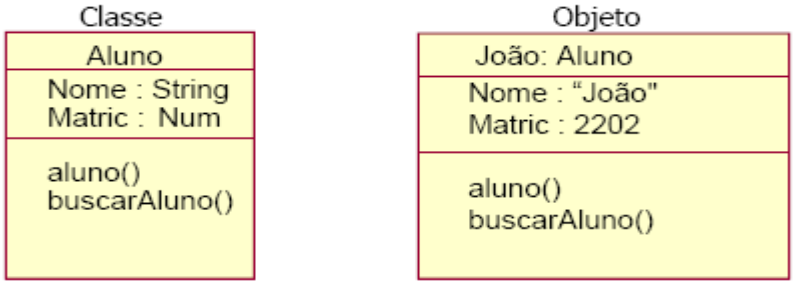
a) Classe x Objeto

Exemplo Classe x Objeto

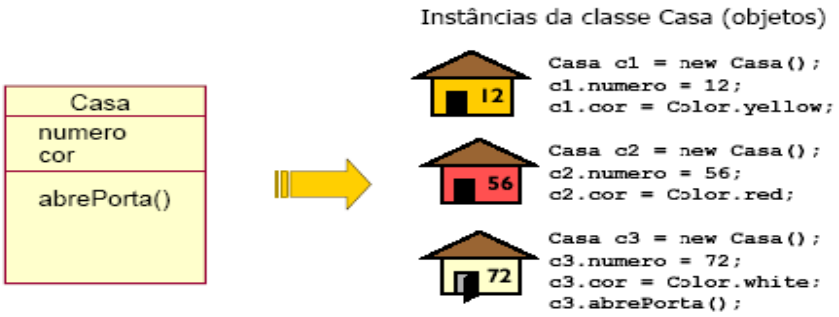
Para criar efetivamente um objeto, nós devemos instanciá-lo:

```
class Aluno {
        private String nome;
        private int matric;
        aluno(String s,int m) {
                nome = s;
                matric = m;
        }
        Aluno a1 = new Aluno("João",2202);  }
```

Método Construtor

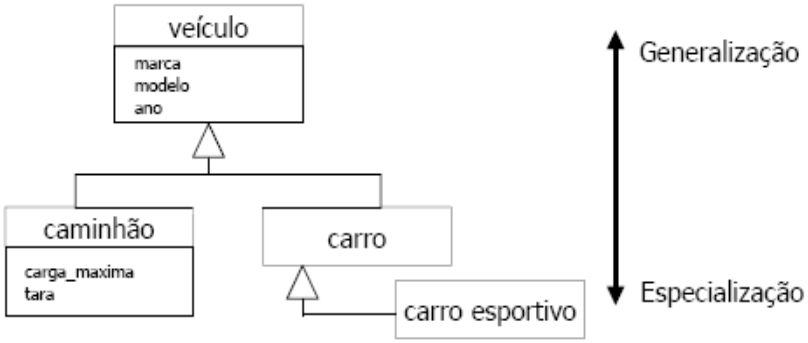


Outro Exemplo Classe x Objeto



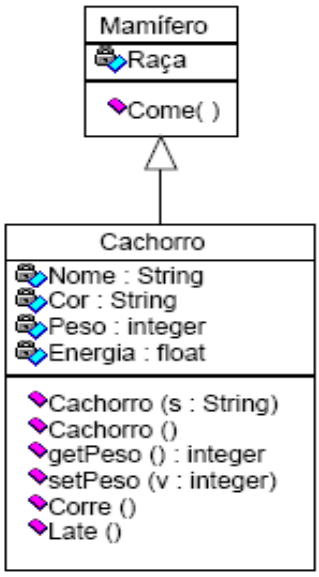
A **classe** casa é como uma forma que irá criar objetos semelhantes. Observe que cada objeto tem característica própria, a cor o número.

Herança - Capacidade de um novo objeto tomar atributos e operações de um objeto existente, permitindo criar classes complexas sem repetir código. Representa generalização e especialização.Uma especialização pode incluir novos métodos e atributos.

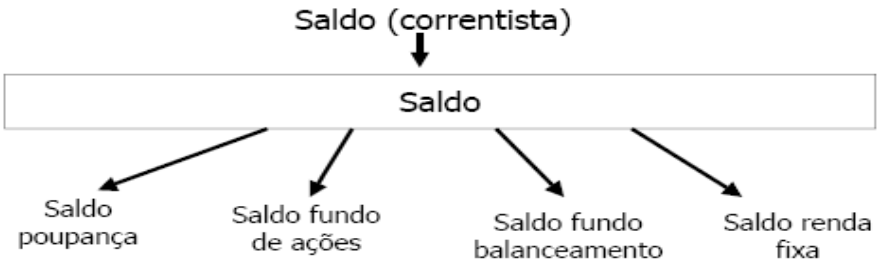


- **Herança - continuação**

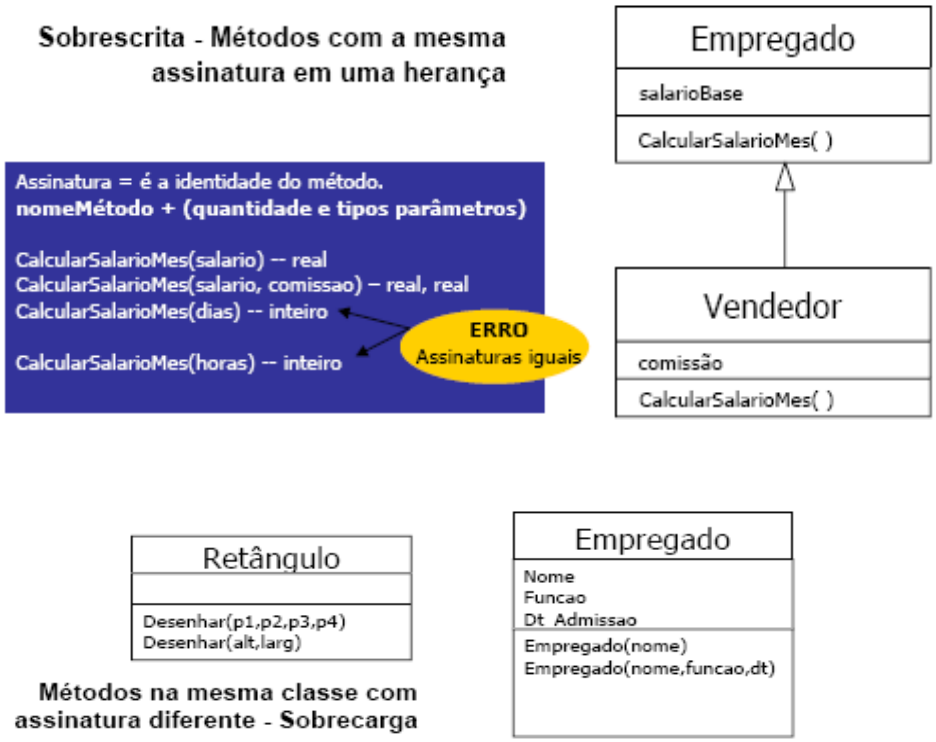
```
class Cachorro extends Mamífero {
    // Atributos dos objetos da classe
    private String nome;
    private String cor;
    private int    peso;
    private float  energia;

    // Construtores (formas da classe)
    Cachorro(String s) { nome = s; }
    Cachorro() { nome = "Sem nome"; }

    // Métodos (comportamentos dos objetos da classe)
    void setPeso(int v) { peso = v; }
    int   getPeso() { return peso; }

    void corre() { ... }
    void late() { ... }
}
```



Polimorfismo (“muitas formas”) - Permite que operações com o mesmo nome, mas com uma semântica de implementação diferente, sejam ativadas por objetos de tipos diferentes. Vários comportamentos que uma mesma operação pode assumir.



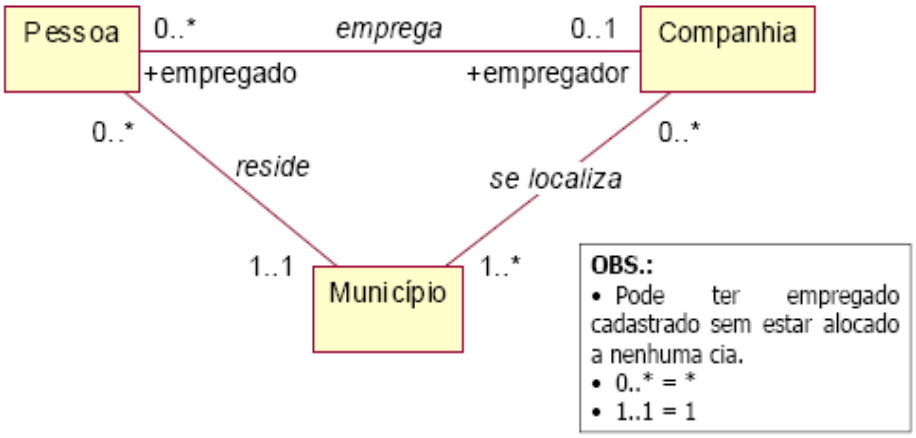
Uma operação é chamada polimórfica quando trabalha de formas diferentes.



Mensagem - É a comunicação entre objetos. A execução de um método é ativada através do envio de uma mensagem. Persistência O objeto pode sobreviver após a consecução do sistema, a existência do objeto pode persistir ao tempo, isto significa que o objeto deve ser armazenado de alguma forma. Um objeto não persistente é também chamado de transiente. Estereótipo É um mecanismo de extensão e estende a semântica de meta-modelo. Podem ser criados pelo usuário através de ícones ou entre aspas << >>. Existem estereótipos pré-definidos na UML (será visto adiante).

Encapsulamento é o processo de impedir que os atributos de uma classe sejam lidos ou modificados por outras classes. Ele garante a inviolabilidade dos dados aumentando a robustez do sistema. Os atributos (valores armazenados) de um objeto devem ser consultados ou modificados através dos métodos da classe do objeto. O uso de métodos como interface deve garantir o reaproveitamento e atualização do objeto. Um objeto externo tem acesso aos métodos públicos e estes tem garantido o acesso a atributos definidos na classe. Existem também métodos privados, acessíveis somente por métodos do mesmo objeto. Vamos ver visibilidade ao final do diagrama de classes.

Papel É a face que a classe próxima a uma das extremidades apresenta à classe encontrada na outra extremidade da associação. A mesma classe pode executar papéis iguais ou diferentes em outras associações.



b) Os Diagramas

Diagrama – é uma apresentação gráfica de um conjunto de elementos; é uma projeção em um sistema. A UML inclui nove diagramas:

Classes – diagrama estrutural que mostra um conjunto de classes, interfaces, colaborações e seus relacionamentos.

Objetos – diagrama estrutural que mostra um conjunto de objetos e seus relacionamentos.

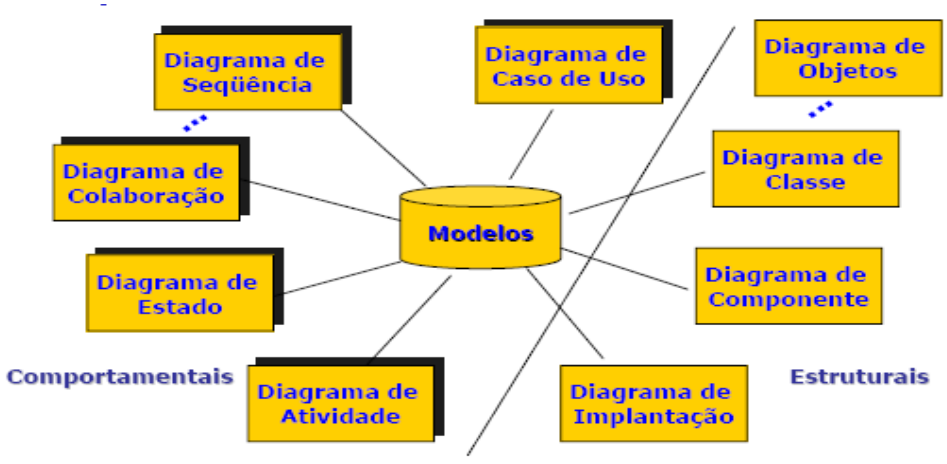
Casos de Uso – diagrama comportamental que mostra uma interação, dando ênfase à organização estrutural de objetos que enviam e recebem mensagens.

Interação (Seqüência e Colaboração) – descrevem o comportamento do sistema de acordo com o tempo e a troca de mensagens entre os objetos. São levantadores de métodos.

Gráfico de Estados – diagrama comportamental que mostra uma máquina de estados, dando ênfase ao comportamento ordenado por eventos de um objeto.

Atividades – diagrama comportamental que mostra uma máquina de estados, dando ênfase ao fluxo de uma atividade para outra.

Componentes – diagrama estrutural que mostra um conjunto de componentes e seus relacionamentos. Implantação – diagrama estrutural que mostra um conjunto de nós e seus relacionamentos.



Os diagramas: de caso de uso, sequência, colaboração, de estado e de atividade são ditos Comportamentais pois modelam aspectos dinâmicos do sistema, mostram, na maioria das vezes, como as entidades interagem para a execução de uma funcionalidade. Os outros diagramas (de classe, de objetos, de componente e de implantação) são estruturais pois modelam, como o nome diz, a estrutura do sistema, sua parte estática, mostram como as entidades são compostas e seus relacionamentos.

c) Exemplo

Sistema de Controle de Horas Para analisar cada diagrama vamos nos basear em um sistema de alocação das horas trabalhadas – Time Sheet O Sistema de Controle de Horas funciona para que a empresa tenha controle sobre as horas trabalhadas dos seus funcionários. Todos os funcionários deverão informar o projeto que trabalharam, quais foram as atividades desempenhadas e o período. Os gerentes ou coordenadores deverão aprovar (ou reprovar) as horas cadastradas dos funcionários para os projetos de sua responsabilidade.

Diagrama de Caso de Uso - Especifica uma interação entre um usuário e o sistema, no qual o usuário tem um objetivo muito claro a atingir.

- O funcionário cadastra alocação das suas horas;
- O gerente aprova horas;
- O funcionário consulta as horas trabalhadas.

Imagine a tela ou o conjunto de telas que fará parte de uma funcionalidade e imagine a interação do usuário com o sistema, qual a ação do usuário e como o sistema irá reagir, quais os campos que terão lista de bancos, quais os campos que o usuário deverá informar valores – aqui devemos prever todas as reações do sistema – se o usuário digitar letra no lugar do número, como o sistema irá reagir ? Neste momento podemos prever inclusive quais as mensagens de alerta/erro que retornarão ao usuário, afinal de contas o caso de uso servirá de base para os testes.

Diagrama de Caso de Uso:

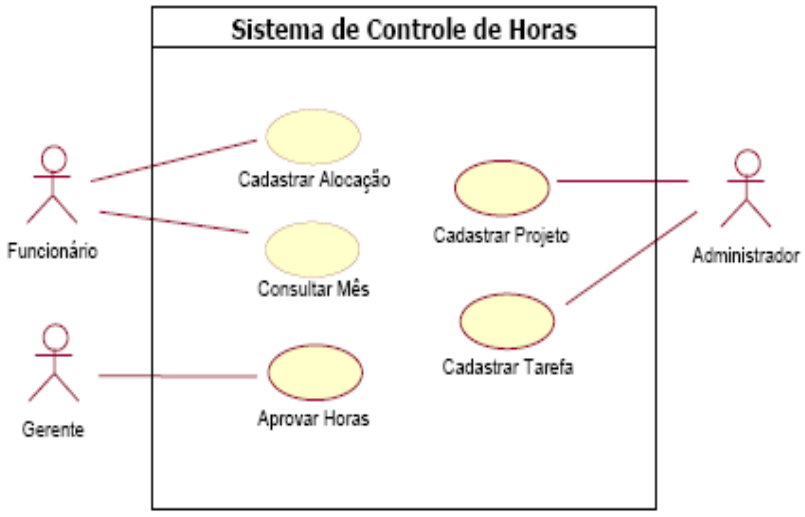


Diagrama de Caso de Uso: Alguns casos de uso que podem ser aplicados ao sistema de controle de horas são (este é um exemplo os casos de uso não estão levantados na totalidade do sistema):

- O funcionário cadastra tarefa: para o cadastramento da tarefa será necessária a interação entre o funcionário e o sistema pois o funcionário deverá informar: a data da atividade, o horário (hora início e fim) de sua realização, o sistema deverá exibir o

logon do funcionário e uma lista de atividades e projetos, o funcionário deverá selecionar uma categoria e uma OP e opcionalmente informar FSA/OS, Sistema, Aplicação e Observação e então solicitar a confirmar a tarefa, o sistema deverá gravar a tarefa e exibir a lista de tarefas.

- O funcionário consulta mês: o funcionário seleciona (<< ou >>) e navega pelos meses e suas tarefas cadastradas.
- O gerente aprova hora: O gerente deverá informar o projeto e o sistema irá listar todos os funcionários que lancaram horas no projeto e que ainda estão aguardando aprovação, o gerente seleciona o funcionário e seleciona todas as horas do funcionário e então confirma a aprovação das suas horas.

Diagrama de Caso de Uso - Descrição

Nome: Cadastrar Alocação

Descrição: O Cadastro da alocação pode ser feito por período (data inicial e final) se o funcionário estiver trabalhando na mesma atividade e projeto e pode também ser feita pontualmente caso o funcionário trabalhe para um projeto fora do seu cotidiano.

Ator: Funcionário

Curso Normal: – Cadastro de Período (sistema cadastra trabalho de 9:00 até as 18:00hs)

1. Funcionário informa data inicial e final
2. Sistema exibe logon do usuário cadastrado e lista Tarefas e Projetos
3. Funcionário seleciona tarefa e projeto e informa OS, Sistema e Observação
4. Funcionário confirma cadastro de trabalho
5. Sistema registra Alocação

Cursos Alternativos:

- Passo 1 - Cadastro de Alocação Pontual

1. Funcionário informa data e hora início e fim.
2. Retornar ao passo 2 do curso normal.

Exceções:

- Passo 1 – Caso data inválida

1. Sistema exibe mensagem de data inválida.
2. Retornar ao passo 1 do curso normal.

- Passo 5 – Caso Tarefa, Projeto ou Data não informados:

1. Sistema exibe mensagem "a tarefa, o projeto e a data devem ser informados".
2. Retornar ao passo 3 do curso normal.

Diagrama de Classe - A próxima tarefa é a classificação dos objetos envolvidos neste processo e a relação de uns com os outros. Diagramas de classe mostram a estrutura geral do sistema e também as suas propriedades relacionais e de comportamento.

- Funcionário;
- Horas Trabalhadas;
- Projeto;
- Tarefa.

No Diagrama de Classe os objetos envolvidos no sistema de controle de horas são agrupados em classes. Uma classe é um conjunto de objetos. Na classe de funcionários, os atributos incluem o logon, nome, unidade, filial e número funcional. Esta classe também armazena a operação buscarLogon (dentre outras aqui ainda não definidas). Os Diagramas de Classe suportam herança de outros sistemas.

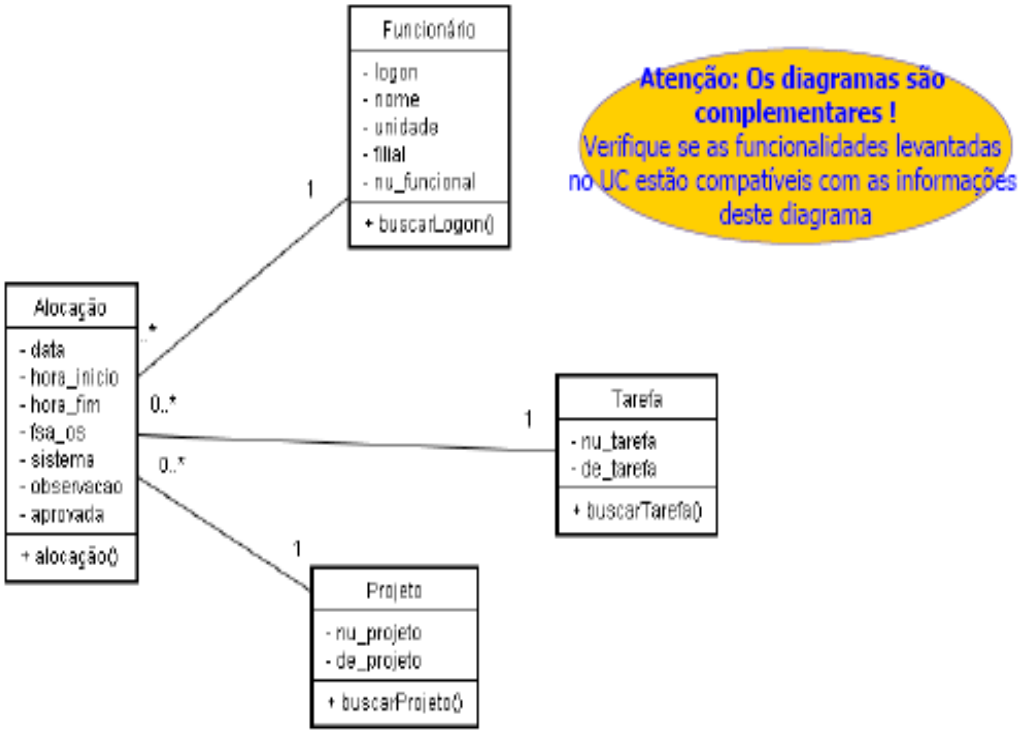


Diagrama de Seqüência - Mostra uma interação organizada em forma de uma seqüência, dentro de um determinado período de tempo. Os participantes são apresentados dentro do contexto das mensagens que transitam entre eles. O diagrama de seqüência é um diagrama de interação.

Um Diagrama de Seqüência oferece uma visão detalhada de um caso de uso. Ele mostra uma interação organizada em forma de uma seqüência, dentro de um determinado período de tempo, e contribui para que se processe a documentação do fluxo de lógica dentro da aplicação. Os participantes são apresentados dentro do contexto das mensagens que transitam entre eles. Num sistema de software abrangente, um Diagrama de Seqüência pode ser bastante detalhado, e pode incluir milhares de mensagens.

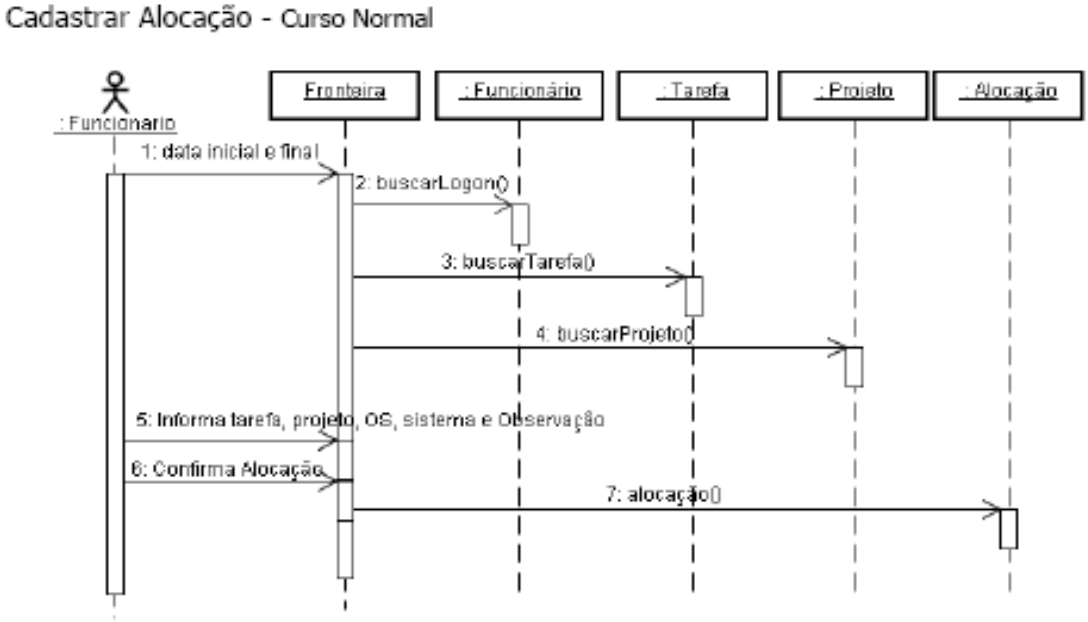
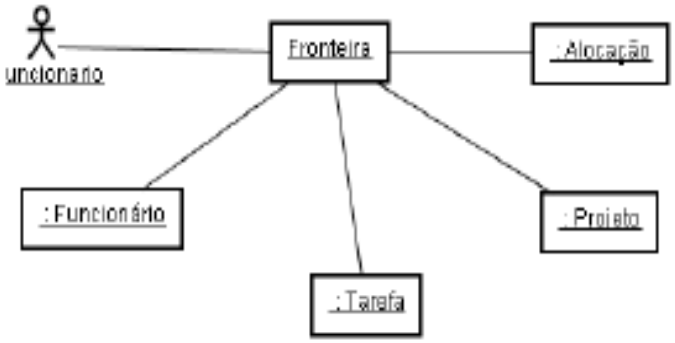


Diagrama de Colaboração - Mostra como um grupo de objetos num caso de uso interage com os demais. Cada mensagem é numerada para documentar a ordem na qual ela ocorre. O diagrama de colaboração também é um diagrama de interação.

Um Diagrama de Colaboração é outro tipo de diagrama de interação. Assim como no Diagrama de Seqüência, o Diagrama de Colaboração mostra como um grupo de objetos num caso de uso interage com os demais. Cada mensagem é numerada para documentar a ordem na qual ela ocorre. Os diagramas de colaboração e de seqüência são equivalentes a diferença é que o diagrama de seqüência tem o tempo como participante fundamental.



O diagrama da figura é equivalente ao diagrama de seqüência anterior. Podemos dizer que o diagrama de colaboração e o diagrama de seqüência são isomórficos, ou seja, podemos obter uma visão a partir da outra. Enquanto o diagrama de seqüência tem uma visão temporal, o diagrama de colaboração apresenta uma visão espacial dos objetos.

Diagrama de Estado - Mapeia diferentes estados em que se encontram os objetos, e desencadeia eventos que levam os objetos a se encontrarem em determinado estado em um dado momento.

O estado de um objeto é definido pelos seus atributos em um determinado momento. Os objetos se movem através de diferentes estados, por serem influenciados por estímulos externos. O Diagrama de Estado mapeia estes diferentes estados em que se encontram os objetos, e desencadeia eventos que levam os objetos a se encontrarem em determinado estado em um dado momento.

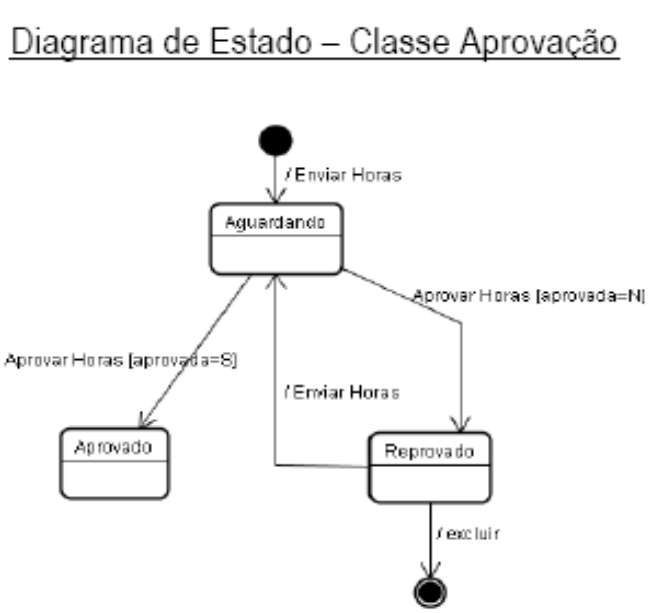


Diagrama de Atividade - Descreve a seqüência de atividades, com suporte para comportamento condicional e paralelo. Componentes:

- Atividade (ou estado de atividade): é o estado de estar executando algo.
- Comportamento condicional: a) decisão (branches) - uma transição de entrada única com várias saídas, na execução só pode ter um fluxo de saída, ou seja as saídas são mutuamente exclusivas. b) intercalações (merges) - múltiplas transições de entrada

convergindo para uma de saída, marca o final de um comportamento condicional.

- Comportamento paralelo: a) Junção (joins) - converge duas ou mais transações em uma, mas esta só ocorre se as iniciais tiverem sido terminadas. b) Separação (forks) - tem uma transição de entrada e várias transições, em paralelo, de saída.

O Diagrama de Atividade é uma ferramenta útil para desenhar a solução de implementação e pode estar baseada em um caso de uso ou pode definir um método mais complexo. Esta figura mostra as atividades necessárias, do sistema, para a execução do Cadastro de Alocação, exibindo todos os cenários do caso de uso (uso da decisão). Observe que há atividades que podem ser executadas em paralelo (threading). O diagrama de atividade é baseado em um caso de uso inteiro (todos os cenários) ou em uma classe.

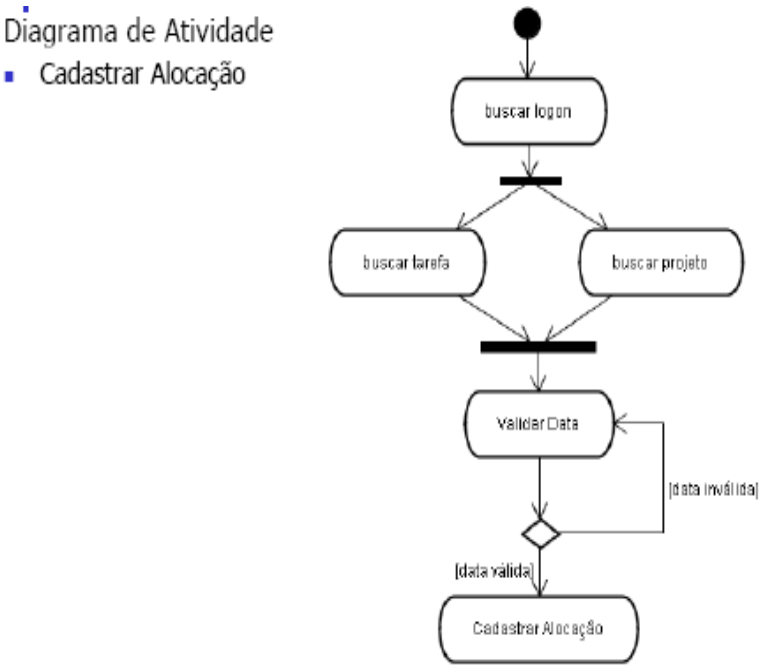
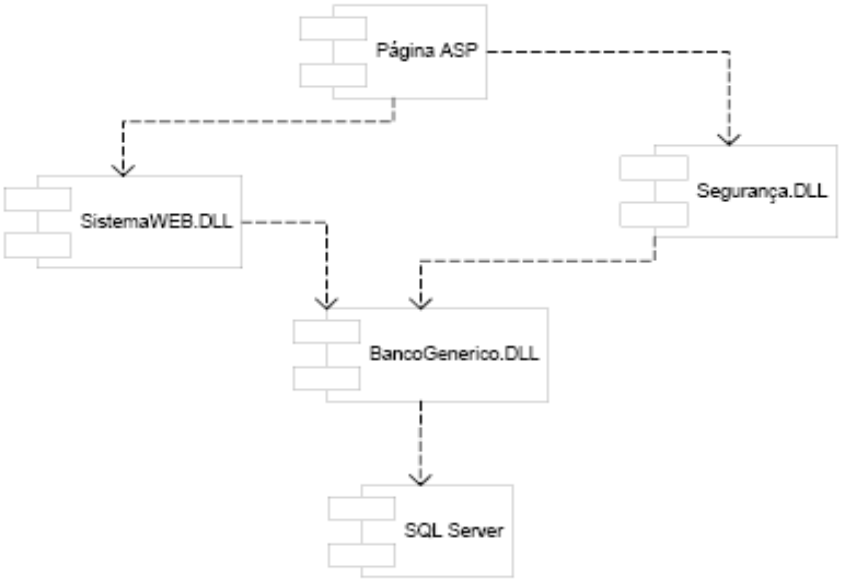


Diagrama de Componentes - Mostra como os diferentes subsistemas de software formam a estrutura total de um sistema.

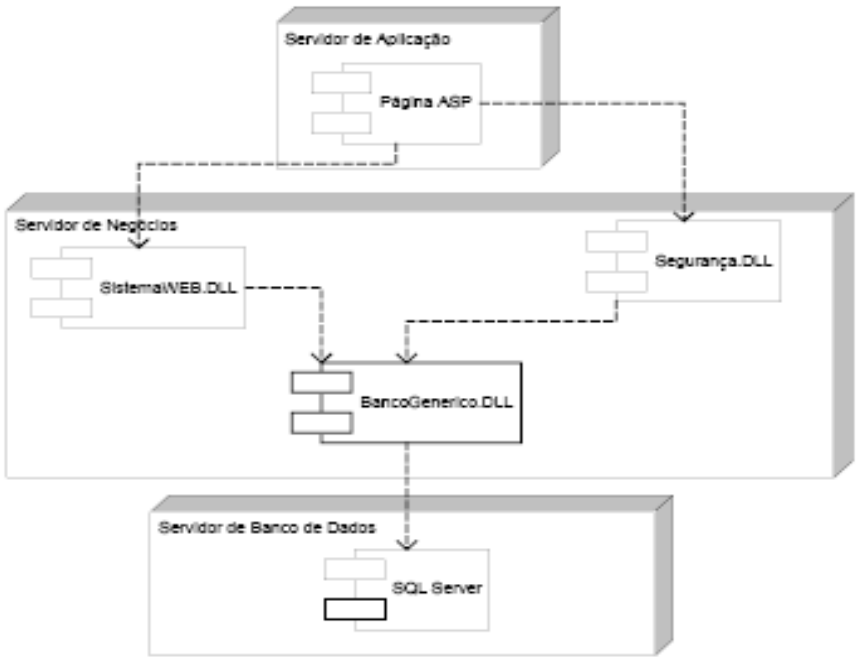
Um Diagrama de Componentes exibe a hierarquia de dependência entre os componentes do sistema. Um componente pode ser entendido como uma tela do sistema, uma biblioteca (dlls), um arquivo de críticas (arquivos js), um executável, etc.



O site tem várias camadas, as páginas asp chamam algumas regras, neste diagrama vemos qual a dependência entre elas. Lembre que os componentes são bibliotecas, executáveis, telas, tabelas, uma parte qualquer do sistema.

Diagrama de Implantação - Mostra como estão configurados o hardware e o software dentro de um determinado sistema.

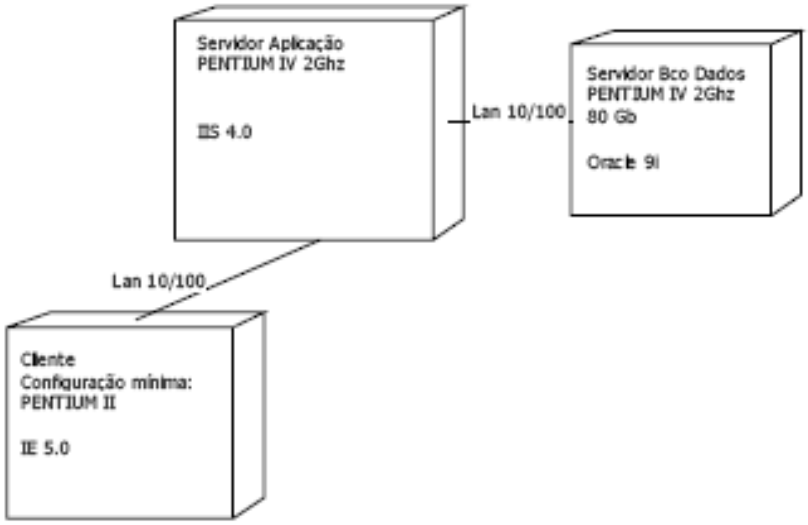
O Diagrama de Implantação mostra o layout físico da rede, onde cada nó representa uma máquina, podendo mostrar a configuração delas (hardware e software) e também onde cada componente irá ser instalado.



A locadora tem a necessidade de processar seus dados num sistema cliente-servidor com um banco de dados centralizado, contendo todos os registros que os profissionais da empresa terão que acessar. Os representantes de locação de veículos precisam ter acesso imediato aos dados sobre a disponibilidade de veículos. Por outro lado, os mecânicos precisam ter meios para destacar que determinado carro está passando por um processo de manutenção.

Exemplo de Diagrama de Implantação:

O diagrama de implantação poderá ser usado para caracterizar o hardware e o software necessário a ser instalado nos nós (máquinas ou periféricos). Inclusive o barramento de conexão entre os nós.



8.2 Casos de Uso

Caso de Uso é: "Um conjunto de seqüências de ações que um sistema desempenha para produzir um resultado observável de valor a um ator específico". Deve ser usado quando se deseja visualizar o comportamento de vários objetos dentro de um único caso de uso.

Objetivo dos casos de uso: a) Descrever os requisitos funcionais do sistema de maneira consensual entre usuários e desenvolvedores de sistemas; b) Fornecer uma descrição consistente e clara sobre as responsabilidades que devem ser cumpridas pelo sistema, além de formar a base para a fase de desenho; c) Oferecer as possíveis situações do mundo real para o teste do sistema.

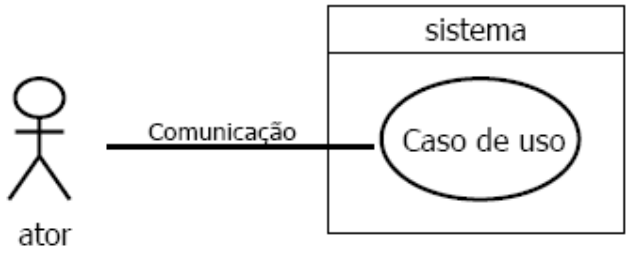
Requisito é uma característica do sistema ou a descrição de algo que o sistema é capaz de realizar para atingir seus objetivos. O requisito pode ser de dois tipos:

- **Funcional** quando descreve uma interação entre o sistema e seu ambiente. Exemplo: Cadastro do Cliente, Consulta à pedidos, etc; ou
- **Não Funcional** (também chamado de requisito de qualidade) quando descreve uma restrição do sistema. Exemplo: amigável, tempo de resposta de 3 segundos, etc.

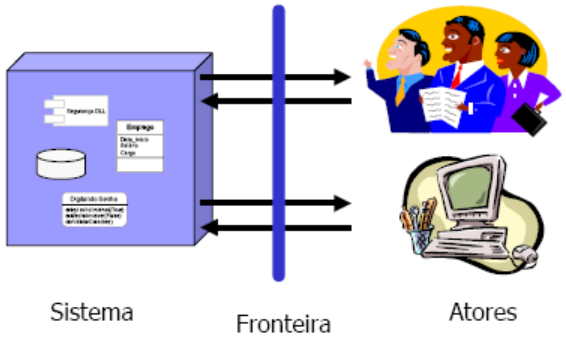
Cenário – descreve a interação entre o ator e o sistema. Um caso de uso pode ter várias terminações (sucessos e insucessos) cada enredo, cada instância é chamada de cenário.

Iterar - Tornar a dizer; repetir Interação - Ação recíproca de dois ou mais corpos, uns nos outros.

A interação ou comunicação é o relacionamento entre o ator e o sistema. Pode existir relacionamento entre casos de uso (extensão, inclusão e generalização).



Os **atores** (pessoas, outros sistemas, periféricos) interagem com o sistema através de uma fronteira. No diagrama de caso de uso teremos todas as funcionalidades do sistema. Para cada funcionalidade iremos descrever as interações entre o ator e as reações do sistema. O que constitui um bom ator ?



- Atores não são parte do sistema, eles interagem com o sistema. Fornecem dados e/ou recebem informação do sistema;
- Uma mesma pessoa pode assumir papéis diferentes;
- Várias pessoas podem assumir o mesmo papel;
- Identificando o ator:

a) O ator Cliente é o mesmo que ator Inadimplente ? – se o inadimplente usar o sistema de forma diferente (somente poderá consultar novas formas de pagamento) eles são atores diferentes.

b) Não devemos criar ator para cada coadjuvante, temos que identificar um grupo de personagens que desempenham o mesmo papel, por exemplo: Gerente de Projeto e o Coordenador da Equipe acessam as mesmas telas e desempenham o mesmo trabalho no sistema, portanto pertencem ao mesmo grupo de personagens de Gerência.

Um **Ator** é uma classe com um ícone padrão.

Exemplos de atores:

- Cliente;
- Sistema de RH;
- Gerente;
- Atendente;
- Sistema de Contas a Pagar;
- Scanner;
- Leitor ótico.

O ator é aquele que utiliza o sistema para passar informações. Pode ser outro sistema, um hardware, um grupo de pessoas. Mas têm que necessariamente interagir com o sistema. Como Identificar os Atores:

- Quem irá usar as funcionalidades básicas do sistema?
- Existe funcionalidade para administrar e manter o sistema? Quem irá executar tais atividades?
- Algum dispositivo de hardware inicializa alguma funcionalidade, ou interage de alguma forma com o sistema?
- O sistema irá interagir com outros sistemas?

Lembre-se: o ator interage com o sistema.

Conceito de **Caso de Uso** - É uma seqüência de ações que um sistema desempenha para produzir um resultado observável por um ator específico. É uma classe, não uma instância. O nome do Caso de Uso deve ser uma frase indicando a ação que realiza. Um caso de uso é um conjunto de passos (é a descrição da interação entre ator e sistema) e o tratamento das suas exceções. Um caso de uso tem início, meio e fim.

O Caso de Uso em si é uma seqüência de ações que um sistema desempenha para produzir um resultado observável por um ator específico. Em outras palavras, um caso de uso define uma funcionalidade do sistema com um conjunto de ações tomadas pelo ator e a previsão da reação por parte do sistema. O Caso de Uso é uma classe, não uma instância. A sua especificação descreve a funcionalidade como um todo, incluindo erros, possíveis alternativas e exceções que podem ocorrer durante sua execução. O nome do Caso de Uso deve ser uma frase indicando a ação que realiza. Cuidado para não identificar um caso de uso no lugar de um passo! Um caso de uso tem um conjunto de passos e trata as exceções desses passos. Na descrição do caso de uso é que teremos que pensar quais as ações que o caso de uso desempenhará. Alguns exemplos de casos de uso: Cadastrar Cliente, Registrar Venda, Fechar Caixa, etc.

Características e Regras - É sempre inicializado por um ator, que pode fazê-lo direta ou indiretamente no sistema. São conectados aos atores através de associações de comunicação.

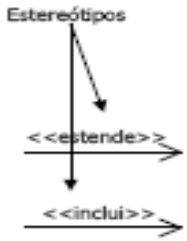
Sempre devolve um valor como resposta. Um caso de uso tem início, meio e fim. É completo, não terá terminado até que um valor tenha sido retornado;

Um caso de uso, como dito anteriormente, representa uma funcionalidade do sistema: tem início, uma entrada, uma solicitação, tem meio, um processamento, uma gravação e tem um fim, uma confirmação, uma impressão, o resultado de uma consulta na tela. Por exemplo "Selecionar exemplar" é utilizando dentro de algum outro lugar, compra de livros talvez.

O caso de uso é mostrado como uma elipse e seu nome pode vir dentro do desenho ou abaixo do mesmo. Exemplos de Caso de Uso:

- Cadastrar cliente,
- Cadastrar pedido,
- Consultar produto,
- Emitir nota fiscal,
- Fechar caixa, etc...

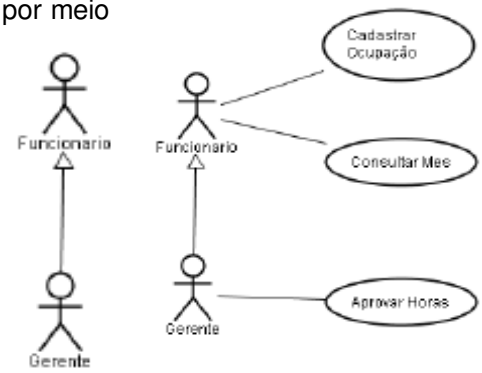
Tipos de Interações/Comunicações usadas (desde a versão 1.3):



- Extensão – mostra a exceção, os casos especiais e cursos alternativos;
- Inclusão – é a utilização de um caso de uso por outros casos de uso que possuem este comportamento em comum. É um estereótipo de dependência. Exemplo: Num caixa automático tanto o caso de uso Retirar Dinheiro quanto Fazer Transferência usam Validar Cliente – é obrigatório;
- Generalização – indica que um caso de uso é uma variação de outro, por exemplo no caso de pagamento de contas poderíamos ter uma variação de pagamento em dinheiro e outra variação pagamento em cartão.

Relacionamentos - Os diagramas de casos de uso podem ser simplificados por meio da herança entre atores.

Os diagramas de casos de uso podem ser simplificados por meio da herança entre atores. Neste caso, mostra-se um caso de uso comum aos atores específicos, que se comunicam apenas com o ator genérico.



A figura mostra as especializações de "Gerente" em "Gerente de Compras" e "Gerente de Vendas". Trata-se de uma relação de herança entre os atores, ou seja, todas as características e funções de "Gerente" serão herdadas pelos atores que estão abaixo dele.

Tanto o "Gerente de Compras" quando o "Gerente de Vendas" podem executar o caso de uso "Emissão de relatórios", porque eles são herdeiros (especializações) de "Gerente". Lembre-se que a seta aponta para o que está sendo especializado. Note que a ponta da seta é "fechada", notação própria para indicar herança.

Extensão - Essa notação pode ser usada para representar fluxos complexos opcionais ou anormais. O caso de uso "estendido" é referenciado nas precondições do caso de uso "extensor".

Extensão: O caso de uso B estende o caso de uso A quando B representa uma situação opcional ou de exceção, que normalmente não ocorre durante a execução de A. Essa notação pode ser usada para representar fluxos complexos opcionais ou anormais. O caso de uso "estendido" é referenciado nas precondições do caso de uso "extensor". As precondições são a primeira parte dos fluxos dos casos de uso.

"Extensão - 1. Ato ou efeito de estender ou estender-se. 2. Qualidade de extenso. 3. Fís. Propriedade que têm os corpos de ocupar certa porção do espaço. 4. Desenvolvimento no espaço. 5. Vastidão. 6. Grandeza, força, intensidade. 7. Porção de espaço. 8. Comprimento. 9. Superfície, área. 10. Ramal telefônico, com o mesmo número do telefone principal, usado geralmente em residências ou escritórios" (Dicionário Aurélio).

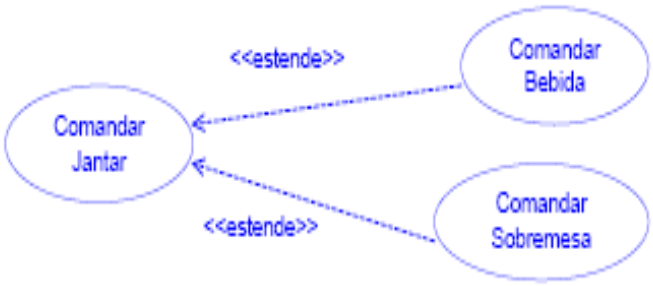
Exemplo: O desenho informa que o cadastro do cliente pode ser chamado diretamente da solicitação do serviço, mas esta é uma ação que pode ocorrer ou não. Observe que o caso de uso que estende tem uma relação de dependência com o caso de uso estendido (seta tracejada), ou seja, a o Cadastro do Cliente só pode ser executada se Solicitar Serviço for executado antes.

Inclusão - Essa notação pode ser usada para representar subfluxos complexos e comuns a vários casos de uso. O caso de uso "incluído" é referenciado no fluxo do caso de uso "que inclui".

O caso de uso A inclui o caso de uso B quando B representa uma atividade complexa, comum a vários casos de uso. Essa notação pode ser usada para representar sub-fluxos complexos e comuns a vários casos de uso. O caso de uso "incluído" é referenciado no fluxo do caso de uso "que inclui".

Exemplo: Um controle de pedidos de um restaurante, em que foram identificados os seguintes casos de uso: Comandar Jantar, Comandar Bebida, Comandar Sobremesa.

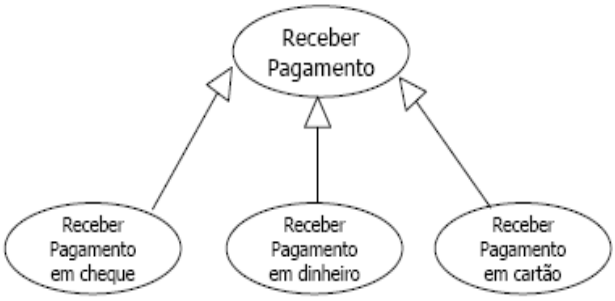
Na figura percebe-se que na funcionalidade Comandar Jantar pode-se ter um atalho para Comandar Bebida ou Comandar Sobremesa. Esta técnica é aplicada para facilitar o uso do sistema. Atenção o uso dos casos de uso estendidos são facultativos, ou seja, pode haver o pedido de jantar sem bebida e sem sobremesa.



Melhorando: Cadastrar Pagamento são passos comuns a Solicitar Serviço e Cadastrar Venda, por este motivo este conjunto de passos é isolado em outro caso de uso. Pode-se observar também que o caso de uso que inclui tem uma relação de dependência com o caso de uso incluído (seta tracejada). Ou seja, Solicitar Serviço e Cadastrar Venda só podem ser executadas se Cadastrar Pagamento for executada.

Neste caso, a solução foi pensada de forma que, quando houver o Comando de Jantar necessariamente haverá o Pagamento de Conta, da mesma forma quando houver Comando de Almoço haverá Pagamento de Conta. O importante é saber ler o que está desenhado, no exemplo o sistema somente será acionado ao final do processo, quando haverá o registro do consumo (almoço ou jantar) e logo em seguida o pagamento da conta.

Generalização entre casos de uso - Os UCs filhos podem adicionar e redefinir o comportamento do UC pai, através da inserção de seqüências adicionais de ações em pontos arbitrários da seqüência do UC pai. Este tipo de relacionamento é mais facilmente detectado na descrição do caso de uso, quando uma exceção não necessariamente é um erro ou ocorre com freqüência.



Receber pagamento em cheque, Receber pagamento em dinheiro e Receber pagamento em cartão são especializações do caso de uso Receber Pagamento.

Os UCs filhos herdam os atributos, operações e seqüências de comportamento do UC pai. Os UCs filhos podem adicionar e redefinir o comportamento do UC pai, através da inserção de sequências adicionais de ações em pontos arbitrários da sequência do UC pai. Os UCs filhos podem substituir o UC pai em qualquer lugar que ele aparece. Relacionamentos de inclusão e extensão do use case filho também modificam o use case do pai. Normalmente a similaridade entre use cases é identificada após a descrição dos casos.

8.2.1 Como fazer o Diagrama de Casos de Uso?

Para facilitar a identificação na construção do diagrama, dividimos a construção em três partes:

- A primeira é a identificação dos possíveis casos de uso – lembrar que caso de uso é uma funcionalidade do sistema e tem início, meio e fim. Exemplo:

Empresa de manutenção em equipamentos médicos

Nº	Evento	Use Case	Ator	Resposta
1	Vendedor fecha novo contrato	Registrar contrato	Vendedor	Contrato registrado
2	Cliente solicita serviço	Registrar chamada	Cliente	Chamada registrada
3	Supervisor solicita chamadas pendentes	Gerar chamadas pendentes	Supervisor	Chamadas pendentes geradas
4	Supervisor aloca técnico	Alocar técnico	Supervisor	Técnico alocado
5	Técnico informa o fechamento da chamada	Fechar chamada	Supervisor	Chamada encerrada
6	Vendedor cadastra novo cliente	Cadastrar cliente	Vendedor	Cliente cadastrado
7	Supervisor informa possível problema	Cadastrar problema	Supervisor	Problema cadastrado

...

Use eventos para facilitar a identificação

- A segunda tarefa é a descrição dos casos de uso, neste momento, pode-se identificar os verdadeiros casos de uso do sistema, casos de uso com um, dois ou três passos podem sair do modelo e seus passos podem ser identificados como sendo de outro caso de uso. Pode ser identificada também a necessidade de criar um caso de uso novo quando identificado um conjunto de passos iguais em mais de um caso de uso. Exemplo:

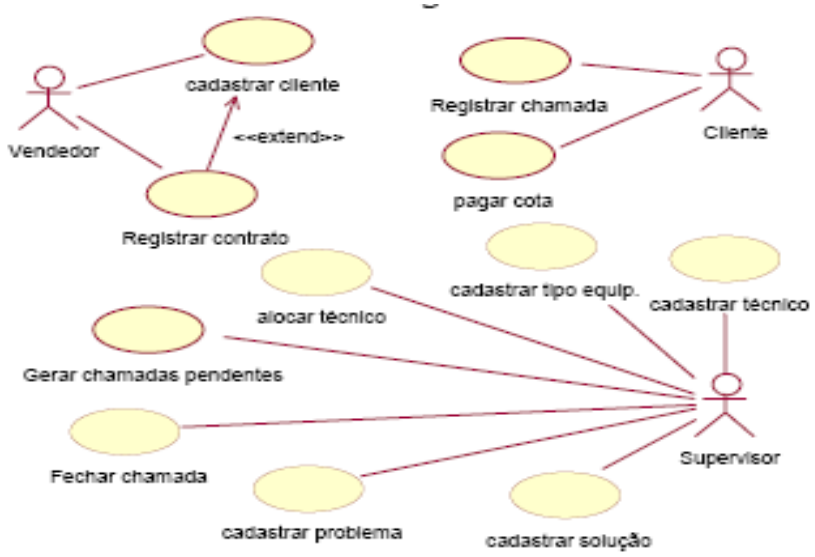
Nome : **Registrar contrato**
Ator : Vendedor
descrição : Este use case é responsável pelo cadastramento de novos contratos no sistema

Curso Normal
1-o sistema apresenta os clientes cadastrados;
São listados os nomes e o CGC dos clientes.
2- o vendedor seleciona um cliente;
3- vendedor informa dados básicos do contrato;
O vendedor deve informar a data de início, data de término e a quantidade de cotas de pagamento.
4- o sistema apresenta os tipos de equipamentos cadastrados;
5- o vendedor informa os equipamentos que farão parte do contrato;
Para cada equipamento, deve-se selecionar um tipo de equipamento e informar seu número de série e data de fabricação.
6- o sistema registra o contrato;
O sistema registra data de início e data de término, e gera um número seqüencial único independente do cliente.
7- o sistema registra os equipamentos do contrato;
cada equipamento registrado no contrato recebe um número de manutenção gerado pelo sistema
8- o sistema registra as cotas de pagamento do contrato;
com base nas informações fornecidas pelo vendedor referentes a quantidade de cotas, data base de pagamento e pesquisando o valor de manutenção de cada tipo de equipamento o sistema registra as cotas de pagamento do contrato

Curso Alternativo
Passo 2
Caso o cliente desejado não esteja cadastrado
1. o sistema gera um aviso de cliente não cadastrado;
2. ESTENDER Cadastrar Cliente;
3. retornar ao passo 3

Todas as exceções devem ser previstas aqui

- A terceira parte é o desenho do diagrama, após a identificação e a descrição dos casos de uso o desenho do diagrama fica fácil. Exemplo:



Observações sobre desenvolvimento de casos de Uso:

- Evitar na descrição, o caso de uso sem fim, aquele que está sempre retornando a algum passo (loop);
- Procurar sempre iniciar um caso de uso com uma ação do Ator;

- Procurar organizar os casos de uso de modo que seja melhor aproveitado o polimorfismo na hora da implementação, agrupando-os a nível macro, ou seja, os casos de uso que envolvem uma mesma idéia de negócio devem ser abstraídos em um único caso de uso;
- Um erro comum é achar que um caso de uso é uma atividade simples, um caso de uso é definido na sua descrição – é um conjunto de passos;
- Evitar palavras estrangeiras, lembrando-se que o Diagrama de Caso de Uso serve para validação com o usuário;
- Não desenhar associação entre atores, somente é permitido a generalização;
- O Diagrama de Casos de Uso é uma excelente ferramenta para levantamento de requisitos, portanto cuidado com os falsos requisitos:

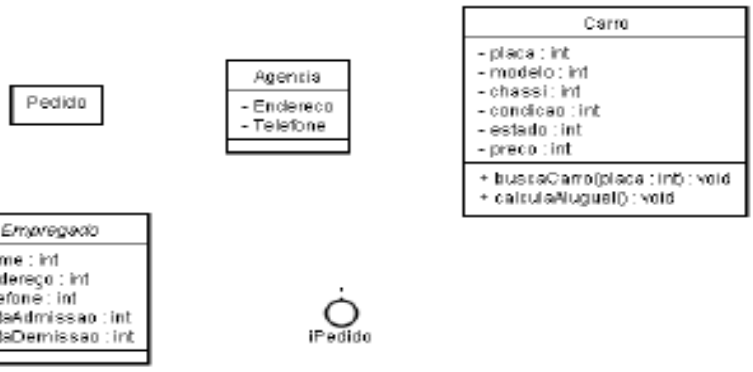
  a) Requisito falso tecnológico: na modelagem de sistemas há o termo tecnologia perfeita, deve-se abstrair todos os problemas de tecnologia.

  b) Requisito falso arbitrário: funcionalidade que o sistema não precisa possuir para atender ao seu propósito.

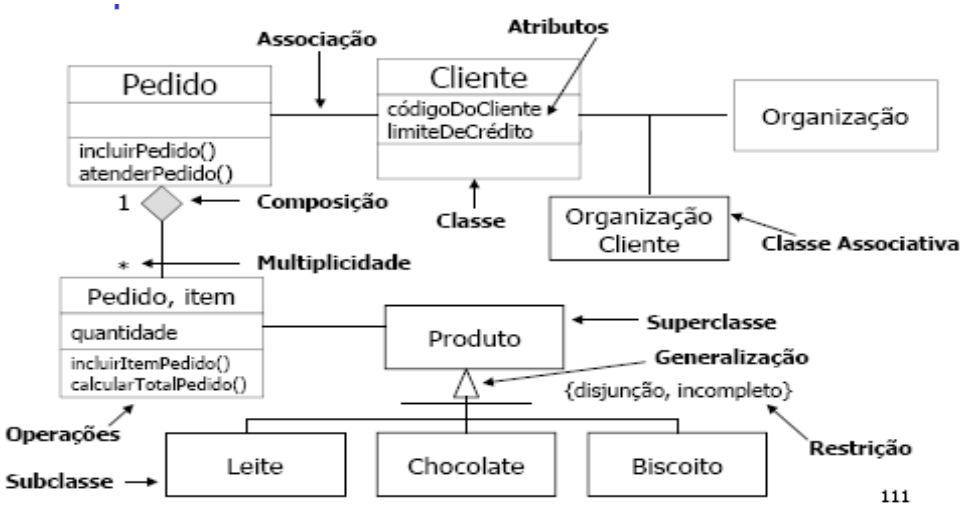
8.3 Diagrama de Classe

É a essência da UML, trata-se de uma estrutura lógica estática em uma superfície de duas dimensões mostrando uma coleção de elementos declarativos de modelo, como classes, tipos e seus respectivos conteúdos e relações. É composto de:

- Classes (atributos e métodos);
- Associações.



Exemplo:

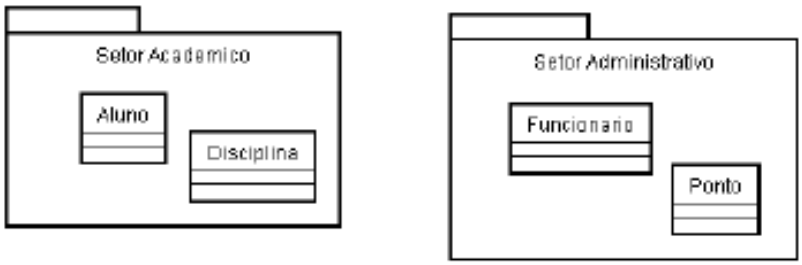


8.3.1 Pacotes

Os pacotes lógicos são agrupamentos de elementos de um modelo. Um sistema pode ser dividido em pacotes para melhorar o entendimento e para aumentar a produtividade.

Os pacotes lógicos são agrupamentos de elementos de um modelo. No modelo de análise, eles podem ser utilizados para formar grupos de classes com um tema comum, pode ser útil para a divisão do trabalho na equipe de desenvolvimento.

Os pacotes da figura são usados para agrupar classes especializadas em relação a esses aspectos. Pacotes lógicos podem ter relações de dependência e pertinência entre si.

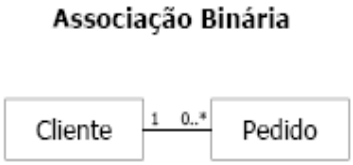


8.3.2 Associação

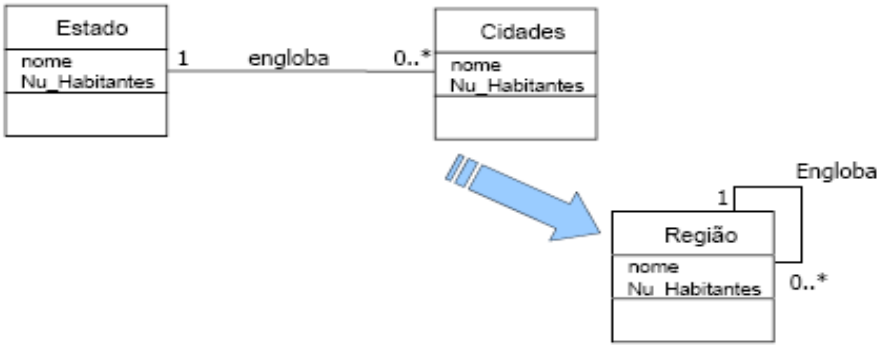
Associação é o relacionamento entre as classes, estabelece um vínculo entre objetos. Tipos de associação:

- Associação simples (binária, unária e n-ária);
- Agregação/Composição: Generalização

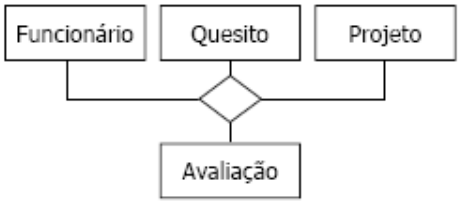
a) A associação **binária** é a mais comum nos diagramas de classe.



b) Associação **Unária ou Autorelacionamento** - Ocorre quando há necessidade de relacionar dois objetos de uma mesma classe.



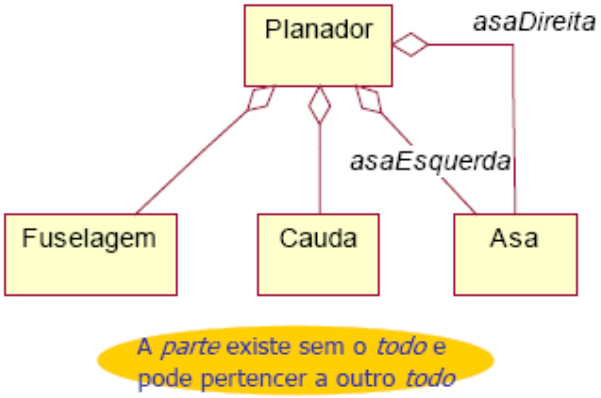
c) Associação **n-ária** - Representa o relacionamento com mais de duas classes.



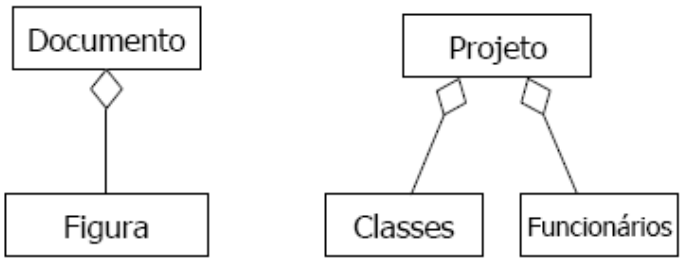
O exemplo acima está exibindo o relacionamento de avaliação de um funcionário em um projeto, e esta avaliação é para um quesito. Então, por exemplo, o funcionário João será avaliado em pontualidade no projeto A e terá uma nota de avaliação, quando ele for trabalhar no Projeto B ele poderá ter outra nota de avaliação no mesmo quesito. Os quesitos servem para pontuar os funcionários e são exemplos: organização, limpeza, pontualidade, etc.

8.3.3 Agregação

Usada para denotar relacionamentos todo/parte, por exemplo: um avião tem cauda, fuselagem e asas como partes. Neste exemplo a cauda a fuselagem e as asas do avião poderão ser utilizadas em outro avião. E se o planador for excluído talvez as partes continuem existindo para um dia compor outro planador.

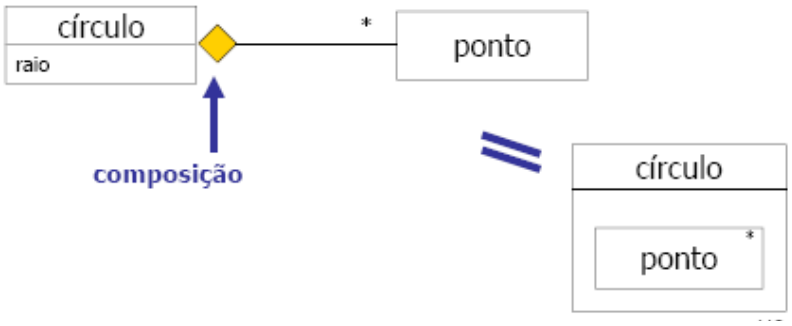


As figuras mostram dois exemplos de agregação: um documento pode ter figuras e parágrafos. Estes serão membros do documento (todo-parte). Da mesma forma um projeto pode trabalhar com classes e funcionários, que poderão ser utilizados por outro projeto, significa que ao chegar ao fim de um projeto, as classes e os funcionários podem ser reaproveitados.



8.3.4 Composição

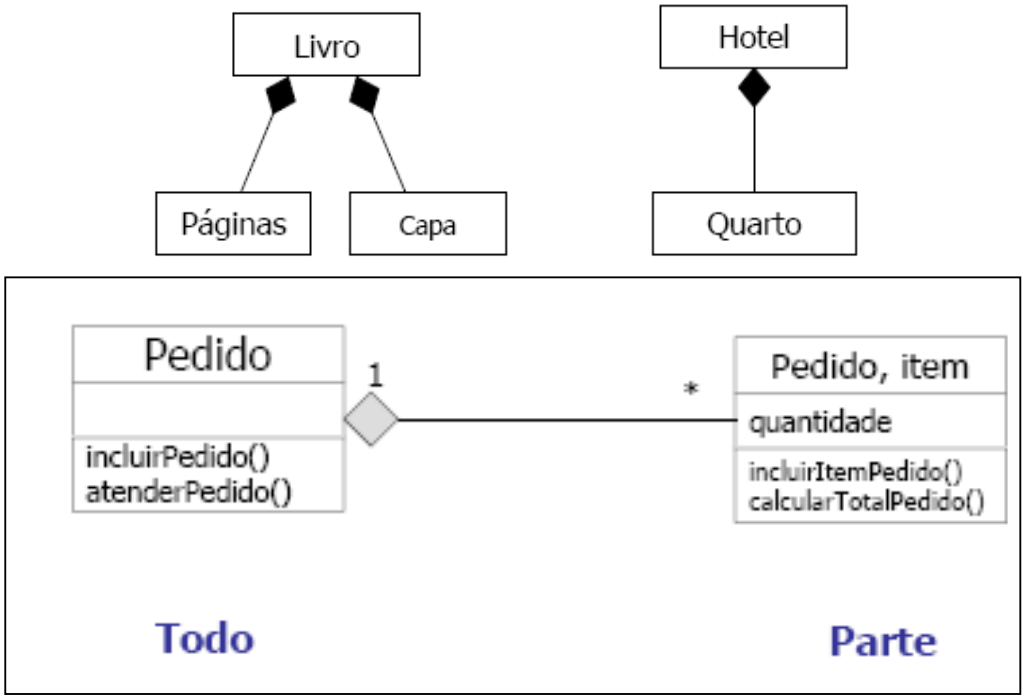
É uma variação da agregação, mas sendo que o objeto parte só pertence a um todo, e são extremamente dependentes do todo, podemos considerar que qualquer deleção do todo gera um efeito cascata nas partes.



Composição - Um tipo mais forte de relacionamento todo-parte é o relacionamento de

composição. Nesse caso, os objetos da classe parte não têm existência independente da classe todo. É uma especialização da Agregação.

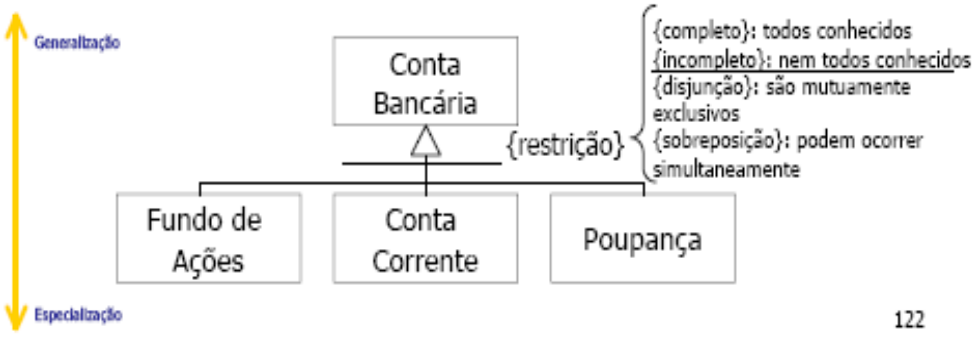
A composição impõe a cardinalidade de no mínimo 1. Os exemplos sugerem que as páginas e a capa do livro compõem um livro e não podem compor outro livro. Assim como os quartos do hotel somente pertencerão a ele, não faz sentido retirar o quarto do cadastro e aproveitar para outro hotel (a parte não existe sem o todo). Exemplos:



8.3.5 Associações

Existem quatro tipos principais de associações:

a) Generalização/Especialização – Relacionamento entre um elemento mais geral e um mais específico, também conhecido como herança ou classificação. O elemento mais específico pode conter somente informação adicional acerca do elemento mais geral.

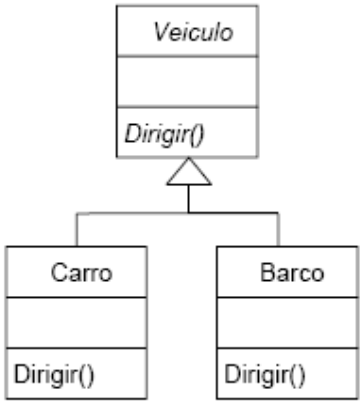


O uso das restrições é facultativo e serve para identificar o que está desenhado nas especializações da super-classe. Pode ser que estejam todos os sub-tipos desenhados no diagrama, se fosse colocado no exemplo acima poderíamos dizer que todas as contas são do tipo Fundo, Corrente ou Poupança, neste caso a restrição seria 'completo'. Caso tenha algum tipo de conta não mostrado no desenho, como Conta Aplicação teríamos a restrição sendo

'incompleto'. Para as restrições de sobreposição temos que uma conta bancária pode se comportar como poupança e como conta corrente, o dinheiro não fica parado nunca está no mínimo rendendo os juros da poupança. Se o sistema não funcionar desta forma, ou seja, uma conta bancária é uma conta poupança ou corrente ou fundo de ações temos a restrição de 'disjunção'.

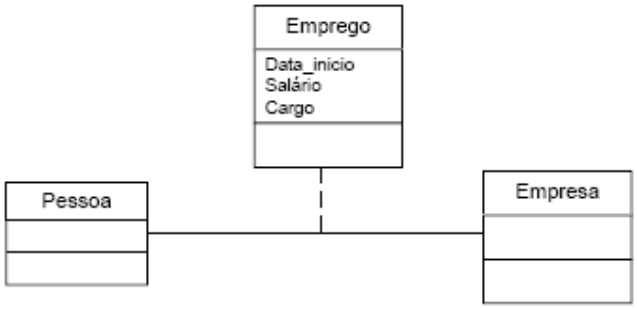
b) Classes Abstratas:

- É uma classe que define o comportamento e atributos para subclasses.
- Não é instanciada diretamente.
- Uma classe abstrata pode conter operações abstratas. São operações cuja implementação não é especificado na superclasse, somente sua assinatura.
- As classes que herdarem essa operação deverão implementá-la, sendo a implementação diferente para cada classe, ou mantê-la abstrata.



c) Classe associativa - Ocorre quando há necessidade de colocar informação em uma associação.

Os dados de emprego só existem se houver uma pessoa e uma empresa, se não houver esta associação não existirá emprego.

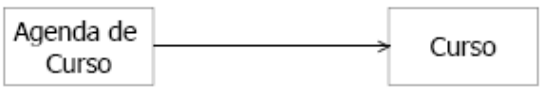


Dependência entre Classes:

- Relacionamento de dependência é uma conexão semântica entre dois elementos do modelo.
- Qualquer alteração no elemento independente (Curso) poderá afetar o elemento dependente (Agenda de Curso).
- Como identificar dependências?

  i) Quando uma classe tem uma operação que usa uma instância de outra classe como parâmetro;

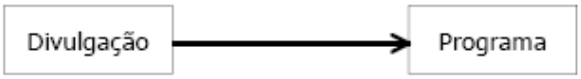
  ii) Uma classe chama uma operação que é de escopo de outra classe.



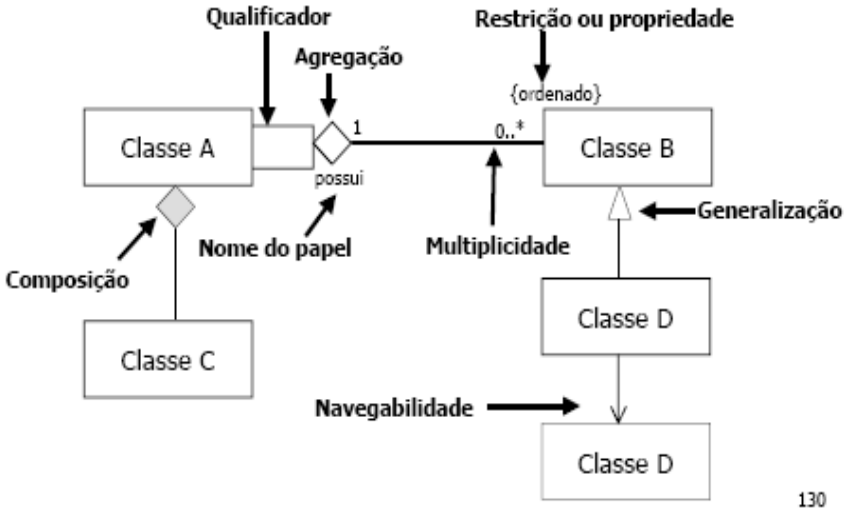
8.3.6 Navegabilidade

Indica a referência de objetos de um classe a objetos de outra classe. Como no exemplo, indica-se que uma divulgação feita tem a responsabilidade de informar a qual programa pertence.

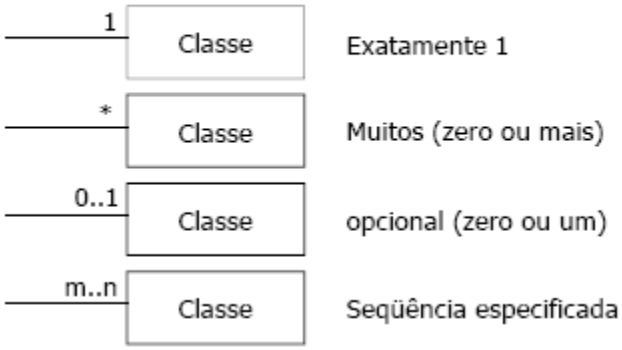
Como no diagrama de classe não tem a representação de chave estrangeira, a navegabilidade indica que a divulgação terá a chave estrangeira da entidade programa, pois divulgação terá a responsabilidade de saber qual programa faz referência.



Papeis em associação:

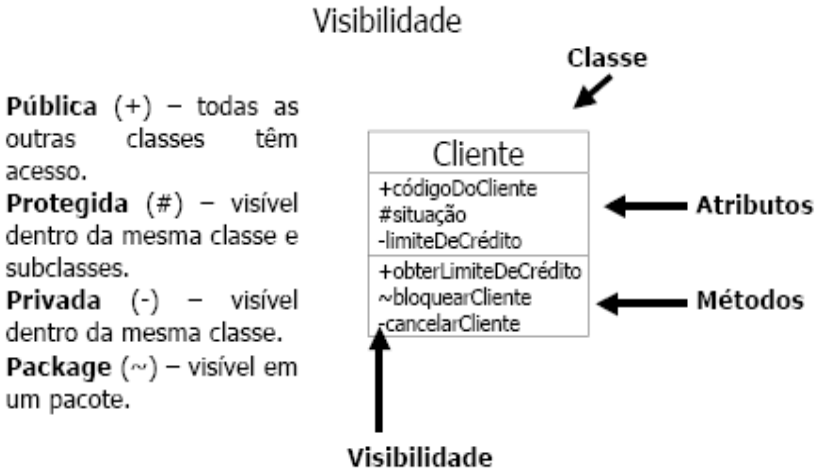


Multiplicidade



8.3.7 Visibilidade

Uma Classe pode definir o tipo de acesso que seus membros (propriedades e métodos) permitirão às demais partes do sistema. Em uma escala progressiva de “privacidade” dos membros, os tipos de acesso possíveis são: público, protegido e privado.



Para facilitar a criação do diagrama de classes, deve-se seguir os passos:

1º Passo: Identificar as Informações do Sistema (classes e atributos);

2º Passo: Desenhar o diagrama. Agora o que foi identificado anteriormente entrará no desenho: como uma classe, ou atributo, ou método, ou associação – ou não entra no modelo

3º Passo: Relacionar todas as classes levantadas. Os objetos de uma determinada classe fazem referência a quais outros objetos?

4º Passo: detalhar atributos e métodos.

Atributos – Leia atentamente os casos de uso e identifique as necessidades de informação;

Métodos – Inicialmente podemos ter idéia dos métodos, mas o diagrama de seqüência irá ajudar bastante a levantar os métodos.

5º Passo: fazer batimento com as descrições dos casos de uso para verificar se há divergências.

Pode haver necessidade de criação de novos casos de uso. Podem ser descobertas novas classes/atributos. Observações:

- Faça o diagrama pensando em conjuntos. O conjunto de clientes, serviços e o relacionamento entre eles, aliás, o que é classe senão um conjunto de objetos?

- O diagrama de classes pode ser feito com perspectivas diferentes, podemos fazer uma versão visando a parte conceitual (entendimento das classes e suas associações), outra versão para especificação, e outra para implementação (com: visibilidade, todos os atributos e métodos, navegabilidade, restrições...).
- Cuidado com associações que tenham multiplicidades mínimas 1, em perspectiva de implementação. Como no exemplo, o cliente tem no mínimo um serviço e o serviço tem no mínimo um cliente associado.

Dúvidas que podem aparecer:

- Os atores identificados viram classes ? Cuidado nem todo ator é uma classe! O ator é quem executa a ação, se houver a necessidade de armazenar quem executou a ação aí sim teremos o ator como classe no diagrama de classe.

- Identifiquei as classes, mas os métodos em qual classe coloco? Os métodos trabalham com as informações da classe onde ele irá ficar, lembre-se do encapsulamento.
- Identifiquei alguns métodos genéricos que trabalham informações de várias classes, como fazer? Existem as classes de projetos (ou Utilitários) que facilitam o agrupamento de métodos genéricos, por exemplo emitirNotaFiscal() é um método que trabalha com informação da filial, do produto, do pedido e do cliente.

8.4 Diagrama de Seqüência

Interação é uma especificação comportamental que inclui uma seqüência de trocas de mensagem entre um conjunto de objetos dentro de um contexto. Existem dois diagramas de Interação:

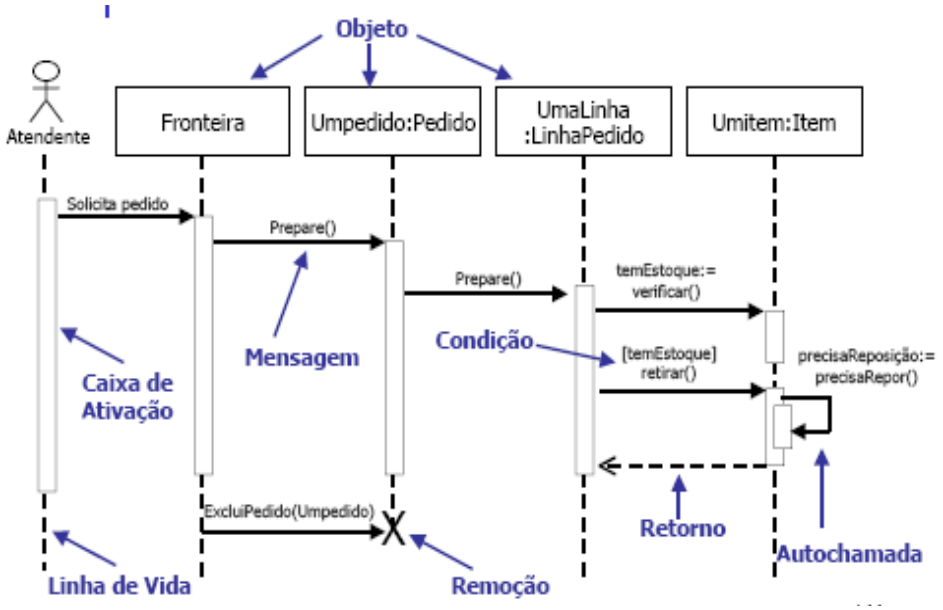
- Diagrama de Seqüência – interação entre objetos ao longo do tempo, mostrando todos os objetos que participam do contexto e a seqüência de mensagens trocadas.
- Diagrama de Colaboração – representa a troca de mensagens entre um conjunto de objetos.

8.4.1 O Que é o Diagrama de Seqüência?

Os **diagramas de seqüência** são orientados para exprimir o desenrolar temporal de seqüências de ações. É difícil representar lógicas de seleção e repetição sem prejudicar a legibilidade do diagrama. Os roteiros representam desdobramentos da lógica do caso de uso. É preferível usar diagramas separados para representar roteiros resultantes de diferentes caminhos lógicos.

Os **diagramas de colaboração** são elaborados baseados na descrição de um fluxo do caso de uso juntamente com o diagrama de classes, visto que os diagramas de interação representam as mensagens entre os objetos, que são os passos levantados na descrição e os objetos são as classes identificadas no diagrama de classes.

Componentes do Diagrama de Seqüência: Objetos, Mensagem (retorno, auto-chamada, condição), Linha de Vida, Caixa de Ativação, Remoção.



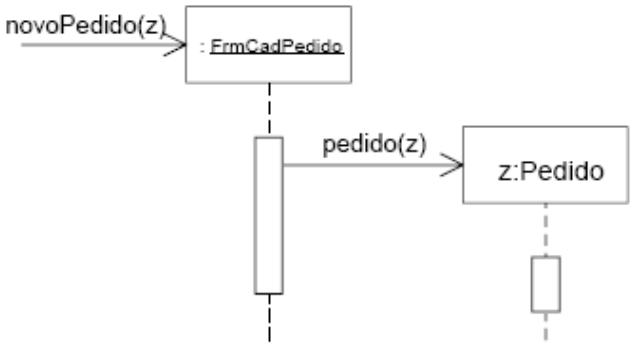
- Mensagem – pode ser um evento ou qualquer outra informação necessária, mas sempre com expectativa de ação a ser executada. Pode incluir parâmetros contendo algum dado adicional e a resposta do objeto destinatário depende do método associado à mensagem. Pode ser também etiquetada com uma expressão booleana (veja exemplo).
- Linha de vida do objeto – representa a existência do objeto.
- Auto-chamada ou auto-delegação – quando uma operação chama a si própria.
- Caixa de ativação – é o tempo em que o objeto fica ativo em memória.

Restrição – a restrição é utilizada sempre quando não é possível demonstrar algum requisito no desenho do diagrama. Podemos utilizar no Diagrama de Seqüência a restrição para indicar uma restrição de tempo de retorno de uma resposta, o que é bastante válido para um sistema de tempo real. Sistemas de Tempo Real são caracterizados por terem que atender, não apenas a correta execução lógica das tarefas, como também a limites de tempo de execução. Para que uma tarefa em tempo real seja considerada bem sucedida, ela deve ser concluída com sucesso e dentro de um tempo pré-determinado, como representar este requisito na UML? Através do Diagrama de Seqüência colocando restrições de tempo nas execução das atividades.

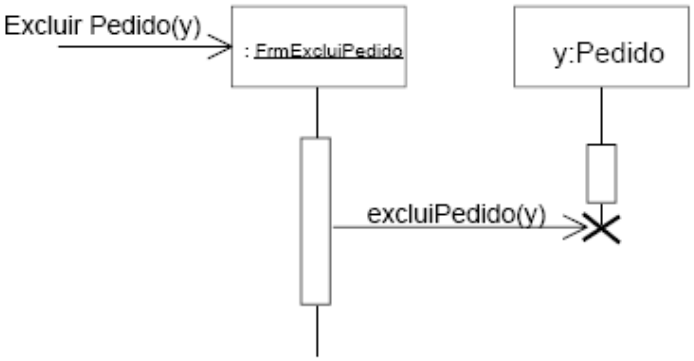
Criação e exclusão de Objetos:

- Criação e exclusão de objetos também são representados em um diagrama de seqüência;
- Quando uma mensagem de criação é enviada (geralmente síncrona), o símbolo do objeto é mostrado no local onde foi criado;
- Quando for destruído, é marcado com um X.

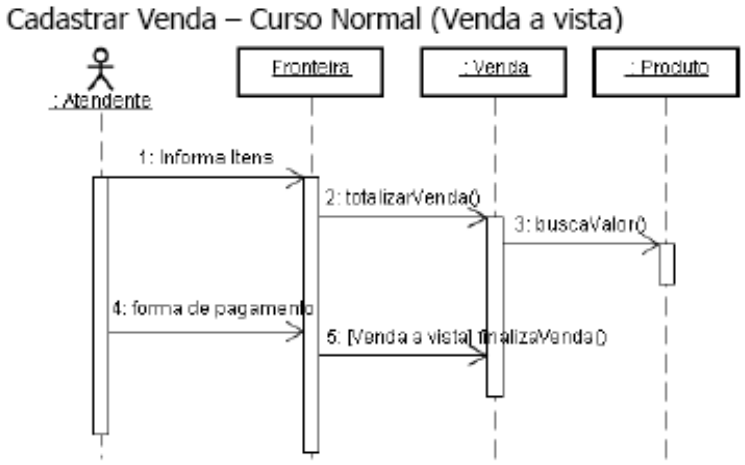
Exemplo: No diagrama pode-se observar que a instância de Cliente é criada a partir da Janela de Atendimento. Pode-se notar que a instância de Cliente aparece um pouco abaixo da instância da janela, indicando que ela foi criada depois.



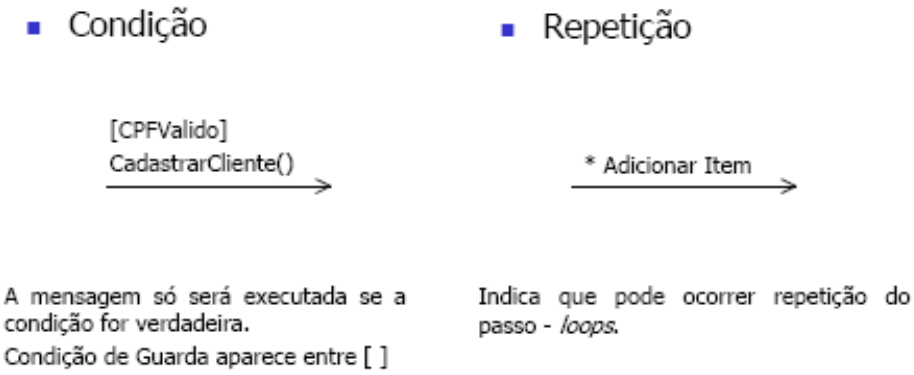
A figura mostra um 'X' na linha de vida da instância de cliente indicando a sua exclusão. Note que a linha de vida do objeto não segue adiante.



Exemplo de diagrama de seqüência:



A **condição** no diagrama de seqüência é representada através de uma expressão entre colchetes (Condição de Guarda), e indica que a mensagem só será executada se a condição for verdadeira. A repetição é vista com o uso do asterisco e indica que o passo poderá ser repetido indefinidamente.



Inclusão e Extensão no Caso de Uso: Cada cenário do caso de uso tem um diagrama de seqüência, teremos um diagrama de seqüência para o curso principal do caso de uso estendido ou incluído e no diagrama de seqüência do que inclui colocamos uma nota com a observação. Observações para o desenvolvimento do diagrama:

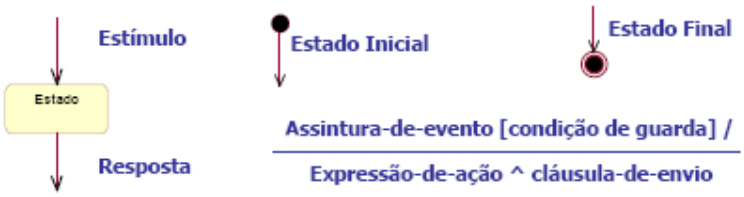
- É permitido o uso de Notas a qualquer momento, mas cuidado para não poluir o desenho.
- Existem autores que utilizam a fronteira como objeto de interface entre o sistema e o ator, representando as telas do sistema, outros autores preferem instanciar cada fronteira com o nome das telas, outros autores não utilizam fronteira de forma nenhuma. Ache sua forma de desenhar.
- O Diagrama de Seqüência é um descobridor de métodos e um validador da seqüência de passos da descrição do caso de uso.
- Se houver necessidade de colocar algum método na fronteira podemos inserir uma (ou n) classe de fronteira (atualize o diagrama de classes) contendo os métodos genéricos e parâmetros do sistema.

8.5 Diagrama de Estado

O que é o Diagrama de Estados? Representa os estados que um objeto pode possuir, e a ordem pela qual ele passa por estes estados, desde sua criação até sua destruição. Estuda o ciclo de vida de uma classe, um caso de uso, um pacote ou uma operação, mas é mais comumente utilizado para classe, ou melhor, para os objetos de uma classe. Serve para estudar certos tipos de lógicas que envolvem transições possíveis entre diferentes estados de um sistema.

O diagrama de estados visa o estudo do comportamento do objeto. Como os diagramas na UML são complementares, o diagrama de classe deve ter uma propriedade para a informação do estado do objeto criado e o diagrama de caso de uso deve prever a alteração do estado deste objeto nas várias etapas do seu ciclo de vida. O diagrama de estados observa o comportamento de uma instância.

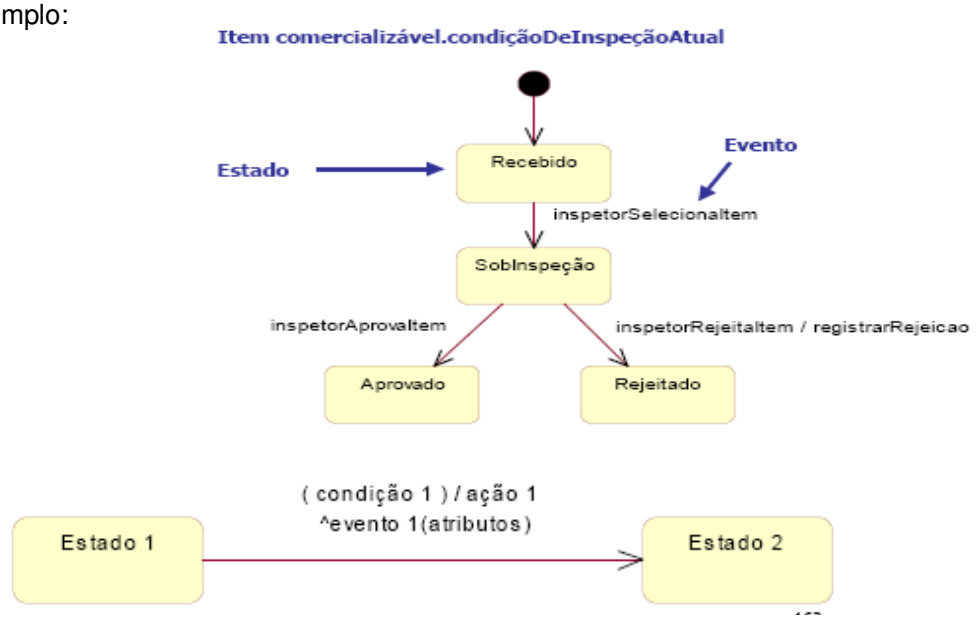
Em sistemas complexos o trabalho é muito extenso e difícil, a UML indica o uso do diagrama de estado por classe, mostrando os estados que os objetos dessa classe podem assumir e as transições que eles podem fazer entre estados, sendo ideal para descrever o comportamento de um único objeto. O sistema em um determinado estado recebe estímulos (entradas) os trata para exibir respostas (saídas).



Um Diagrama de Estado relaciona eventos e estados. Quando um evento é recebido, o estado subseqüente depende do estado corrente e do evento. A modificação de estado causada por um evento é chamada transição.

Um diagrama de estados é um grafo cujos nós são estados e cujos arcos direcionados são transições rotuladas com nomes de eventos. Deve ser construído um Diagrama de Estados para cada classe cujos objetos apresentem algum comportamento dinâmico significante.

Exemplo:



Conceitos usados na Transição:

- Evento – é a especificação de uma ocorrência, o que irá ocasionar a alteração do estado de um objeto.
- Ação – atividade atômica que não pode ser interrompida.
- Condição de Guarda – Somente irá mudar para o estado se a condição for verdadeira. Aparece entre colchetes '[ ]' na transição.

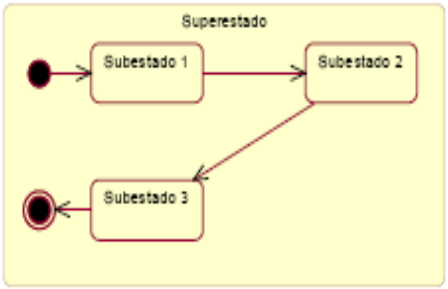
Na transição (a seta entre estados) fica com o nome do evento ou o que foi responsável na mudança de estado do objeto. Na alteração do estado de um objeto pode ser necessário executar alguns métodos, isso é representado através da ação, que também poderá aparecer na transição ou dentro do estado. A condição de guarda também será exibida na transição.

Exemplos dos Componentes:

- Estado: Pronto, em Manutenção, Checando, Finalizando, Efetuando Transação, em Consulta, em Internação.
- Evento: Cadastrar Empregado, Demitir Funcionário.
- Ação: identificarCartao(), validarSenha().
- Condição de Guarda: Senha inválida, falha detectada.
- Observe que o evento é o que faz com que o objeto mude de estado, seu conceito está intimamente ligado ao da transição.

8.5.1 Máquina de Estados:

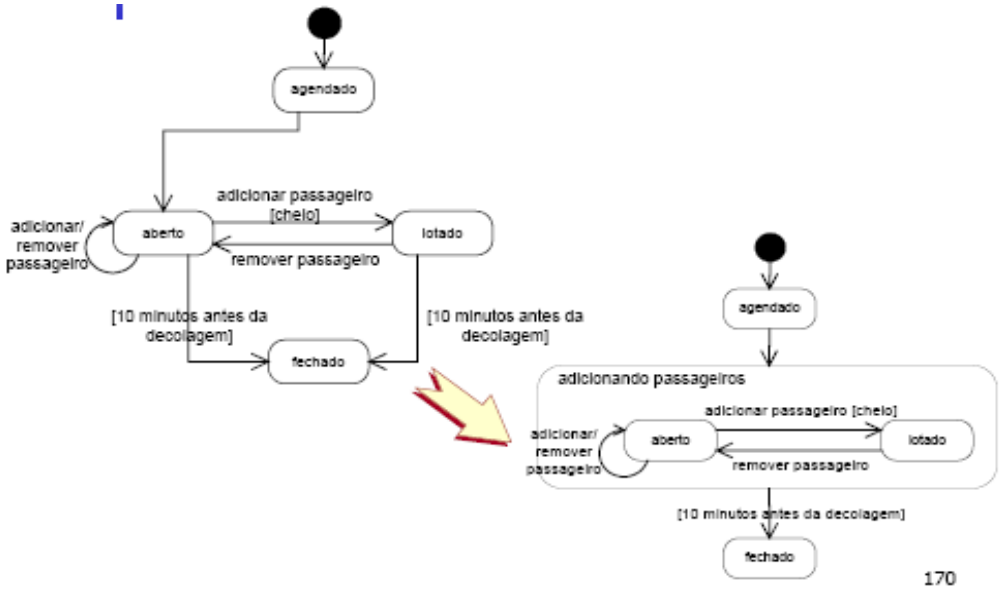
Os superestados são utilizados para minimizar a desorganização do diagrama de estados. Os superestados são compostos de dois ou mais subestados de uma mesma transição.



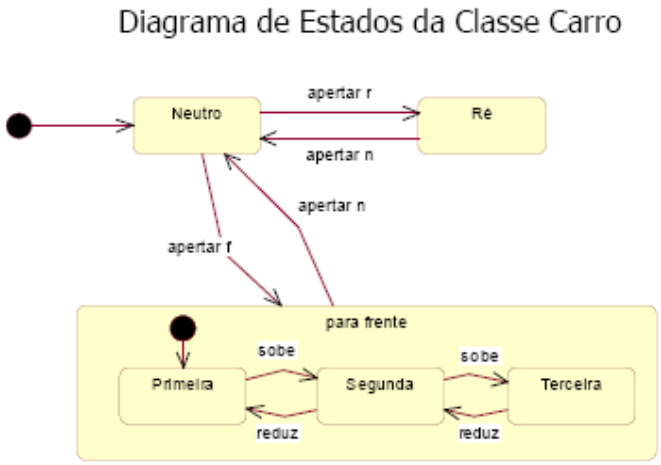
A máquina de estados (ou Superestado ou Estado Composto) foi criada para agrupar estados e facilitar a leitura do diagrama.

Máquinas de estado aninhadas servem para subdividir estados complexos em estruturas mais simples. Subestado é um estado que é parte de um estado composto.

Nas figuras exibidas abaixo, percebe-se a facilidade da leitura permitida pelo superestado. Perceba que tanto o voo aberto quanto lotado passam para a situação fechado, independente da situação anterior o que faz com que o voo passe para fechado são os 10 minutos antes da decolagem.



Exemplo:



Para definição dos estados, o que deve ser feito é:

- Pensar nas etapas de vida, ou nos estados que o objeto pode passar.
- Levantar todos os eventos, ou as ações ou funcionalidades do sistema que alteram o objeto;

Passos para o desenvolvimento:

1º passo: Identifique os estados da classe:

- Em definição (ou 'definindo');
- Entrevistando;
- Finalizada.

Voltar ao enunciado e descubrir os estado do objeto (Pesquisa).

2º passo: Identifique os eventos (ou ações) do sistema que alteram o estado do objeto:

- Cadastrar pesquisa;

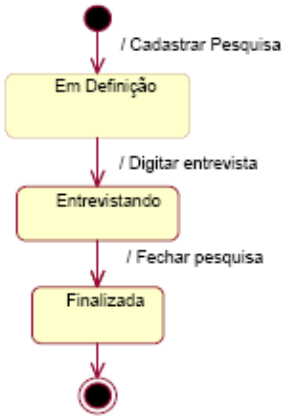
- Digitar entrevista;
- Fechar pesquisa.

Na identificação dos eventos tem-se que buscar as ações que são executadas no sistema, para a classe em questão, ou seja, tem-se que identificar quais são as ações no sistema que alteram o estado da 'Pesquisa'.

3º passo: Desenhe o Diagrama:

Depois de empregar os passos anteriores o desenho do diagrama se torna muito fácil.

Lembre-se que o diagrama pode ser enriquecido com outras características do sistema como a ação, condições de guarda, agrupamento de estados e o compartimento de transições internas.



Quando utilizar este diagrama?

Nem todas as classes podem ser representadas por um diagrama de estados, simplesmente por não possuir estados a serem analisados.

Se a classe tem um atributo status (ou situação) é um indicativo de que o objeto terá comportamentos diferentes de acordo com os valores atribuídos neste atributo.

Esta indicação pode ser vista por associação com minimalidade 0, indica que o estado da classe pode ser diferente se tiver esta associação.

O uso do Diagrama de Estados é facultativo, ele só será utilizado quando houver necessidade de representar os estados de uma classe, é o estudo do ciclo de vida que um objeto poderá trilhar.

Em um contexto geral percebe-se que o diagrama de estados depende do diagrama de classes e do diagrama de casos de uso e pode-se ter mais de um diagrama de estados por sistema. Lembrar mais uma vez que os diagramas se completam e conforme vão sendo identificados os eventos, estados devem ser revistos e o diagrama de classes e casos de uso. Os diagramas devem estar coerentes.

8.5.2 Para terminar

- Busque a padronização nos nomes – Atributos, Métodos e Classes (co_cliente, id_peça, num_func, nu_endereço,...);
- Todos os elementos de modelagem (Caso de Uso, Classe, Objeto, etc…) devem estar no singular;
- Faça sempre dicionários de dados ou negócios, não confie na sua memória;
- Atenção ao Negócio – Não desvie de foco, nem inclua funcionalidades extras se o cliente não solicitou.

9 GERENCIAMENTO DE PROJETOS

O fracasso de muitos projetos de softwares grandes nas décadas de 60 e 70 foi uma indicações das dificuldades de gerenciamento de software que as empresas enfrentavam. Muitos destes projetos fracassaram porque a abordagem gerencial estava errada, as necessidades de gerenciamento de projetos de softwares são diferentes das empregadas em empresas de manufatura ou de fabricas.

O trabalho do gerente de software é garantir que o projeto cumpra as restrições impostas (orçamentárias, de prazos, de requisitos, etc...) e entregar um produto de software que contribua para as metas da empresa. Eles são os responsáveis por planejar e programar o desenvolvimento do projeto; supervisionam o trabalho para assegurar que ele seja realizado em conformidade com os padrões requeridos e monitoram o progresso para verificar se está dentro do prazo e do orçamento aprovado. O bom gerenciamento não pode garantir o sucesso do projeto, mas o mau gerenciamento geralmente resulta no fracasso do projeto.

A engenharia de software é distinta de outros tipos de engenharia, tornando o gerenciamento de software dificil. As principais diferenças são:

- O produto é intangível;
- Não há processo de software padrão;
- Grandes projetos de software são frequentemente únicos.

9.1 O que é Gerenciamento de Projetos?

Segundo Carneiro (2000), o Gerenciamento de projetos é a aplicação de **Conhecimento, Habilidades, Ferramentas** e **Técnicas** nas atividades de projetos de forma a atender ou superar as expectativas dos stakeholders (interessados, atores, participantes) e que envolve o balanceamento de":

- Escopo, tempo, custo e qualidade;
- Necessidades (requisitos definidos) e expectativas (subjetivos ou não definidos);
- Diferentes expectativas e necessidades de todos aqueles que participam do projeto direta ou indiretamente.

Indica que para fazer o gerenciamento de projetos não é necessário apenas a vontade ou a necessidade da realização dessa tarefa. É preciso reconhecer que o gerente de projetos precisa de conhecimentos específicos da área para melhor desempenhar as suas funções.

Existe uma tendência das empresas em administrar as operações com a abordagem de projeto. Essa abordagem, de forma simplificada, prevê a aplicação das técnicas, habilidades, ferramentas e conhecimento na condução de operações da empresa. O termo usado para essa tendência ou filosofia é o gerenciamento por projetos, que visa alinhar os grandes objetivos estratégicos da empresa com inúmeros projetos, coordenados e gerenciados, de forma a garantir a sua execução no menor tempo, na melhor qualidade e no melhor custo.

9.2 Atividades de Gerenciamento

O trabalho de um gerente de software varia muito, dependendo da organização e do produto a ser desenvolvido. Na maioria das vezes, ele assume algumas ou todas as seguintes atividades:

- Elaboração de propostas;
- Planejamento e programação de projetos;
- Levantamento dos custos do projeto;
- Monitoramento e revisões de projetos;
- Seleção e avaliação de pessoal;
- Elaboração de relatórios e apresentações.

9.3 As Áreas de Conhecimento em Gestão de Projetos na Visão do PMI

O PMI (Project Management Institute) identifica 9 (nove) áreas de conhecimento em gerenciamento de projetos. Apesar desta abordagem de apresentá-las de forma separada, por razões didáticas, deve-se estudá-las com a percepção de todas estão intimamente interligadas. O gerenciamento de um projeto sem a aplicação do conhecimento de uma ou mais destas áreas poderá implicar em uma deficiência do próprio projeto, que, normalmente só é constatada tardiamente. Depois ter despendido muito esforço, custo e tempo para encontrar as razões desta deficiência.

Como explica Carneiro (2000), durante algum tempo o principal enfoque do gerenciamento de projetos era a gerência do tempo: fazer com que as coisas acontecessem dentro do prazo esperado. Em algumas empresas, em especial no governo, o enfoque de gerenciamento de projeto era mais orçamentário, ou seja, quando acabasse o dinheiro, acabaria o projeto.

Outros elementos foram se juntando ao gerenciamento do projeto, além do prazo e custos, tais como o escopo do projeto (o que deveria ser feito), foram identificados como importante para a gestão de um projeto tratar a questão de qualidade, ou seja, atender ao especificado e tratado entre as partes.

Outros aspectos foram sendo agregando ao gerenciamento de projeto, tais como risco e comunicação. Todo projeto tem risco e o planejamento e tratamento desses riscos faz parte das atividades de gerenciamento do projeto. Comunicação também é fator fundamental em um projeto. Sem a devida comunicação entre os envolvidos um projeto pode estar fadado ao insucesso. As informações devem fluir no tempo, na profundidade e no conteúdo desejado por cada envolvido, de acordo com a sua necessidade ou grau de envolvimento.

Não deve-se esquecer que os projetos são executados por pessoas. Então o cuidado com o ser humano, tais como: a motivação da equipe, o recrutamento de pessoas especializadas para cada tipo de tarefa, o treinamento, a formação de time e outros aspectos são também fundamentais ao sucesso do projeto.

Em alguns projetos têm que adquirir bens ou produtos e o gerenciamento dessas aquisições também são importantes para a condução de um projeto.

Percebe-se então que o gerenciamento de projetos envolve o tratamento de vários aspectos importantes. Esses aspectos são chamados na metodologia PMI de disciplinas. S disciplinas são:

- Gerenciamento de Integração entre os elementos do projeto;
- Gerenciamento de Escopo de Projeto;
- Gerenciamento de Tempo do projeto;
- Gerenciamento do Custo;
- Gerenciamento da Qualidade;

- Gerenciamento de Recursos Humanos;
- Gerenciamento de Comunicação;
- Gerenciamento de Risco;
- Gerenciamento de Contratos.

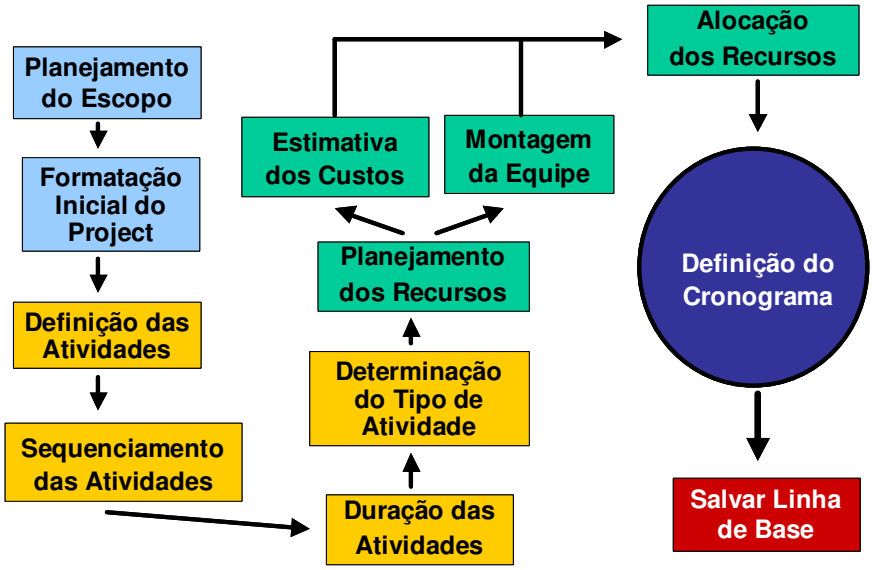
Todas estas disciplinas, aliadas às técnicas, métodos e ferramentas de cada uma, apoiam a condução do projeto de forma a garantir qualidade, atendimento aos prazos, custos e requisitos desejados.

Cabe ao gerente de projetos integrar todas essas disciplinas, cuidando de todos esses aspectos desde o início do projeto, passando pelo planejamento e outras fases do projeto, até a sua conclusão. A Integração dos processos também é tratada como uma disciplina pelo PMI."

Na perspectiva da utilização do software MS Project, todas estas disciplinas ou áreas de conhecimento podem e devem ser contempladas na realização de um projeto. A metodologia de gerenciamento de projetos é a base sobre a qual todos os trabalhos deverão ser efetuados. O software apenas auxiliará o gerente a estruturar a metodologia a uma forma mais eficaz e integrada de trabalho.

9.4 Etapas essenciais do Planejamento no MS Project

Segundo Kimura (2002, p. 11), deve-se seguir uma estrutura simples, mas eficaz, de planejamento para a utilização do MS Project.. Na figura abaixo, são apresentadas as etapas essenciais para que se possa elaborar o escopo do projeto no software MS Project. Cada etapa será definida no software.



10 QUALIDADE DE SOFTWARE

10.1 Introdução

Hoje em dia, muito se fala em qualidade de software, mas nem sempre as pessoas têm uma noção desse conceito. Pode-se considerar qualidade sob diferentes pontos de vista e, portanto, pode-se ter diferentes definições sendo algumas das mais comuns listadas a seguir:

- • Software sem defeitos.
- • Software adequado ao uso (conforme a definição de qualidade de Juran).
- • Software que atende as especificações (conforme a definição de qualidade de Crosby).
- • Software produzido por uma empresa que possui o certificado ISO9000 para seu sistema de qualidade.
- • Software que possui confiabilidade / usabilidade / manutenibilidade.

Pessoas com diferentes interesses sobre um produto têm visões diferentes sobre o conceito de qualidade. Por exemplo, clientes (mercado) usualmente consideram que o software tem qualidade se possui características que atendam suas necessidades. Desenvolvedores usualmente vêem a qualidade através das medidas de suas propriedades que são comparadas com indicadores de qualidade preestabelecidos. Para o setor de software um produto de qualidade é aquele com custo mínimo associado ao re-trabalho durante o desenvolvimento e após a entrega do produto.

Não tem sentido produzir um grande sistema de software se ele não funciona, não faz exatamente o que o cliente espera, não fica pronto no prazo ou se não puder merecer confiança, ou ainda, se não puder ser modificado ou mantido. O software é desenvolvido a um custo cada vez maior, com menor produtividade e com menos qualidade. Equipes individuais de projeto tentam ser mais produtivas mas, no contexto de uma empresa que não se preocupa com a qualidade, tais esforços no nível de projeto provavelmente não trarão resultados expressivos, visto que na maioria dessas empresas vários projetos de desenvolvimento de sistemas são cancelados antes de serem concluídos. Isto significa que iniciativas individuais de programadores e analistas de sistemas para melhoria de qualidade dificilmente mudarão a cultura da empresa onde os sistemas são desenvolvidos. Mais importante são as iniciativas no nível corporativo.

10.2 Gerenciamento da Qualidade de Software

O Gerenciamento da Qualidade exerce um papel fundamental para o desenvolvimento de software. Desde o início de um projeto, a qualidade deve ser vista como um fator crítico para o sucesso do software e deve ser considerada durante todo o ciclo de vida do mesmo.

De acordo com o [PMBOK2000], o Gerenciamento da Qualidade do Projeto inclui os processos necessários para garantir que o projeto irá satisfazer as necessidades para a qual ele foi empreendido. Isso envolve todas as atividades da função gerencial que determinam as políticas, os objetivos e as responsabilidades da qualidade e os implementam no sistema através de meios como planejamento da qualidade, controle da qualidade, garantia da qualidade e melhoria da qualidade.

A seguir é apresentada uma visão geral dos principais processos do Gerenciamento da Qualidade do Projeto:

10.2.1 Planejamento da Qualidade

O processo de planejamento da qualidade de um projeto consiste em identificar que padrões de qualidade são relevantes para o projeto e determinar como satisfazê-los. Este processo deve ocorrer em paralelo com os processos de planejamento do projeto como um todo, pois qualquer mudança no produto do projeto, necessária para atender os padrões de qualidade, podem exigir ajustes no prazo, no custo, ou mesmo, uma análise detalhada de risco do projeto. No contexto do desenvolvimento de software, o planejamento da qualidade deve incluir a documentação das atividades de prevenção e detecção de defeitos, tais como auditorias, inspeções, teste, coleta de métricas, além do uso de padrões e descrição do processo de software adotado.

10.2.2 Garantia da Qualidade

A Garantia da Qualidade consiste de todas as atividades planejadas e sistematizadas, implementadas no sistema de qualidade para prover segurança de que o projeto satisfaz os padrões da qualidade relevantes.

As atividades de garantia da qualidade de software se baseiam na prevenção de defeitos e problemas, que podem surgir nos produtos de trabalho.Definições de padrões, metodologias, técnicas e ferramentas de apoio ao desenvolvimento fazem parte deste contexto.

10.2.3 Controle da Qualidade

O controle da qualidade envolve monitorar os resultados específicos do projeto para determinar se eles estão de acordo com os padrões da qualidade relevantes e identificar formas de eliminar as causas dos resultados insatisfatórios. Os resultados do projeto incluem tanto os resultados do produto quanto os resultados do gerenciamento do projeto.

10.2.4 Modelos e Padrões da Qualidade

Com o crescimento do setor de software, vários modelos e padrões têm sido propostos ao longo dos últimos anos. Nesse contexto, alguns conceitos fundamentais são importantes de serem entendidos. O primeiro deles é a “política”, que representa um princípio governamental, tipicamente usado como base para regras, procedimentos ou padrões e geralmente estabelecido pela mais alta autoridade da organização. Um “padrão” pode ser entendido como uma base para comparação e é usado para suportar tamanho, conteúdo, valor ou qualidade de um objeto ou atividade. Um “guia”, por sua vez, é uma prática, método ou procedimento sugerido. Finalmente, um “modelo” pode ser definido como uma representação abstrata de um item ou processo, a partir de um ponto de vista específico. O objetivo do modelo é expressar a essência de algum aspecto de um item ou processo, sem prover detalhes desnecessários.

Diversos modelos de qualidade de software vêm sendo propostos ao longo dos últimos anos. Esses os modelos têm sido fortemente adotados por organizações em todo o mundo. Alguns destes modelos serão apresentados a seguir.

10.3 ISO

A entidade ISO (International Organization for Standardization) é a instituição internacional responsável pela padronização. Tem instituído várias normas para atender as necessidades.

10.3.1 ISO 9000

Em sua abrangência máxima engloba pontos referentes à garantia da qualidade em projeto, desenvolvimento, produção, instalação e serviços associados; objetivando a satisfação do cliente pela prevenção de não conformidades em todos os estágios envolvidos no ciclo da qualidade da empresa.

Foi aprovada em Genebra no ano de 1947. No Brasil, em 1987, tinha estrutura idêntica à norma britânica BS 5750, mas também influenciada por outras normas existentes nos Estados Unidos da América e por normas de defesa militar. Subdividia-se em três modelos de gerenciamento da qualidade:

- ISO 9001:1987: Modelo de garantia da qualidade para projeto, desenvolvimento, produção, montagem e prestadores de serviço - aplicava-se a organizações cujas atividades eram voltadas à criação de novos produtos.
- ISO 9002:1987: Modelo de garantia da qualidade para produção, montagem e prestação de serviço - compreendia essencialmente o mesmo material da anterior, mas sem abranger a criação de novos produtos.
- ISO 9003:1987: Modelo de garantia da qualidade para inspeção final e teste - abrangia apenas a inspeção final do produto e não se preocupava como o produto era feito.

Em 1994 cria-se a ISO 9001:1994. A norma tinha os seguintes requisitos:

- Responsabilidade da Direcção (Trata do papel da alta direcção na implementação do sistema da Qualidade);
- Sistema da qualidade (Descreve a documentação que compõe o sistema da qualidade);
- Análise do contrato (Trata da relação comercial entre a empresa e os seus clientes);
- Controle da concepção e projeto (Trata da concepção e desenvolvimento de novos produtos para atender aos clientes);
- Controle dos documentos e dados (Trata da forma de controlar os documentos do sistema da qualidade);
- Compras (Trata da qualificação dos fornecedores de materiais / serviços e do processo de compras);
- Produto fornecido pelo Cliente (Trata da metodologia para assegurar a conformidade dos produtos fornecidos pelo Cliente para incorporar ao produto final);
- Rastreabilidade (Trata da história desde o início do fabrico do produto ou da prestação do serviço);
- Controle do processo (Trata do processo de produção dos produtos da empresa);
- Inspecção e ensaios (Trata do controlo da qualidade que é realizado no produto ou serviço);
- Controle de equipamentos de inspecção, medição e ensaio (Trata do controle necessário para a calibração / verificação dos instrumentos que inspeccionam, meçam ou ensaiem a conformidade do produto);
- Situação da inspecção e ensaios (Trata da identificação da situação da inspecção do produto ou serviço em todas as etapas da sua produção);
- 13 Controle do produto não conforme (Trata da metodologia de controlo para os produtos fora de especificação);
- Ação corretiva e preventiva (Trata das acções necessárias para as não conformidades

identificadas de forma a evitar que aconteça e a sua repetição);

- Manuseamento, armazenamento, embalagem, preservação e expedição (Trata dos cuidados com o produto acabado até a sua expedição para o cliente);
- Controle dos registros da qualidade (Trata da metodologia do controlo dos registos da qualidade para facilitar a sua identificação,recuperação);
- Auditorias internas da qualidade (Trata da programação das auditorias internas da qualidade);
- Formação (Trata do levantamento de necessidades de formação e da programação das respectivas formações);
- Serviços após - venda (Trata dos serviços prestados após venda);
- Técnicas estatísticas (Trata da utilização de técnicas estatísticas na empresa);

Em 2005 foi lançada a ISO 9000:2005, descrevendo os fundamentos de sistemas de gestão da qualidade e definindo os termos a ela relacionados.

Hoje existe a ISO 9001:2008, que é considerada a versão final, pois apenas pequenas mudanças foram feitas.

10.3.2 Aspectos a serem abordados no momento da implementação

No momento da implementação da norma ISO 9001, devem ser observados:

- **Responsabilidade da direção:** requer que a política de qualidade seja definida, documentada, comunicada, implementada e mantida. Além disto, requer que se designe um representante da administração para coordenar e controlar o sistema da qualidade.
- **Sistema da qualidade:** deve ser documentado na forma de um manual e implementado também.
- **Análise crítica de contratos:** os requisitos contratuais devem estar completos e bem definidos. A empresa deve assegurar que tenha todos os recursos necessários para atender às exigências contratuais.
- **Controle de projeto:** todas as atividades referentes à projetos (planejamento, métodos para revisão, mudanças, verificações, etc.) devem ser documentadas.
- **Controle de documentos:** requer procedimentos para controlar a geração, distribuição, mudança e revisão em todos os documentos codificados na empresa.
- **Aquisição**: deve-se garantir que as matérias-primas atendam às exigências especificadas. Deve haver procedimentos para a avaliação de fornecedores.
- **Produtos fornecidos pelo cliente:** deve-se assegurar que estes produtos sejam adequados ao uso.
- **Identificação e rastreabilidade do produto:** requer a identificação do produto por item, série ou lote durante todos os estágios da produção, entrega e instalação.
- **Controle de processos:** requer que todas as fases de processamento de um produto sejam controladas (por procedimentos, normas, etc.) e documentadas.
- **Inspeção e ensaios:** requer que a matéria-prima seja inspecionada (por procedimentos documentados) antes de sua utilização.
- **Equipamentos de inspeção, medição e ensaios:** requer procedimentos para a calibração/aferição, o controle e a manutenção destes equipamentos.

- **Situação da inspeção e ensaios:** deve haver, no produto, algum indicador que demonstre por quais inspeções e ensaios ele passou e se foi aprovado ou não.
- **Controle de produto não-conformes:** requer procedimentos para assegurar que o produto não conforme aos requisitos especificados é impedido de ser utilizado inadvertidamente.
- **Ação corretiva:** exige a investigação e análise das causas de produtos não-conformes e adoção de medidas para prevenir a reincidência destas não-conformidades.
- **Manuseio, armazenamento, embalagem e expedição:** requer a existência de procedimentos para o manuseio, o armazenamento, a embalagem e a expedição dos produtos.
- **Registros da qualidade:** devem ser mantidos registros da qualidade ao longo de todo o processo de produção. Estes devem ser devidamente arquivados e protegidos contra danos e extravios.
- **Auditorias internas da qualidade:** deve-se implantar um sistema de avaliação do programa da qualidade.
- **Treinamento:** devem ser estabelecidos programas de treinamento para manter, atualizar e ampliar os conhecimentos e as habilidades dos funcionários.
- **Assistência técnica:** requer procedimentos para garantir a assistência à clientes.
- **Técnicas estatísticas:** devem ser utilizadas técnicas estatísticas adequadas para verificar a aceitabilidade da capacidade do processo e as características do produto.

10.3.3 Vantagens da certificação ISO 9000

A certificação ISO 9000 pode trazer diversos benefícios às empresas, tais como agregação de valor ao negócio; maior participação nos mercados nacionais e internacionais, pois os padrões atendem a especificações técnicas e requisitos internacionais, o que as tornam amplamente aceitas em diversos países; maior facilidade no controle do processo produtivo, através de procedimentos escritos e nas instruções de trabalho específicas, as atividades ficam mais fáceis de serem executadas; melhora a relação com os fornecedores; uso eficiente dos recursos; redução de falhas no processo produtivo, desperdícios e re-trabalhos; redução de custos; aumento da satisfação dos clientes, pois oferece um melhor serviço, melhores produtos, além de fazer o tratamento de suas reclamações. A certificação também é uma excelente estratégia de marketing, a divulgação de selo da certificação na qualidade em conjunto com a marca da empresa, contribui para a formação de uma boa imagem junto ao mercado, como também a conquista do reconhecimento e respeito de seus colaboradores, fornecedores, consumidores, comunidade e governo. A ISO também tem vantagens para:

- **Cliente:** segurança da fonte proveniente; evitam danos a saúde; têm grande satisfação com o produto ou serviço;
- **Meio-ambiente:** evita a poluição, redução de custos;
- **Sociedade:** Menor consumo de energia, menor desperdício, benefícios gerais;
- **Colaboradores** / **Empregados**: menos conflitos no trabalho e maior integração entre setores; maior desenvolvimento individual em cada tarefa, possibilitando melhoria de desempenho; maiores oportunidades de treinamento; menores possibilidades de acidentes de trabalho; melhores condições para acompanhar e controlar os processos; melhoria da qualidade e da produtividade, gerando possibilidades de recompensas.

10.3.4 ISO 9126

A norma 9126 se foca na qualidade do produto de software, propondo Atributos de Qualidade, distribuídos em seis características principais, com cada uma delas divididas em sub-características.

a) Funcionalidade

A capacidade de um software prover funcionalidades que satisfaçam o usuário em suas necessidades declaradas e implícitas, dentro de um determinado contexto de uso. Suas sub-características são:

- **Adequação:** que mede o quanto o conjunto de funcionalidades é adequado às necessidades do usuário;
- **Acurácia (ou precisão):** representa a capacidade do software de fornecer resultados precisos ou com a precisão dentro do que foi acordado/solicitado;
- **Interoperabilidade:** que trata da maneira como o software interage com outro(s) sistema(s) especificados;
- **Segurança:** mede a capacidade do sistema de proteger as informações do usuário e fornecê-las apenas (e sempre) às pessoas autorizadas;

b) Confiabilidade

O produto se mantém no nível de desempenho nas condições estabelecidas. Suas sub-características são:

- **Maturidade:** entendida como sendo a capacidade do software em evitar falhas decorrentes de defeitos no software;
- **Tolerância a Falhas:** representando a capacidade do software em manter o funcionamento adequado mesmo quando ocorrem defeitos nele ou nas suas interfaces externas;
- **Recuperabilidade:** que foca na capacidade de um software se recuperar após uma falha, restabelecendo seus níveis de desempenho e recuperando os seus dados;

c) Usabilidade

A capacidade do produto de software ser compreendido, seu funcionamento aprendido, ser operado e ser atraente ao usuário.

Note que este conceito é bastante abrangente e se aplica mesmo a programas que não possuem uma interface para o usuário final. Por exemplo, um programa batch executado por uma ferramenta de programação de processos também pode ser avaliado quanto a sua usabilidade, no que diz respeito a ser facilmente compreendido, aprendido, etc. Além disto, a operação de um sistema é uma interface Humano-Computador (ver IHC) sujeita às avaliações de usabilidade. Suas sub-características são:

- **Inteligibilidade:** que representa a facilidade com que o usuário pode compreender as suas funcionalidades e avaliar se o mesmo pode ser usado para satisfazer as suas necessidades específicas;
- **Apreensibilidade:** identifica a facilidade de aprendizado do sistema para os seus potenciais usuários;
- **Operacionalidade:** é como o produto facilita a sua operação por parte do usuário, incluindo a maneira como ele tolera erros de operação;
- **Atratividade:** envolve características que possam atrair um potencial usuário para o sistema, o que pode incluir desde a adequação das informações prestadas para o

usuário até os requintes visuais utilizados na sua interface gráfica;

d) Eficiência

O tempo de execução e os recursos envolvidos são compatíveis com o nível de desempenho do software. Suas sub-características são:

- **Comportamento em Relação ao Tempo:** que avalia se os tempos de resposta (ou de processamento) estão dentro das especificações;
- **Utilização de Recursos:** que mede tanto os recursos consumidos quanto a capacidade do sistema em utilizar os recursos disponíveis;

e) Manutenibilidade

A capacidade (ou facilidade) do produto de software ser modificado, incluindo tanto as melhorias ou extensões de funcionalidade quanto as correções de defeitos. Suas sub-características são:

- **Analisabilidade:** identifica a facilidade em se diagnosticar eventuais problemas e identificar as causas das deficiências ou falhas;
- **Modificabilidade:** caracteriza a facilidade com que o comportamento do software pode ser modificado;
- **Estabilidade:** avalia a capacidade do software de evitar efeitos colaterais decorrentes de modificações introduzidas;
- **Testabilidade:** representa a capacidade de se testar o sistema modificado, tanto quanto as novas funcionalidades quanto as não afetadas diretamente pela modificação;

f) Portabilidade

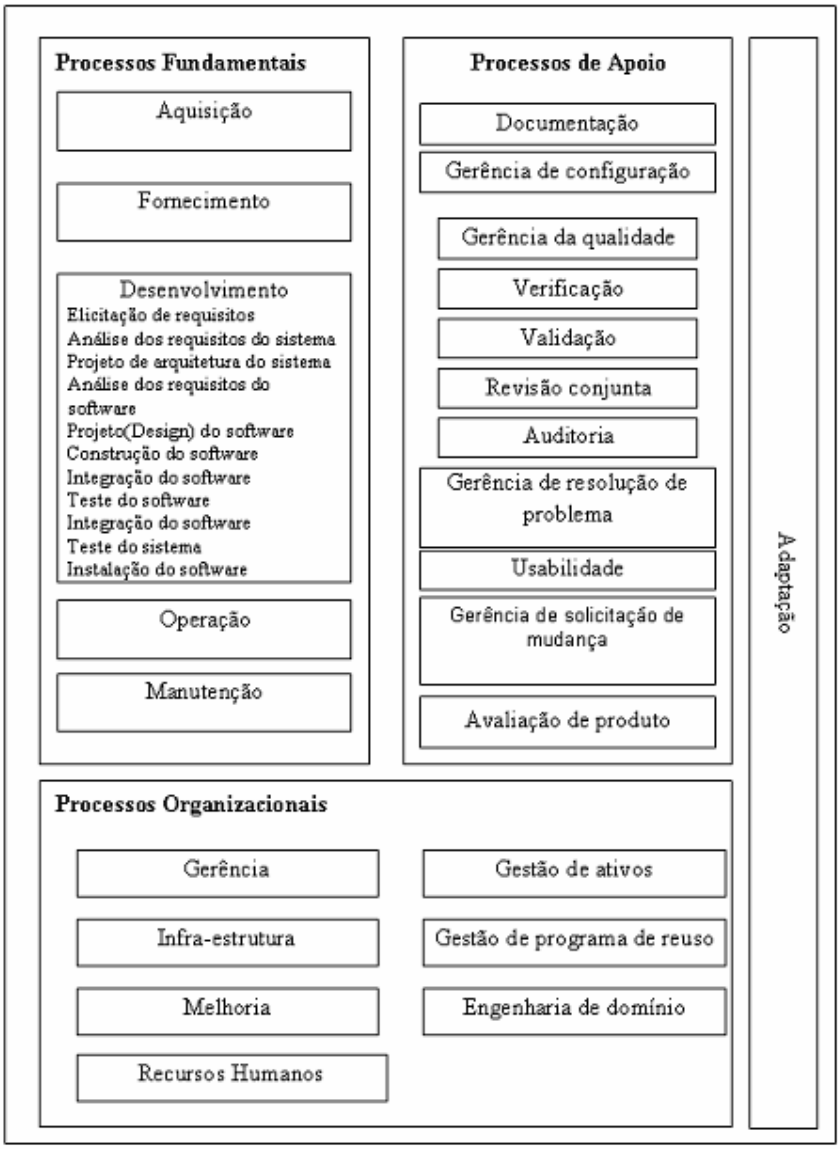
A capacidade do sistema de ser transferido de um ambiente para outro. Como "ambiente", devemos considerar todo os fatores de adaptação, tais como diferentes condições de infra-estrutura (sistemas operacionais, versões de bancos de dados, etc.), diferentes tipos e recursos de hardware (tal como aproveitar um número maior de processadores ou memória). Além destes, fatores como idioma ou a facilidade para se criar ambientes de testes devem ser considerados como características de portabilidade. Suas sub-características são:

- **Adaptabilidade:** representando a capacidade do software ser a adaptar a diferentes ambientes sem a necessidade de ações adicionais (configurações);
- **Capacidade para ser Instalado:** identifica a facilidade com que pode se instalar o sistema em um novo ambiente;
- **Coexistência**: mede o quão facilmente um software convive com outros instalados no mesmo ambiente;
- **Capacidade para Substituir:** representa a capacidade que o sistema tem de substituir outro sistema especificado, em um contexto de uso e ambiente específicos. Este atributo interage tanto com adaptabilidade quanto com a capacidade para ser instalado.

10.3.5 ISO 12207

A Norma ISO/IEC NBR 12207 foi criada pela ISO (*Institute of Organization for Standardization*) e o IEC (*International Electrotechnical Commission*) dentro de um esforço conjunto dessas organizações. A ISO/IEC 12207 teve seu desenvolvimento proposto em 1988 e a primeira versão foi publicada em agosto de 1995 e em 1998 foi publicada a versão

brasileira. Em 2002 e 2004 foram feitas atualizações na norma gerando as ementas 1 e 2 respectivamente [Machado, 2006].



O objetivo da ISO/IEC 12207 é estabelecer uma estrutura comum para os processos de ciclo de vida de software, com uma terminologia bem definida, que pode ser referenciada pela indústria de software. A estrutura contém processos, atividades e tarefas que servem para ser aplicadas durante a aquisição de um sistema que contém software, de um produto de software independente ou de um serviço de software, e durante o fornecimento, desenvolvimento, operação e manutenção de produtos de software [NBR ISO/IEC 12207, 1998].

O escopo da ISO/IEC 12207 abrange todo o ciclo de vida de software, desde sua concepção até a descontinuidade do projeto de software, e por todos os envolvidos com produção, manutenção e operação do software. A norma pode ser aplicada para toda a organização, mas existem casos de aplicação em projetos específicos por imposição contratual ou nas fases iniciais de implantação [NBR ISO/IEC 12207, 1998].

Os processos da ISO/IEC 12207 são agrupados de acordo com sua natureza, ou seja, o seu objetivo principal no ciclo de vida de software. Este agrupamento resultou nas 3 classes de processos a seguir: Processos Fundamentais, Processos de Apoio e Processos Organizacionais. A figura ao lado (MACHADO, 2006) apresenta os processos de cada classe. Será abordado apenas os Processos Fundamentais.

a) Processos Fundamentais

Os Processos Fundamentais são basicamente todas as atividades que a empresa executa nos serviços de desenvolvimento, manutenção ou operação de software. Esses processos comandam a execução de todos os outros processos. Os cinco processos fundamentais de ciclo de vida são:

- Aquisição;
- Fornecimento;
- Desenvolvimento;
- Operação;
- Manutenção.

Na prática, o processo de Aquisição inicia o ciclo de vida de software. O processo de Fornecimento responde sobre a execução dos processos de Desenvolvimento, Operação e/ou Manutenção [Machado, 2006].

i) Aquisição

O Processo de Aquisição define as atividades a serem executadas pela organização de adquire ou sub-contrata um produto ou serviço de software. O propósito do Processo de Aquisição é obter um produto e/ou serviço que satisfaça a necessidade expressa pelo cliente. O processo inicia com a identificação de uma necessidade do cliente e termina com a aceitação do produto e/ou serviço [NBR ISO/IEC 12207, 1998].

A ISO/IEC 12207 define o propósito e os resultados para os sub-processos de Preparação para Aquisição, Seleção de Fornecedor, Monitoração do Fornecedor e Aceitação pelo Cliente.

ii) Fornecimento

O Processo de Fornecimento é a sustentação para a execução dos processos de desenvolvimento, manutenção e/ou operação do produto ou serviço de software. O processo se inicia na preparação de uma proposta para atendimento de um pedido de proposta de um adquirente e encerra-se com a entrega do produto ou serviço de software. O propósito do Processo de Fornecimento é estabelecer um produto ou serviço para o cliente que atenda os requisitos acordados [NBR ISO/IEC 12207, 1998].

A ISO/IEC 12207 define o propósito e os resultados para os sub-processos de Proposta do Fornecedor, Acordo Contratual, Liberação do Produto e Suporte à Aceitação do Produto.

iii) Desenvolvimento

O Processo de Desenvolvimento contém as atividades e tarefas para o desenvolvimento do software, dentre elas: Elicitação de requisitos, análise de requisitos, projeto, construção, integração, testes e instalação.

O propósito do Processo de Desenvolvimento é transformar um conjunto de requisitos em um produto de software ou um sistema baseado em software que atenda às necessidades explicitadas pelo cliente [NBR ISO/IEC 12207, 1998].

A ISO/IEC 12207 define o propósito e os resultados para os sub-processos de Elicitação de Requisitos, Análise dos Requisitos do Sistema, Projeto da Arquitetura do Sistema, Análise dos Requisitos do Software, Projeto do Software, Construção do Software, Integração do Software, Teste do Software, Integração do Sistema, Teste de Sistema e Instalação do Software.

iv) Operação

O Processo de Operação contém as atividades e tarefas para a operação do software e suporte operacional aos usuários. O propósito do Processo de Operação é operar o produto de software no seu ambiente e fornecer suporte aos clientes desse produto [NBR ISO/IEC 12207, 1998].

A ISO/IEC 12207 define o propósito e os resultados para os sub-processos de Uso Operacional e Suporte ao Cliente.

v) Manutenção

O Processo de Manutenção é ativado quando o produto de software é submetido a modificações no código e na documentação associada devido a um problema ou a uma necessidade de melhoria ou adaptação. Seu objetivo é modificar o produto de software garantindo sua integridade. Este processo ainda inclui as possibilidades de migração e descontinuidade do produto de software.

O propósito do Processo de Manutenção é modificar um produto de software ou sistema após a sua entrega apara corrigir falhas, melhorar o desempenho ou outros atributos, ou adaptá-lo a mudanças do ambiente [NBR ISO/IEC 12207, 1998].

10.3.6 ISO 12119

A Norma NBR ISO/IEC 12119 Pacotes de software – Teste e requisitos de qualidade é aplicável a pacotes de software. São exemplos: processadores de texto, planilhas eletrônicas, bancos de dados, software gráficos, programas para funções técnicas ou científicas e programas utilitários.

Esta norma estabelece requisitos de qualidade e instruções de teste (em particular para teste por terceira parte) para pacotes de software, sendo estruturada conforme figura abaixo [NBR ISO/IEC 12119].

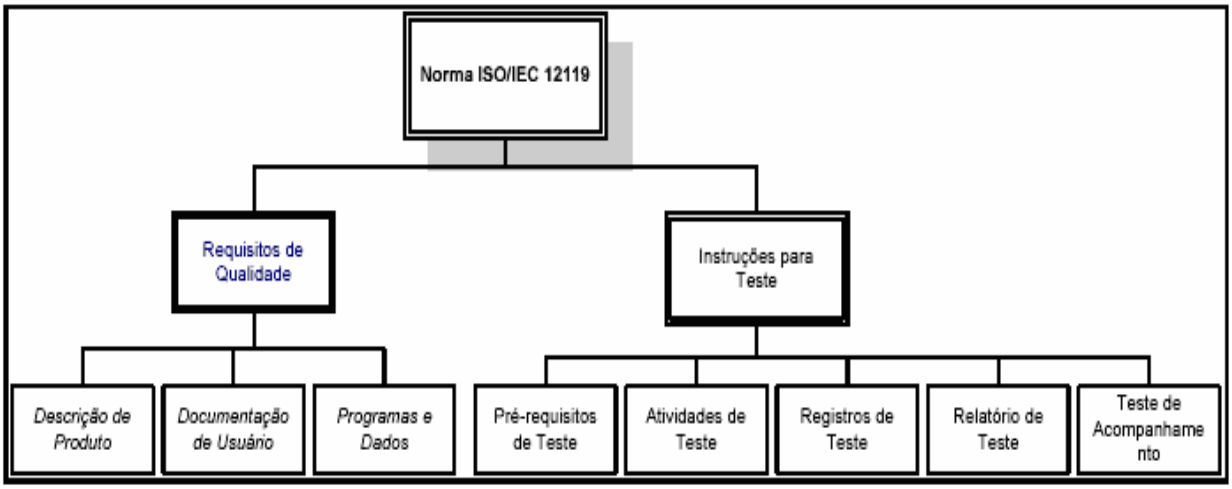


***Figura** – Estrutura da Norma NBR ISSO / IEC 12119*

Desta forma, a norma ISO 12119 não trata de processos de produção de software, trata de pacotes de software da forma como são oferecidos e liberados para uso. Ou seja, o sistema de qualidade do produtor está fora do escopo desta norma. Neste sentido, os potenciais usuários desta norma podem ser:

- Fornecedores que estejam especificando os requisitos para um pacote de software, projetando um modelo para descrever produtos, divulgando seus próprios produtos, submetendo produtos à certificação;
- Entidades de certificação que pretendam estabelecer um modelo de certificação por terceira parte;
- Entidades de credenciamento que credenciam entidades de certificação ou laboratórios de teste;
- Laboratórios que deverão seguir as instruções de teste para certificação ou para a emissão de marca de conformidade com a norma;
- Auditores quando julgam a competência de laboratórios de teste;

- Compradores que pretendam comparar os requisitos necessários para executar uma determinada tarefa com a informação presente nas descrições de produtos existentes ou verificar se os requisitos foram atendidos; Usuários que pretendam se beneficiar com produtos melhores.

O contexto deste trabalho se encaixa como um usuário do tipo “fornecedor” que deseja avaliar a conformidade do seu produto com a norma.

A seguir temos a listagem de algumas das definições utilizadas pela norma e seu entendimento é de fundamental importância para o entendimento da avaliação apresentada adiante. Estas são:

- Documento de Requisitos: documento contendo quaisquer combinações de recomendações, requisitos ou regulamentações a serem atendidas por um pacote de software.
- Descrição de Produto: documento expondo as propriedades de um pacote de software, com o principal objetivo de auxiliar os potenciais compradores na avaliação adequada do produto antes de sua aquisição.
- Documentação de Usuário: conjunto completo de documentos, disponível na forma impressa ou não, que é fornecido para a utilização do produto, sendo também uma parte integrante do produto.
- Documentação de Pacote: documentação de usuário e o documento de requisitos.
- Função: implementação de um algoritmo em um programa com o qual o usuário ou o programa pode realizar toda uma tarefa ou parte dela.
- Guia de teste (test case): instrução documentada para o responsável pelo teste que especifica como deve ou convém que seja testada uma função ou uma combinação de funções.

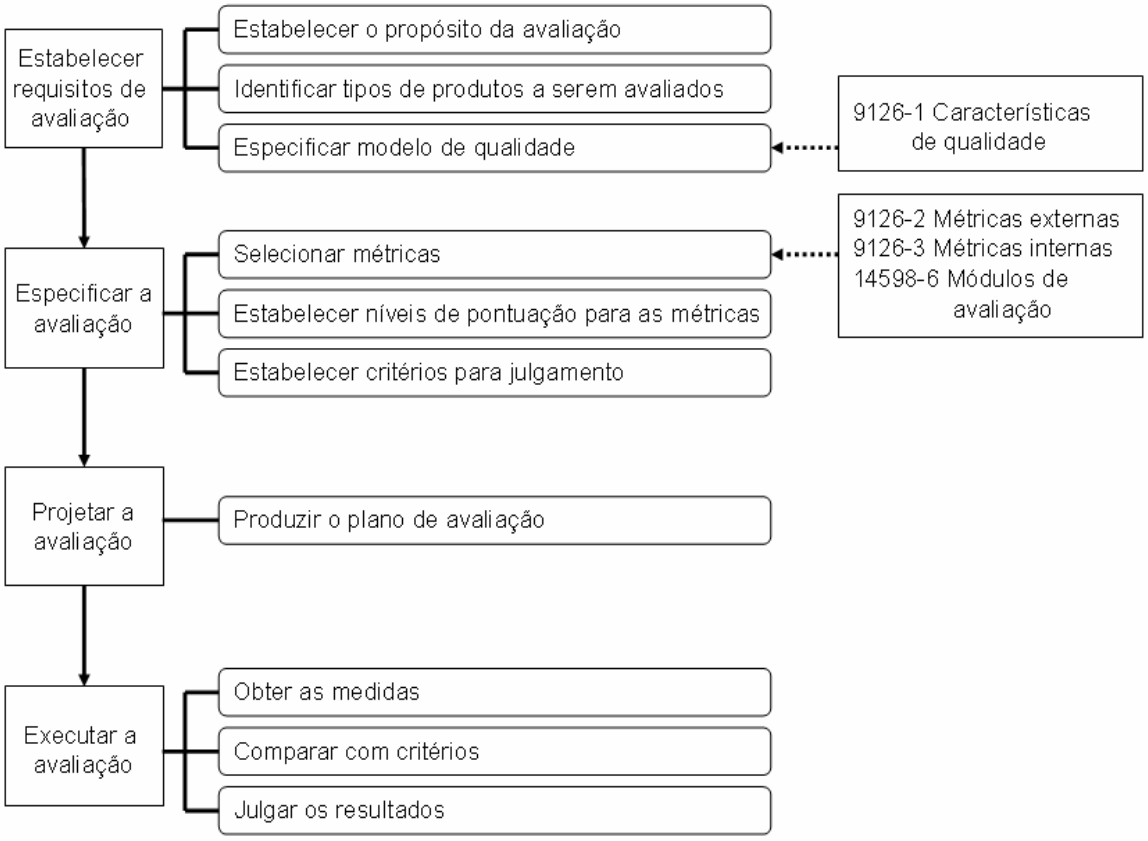
A norma ISO/IEC 12119 estabelece que um pacote de software está em conformidade com a mesma se atende a todos os requisitos1 estabelecidos para o Documento de Requisitos, Documentação de Usuário e Programas e Dados integrantes do software. Os requisitos indicados pelo uso da forma verbal “convém que” são opcionais [NBR ISO/IEC 12119].

10.3.7 ISO 14598

A norma fornece requisitos e recomendações para implementação prática da avaliação de produtos de software. O processo de avaliação é baseado na norma ISO 9126, que já define métricas de qualidade de software e pode ser usado tanto para avaliar produtos prontos como produtos em desenvolvimento.

a) ISO 14598 - 1: Visão Geral

Apresenta uma visão genérica do processo de avaliação, se aplica tanto à avaliação de componentes como do sistema, e pode ser aplicada a qualquer fase do ciclo de vida do produto. O processo de avaliação encontra-se subdividido em quatro fases principais, como ilustra a Figura abaixo, que serão comentados a seguir:



i) Estabelecer requisitos de avaliação

Esta fase se divide em três passos, a saber:

- Estabelecer o propósito da avaliação: Neste passo, deve ser definido qual o objetivo da avaliação, de forma a garantir que o produto forneça a qualidade necessária.
- Identificar tipos de produtos a serem avaliados: Nessa etapa deve ser definido o tipo de produto que será trabalhado. Os tipos de produtos aqui mencionados não dizem respeito à aplicação do software, mas ao estágio atingido pelo produto em seu ciclo de vida, o que determina quando e qual produto intermediário ou final será avaliado.
- Especificar o modelo de qualidade: Este passo se refere a definição de um modelo de qualidade sobre o qual será realizada a avaliação. O modelo de qualidade a ser definido através da utilização da norma 9126-1 como guia.

ii) Especificar a avaliação

Esta fase também se divide em três passos, são eles:

- Selecionar métricas: Características e sub-características de qualidade não podem ser medidas diretamente. Consequentemente, métricas correlacionadas às características de qualidade devem ser definidas.
- Estabelecer níveis de pontuação para as métricas: Uma métrica tipicamente envolve a produção de uma pontuação em alguma escala, refletindo a performance particular do sistema a respeito da característica de qualidade em questão.Uma vez que a qualidade se refere às necessidades especificadas, não existem regras genéricas de determinar quando uma pontuação é satisfatória.
- Estabelecer critérios para julgamento: Cada medida contribui para o julgamento geral

do produto, mas não necessariamente de maneira uniforme. Pode ser, por exemplo, que um requisito seja crítico, enquanto outro é desejável, mas não estritamente necessário. Neste caso, se um sistema não se comporta muito bem com respeito à característica crítica, irá ser avaliado negativamente independente do que ocorra a todas as outras características.

iii) Projetar a avaliação

Projetar a avaliação envolve a produção de um plano de avaliação, responsável por descrever os métodos de avaliação e um cronograma das ações do avaliador.

iv) Executar avaliação

Esta fase final encontra-se divida em 3 passos:

- Obter as medidas: consiste em uma pontuação apropriada na escala da métrica utilizada.
- Comparar com critérios: o valor medido é comparado com critérios predeterminados.
- Julgar os resultados: O julgamento é a etapa final do processo de avaliação, onde um conjunto de níveis pontuados é resumido. A qualidade resumida é então comparada com outros aspectos como tempo e custo. Finalmente, uma decisão gerencial será tomada baseada nos critérios gerenciais. O resultado é uma decisão gerencial quanto à aceitação ou rejeição, ou quanto à liberação ou não do produto de software.

b) ISO 14598 - 2: Planejamento e Gestão

Essa norma induz à especificação de um plano de avaliação. Esse inclui o desenvolvimento, a aquisição, a padronização e controle do processo como um todo. Reúne todos os elementos necessários para realizar a avaliação numa organização. Tem por finalidade, não somente, planejar e gerenciar, mas selecionar as métricas e ferramentas que serão utilizadas no processo de avaliação dos produtos de software.

c) ISO 14598 - 3: Processo para desenvolvedores

Propõe medidas e avaliações da qualidade de software durante todo o ciclo de vida. Essa norma é usada por:

- gerentes de projetos com o intuito de monitorar o desenvolvimento dos produtos de software.
- analistas com o fim de melhor levantar os requisitos do sistema;
- pessoal da manutenção que realiza a reengenharia e reprojeto do software para adequá-lo às necessidades explícitas do usuário.

d) ISO 14598 - 4: Processo para adquirentes

Essa parte da norma tem por finalidade orientar o comprador na escolha do melhor produto de software, na hora da compra.

Os requisitos abordados pelo software devem ser conhecidos pelo comprador, isto é, o minimundo que está sendo representado; bem como, os objetivos, as tarefas realizadas e o ambiente que o software necessita para funcionar.

Os aspectos citados anteriormente são concernentes ao software. Deve-se levar em consideração aqueles referentes às normas legais que envolvem um contrato de software entre comprador e vendedor (produtor).

e) ISO 14598 - 5: Processo para avaliadores: pessoa ou organização que executa a avaliação.

Esse padrão provê guias para avaliação dos produtos de software. Define atividades necessárias para a análise da avaliação dos requisitos, especificação, projeto e ações para a execução das avaliações. Procura abranger qualquer tipo de software. O processo é adequado e aplicável a produtos já findados, bem como, àqueles encontrados em fase de desenvolvimento.

f) ISO 14598 - 6: Documentação de módulos de avaliação

Entende-se por módulo de avaliação um pacote contendo a tecnologia da avaliação para as características e sub-características de um determinado software, ou seja, os modelos de qualidade; dados e informações a respeito do plano de aplicação desses modelos.

10.4 Capability Maturity Model (CMM)

Um CMM pode ser definido como uma quantidade de "melhores práticas" para diagnóstico e avaliação de maturidade do desenvolvimento de softwares em uma organização.

O CMM descreve os estágios de maturidade que passam as organizações quando evoluem no seu ciclo de desenvolvimento de software, através de avaliação contínua, identificação de problemas e ações corretivas, dentro de uma estratégia de melhoria dos processos. Este caminho de melhoria é definido por cinco níveis de maturidade:

- Inicial
- Repetível
- Definido
- Gerenciado
- Otimizado

A cada nível de maturidade corresponde um conjunto de práticas de software e de gestão específicas, denominadas áreas-chave do processo. Estas devem ser implantadas para que a organização possa atingir o nível de maturidade desejado.

10.4.1 A Estrutura do CMM

- **A maturidade nivelada:** Um framework com disposição em camadas providenciando uma progressão para disciplina, que é necessário para adquirir melhoramentos contínuos.
- **Processo de chave áreas:** Processo de chave de áreas (Key process área (KPA)) identifica um cluster de atividade relatada que quando em performance coletiva, adquiri um conjunto de metas consideradas importantes.
- **Metas:** As metas de uma KPA resumem os estados que mais existem para aquele KPA. Os estados devem ser implementados de maneira efetiva e durável.
- **Recursos comuns:** Recursos comuns incluem praticas que implementam e institucionalizam a KPA.
- **Chaves Praticas:** As chaves praticas descrevem os elementos de infra-estruturas e praticas que contribuem mais efetivamente para implementação e institucionalização da KPA.

10.4.2 Modelo de Maturidade

Um modelo de maturidade é uma coleção estruturada de elementos que descrevem certos tipos de maturidade de uma organização. Um modelo de maturidade fornece, por exemplo:

- Um ponto de partida;
- Os benefícios dos usuários em experiências anteriores;
- Um vocabulário comum e uma visão compartilhada;
- Um framework para priorizar ações;
- Uma forma de definir as melhorias mais significativas para uma organização.

O modelo descreve a maturidade da empresa baseando-se dos projetos que a mesma está desenvolvendo e nos clientes relacionados.

10.4.3 Os 5 Níveis de Maturidade do CMM

a) Nível 1: Inicial

No nível 1 de maturidade os processos são geralmente ad hoc e caóticos. A organização geralmente não dispõe de um ambiente estável. O sucesso nessas organizações depende da competência e heroísmo dos seus funcionários e não no uso de processos estruturados. Devido ao imediatismo, o nível 1 de maturidade produz produtos e serviços que em geral funcionam, mas frequentemente excedem o orçamento e o prazo dos projetos.

b) Nível 2: Repetível

No nível 2 de maturidade, o desenvolvimento do software é repetido. O processo pode não se repetir para todos os projetos da organização. A organização pode usar ferramentas de Gerência de Projetos para mapear os custos e o prazo do projeto.

A adoção de um processo de desenvolvimento ajuda a garantir que práticas existentes sejam utilizadas em momentos de stress. Quando essas práticas são adotadas, os projetos decorrem (e são gerenciados) de acordo com o planejamento inicial.

O status do projeto e os serviços entregues são visíveis ao gerenciamento (por exemplo: é possível a visualização de marcos do projeto e o término da maioria das tarefas).

Técnicas de gerenciamento de projetos são estabelecidas para mapear custos, prazos, e funcionalidades. Um mínimo de disciplina nos processos é estabelecido para que se possam repetir sucessos anteriores em projetos com escopo e aplicação similares. Ainda há um risco significante de exceder as estimativas de custos e o prazo de desenvolvimento.

Este nível apresenta as seguintes KPAs (Key Process Areas ou áreas chave de processo):

- Gerenciamento de Requisitos
- Planejamento de Projetos
- Acompanhamento e Supervisão de Projetos
- Gerenciamento de Subcontratação
- Garantia de Qualidade de Software
- Gerenciamento de Configuração

c) Nível 3: Definido

No nível 3 de maturidade, uma organização alcançou todas as metas genéricas e específicas das áreas de processo designadas como de níveis 2 e 3. No nível 3 de maturidade,

processos são bem caracterizados e entendidos, e são descritos utilizando padrões, procedimentos, ferramentas e métodos.

A organização possui um conjunto de padrões de processos, os quais são a base do nível 3. Estes estão estabelecidos e são melhorados periodicamente. Estes processos padrões são usados para estabelecer uma consistência dentro da organização. Projetos estabelecem seus processos segundo o conjunto de padrões processuais da organização, de acordo com guias.

O gerenciamento da organização estabelece os objetivos dos processos baseado no conjunto de padrões predefinidos e garante que estes objetivos sejam encaminhados de forma apropriada.

Uma crítica distinção entre os níveis 2 e 3 é o escopo dos padrões e a descrição dos processos e procedimentos. No nível 2, os padrões e a descrição de processos e procedimentos podem ser bem diferentes em cada instância específica do processo (por exemplo, em um projeto particular). No nível 3, os padrões e a descrição de processos e procedimentos para o projeto são guiados pelo conjunto de padrões de processos da organização. O conjunto de padrões de processo da organização inclui os processos do nível 2 e 3. Como resultado, os processos realizados através da organização são consistentes exceto pelas diferenças permitidas pelos guias.

Outra distinção crítica é que, no nível 3, processos são geralmente descritos com mais detalhes e com mais rigor do que no nível 2. No nível 3, processos são gerenciados mais proativamente. Há entendimento acerca das inter-relações entre as atividades dos processos. Há medidas detalhadas dos processos, produtos e serviços.

KPAs deste nível:

- Revisões (peer review)
- Coordenação de Intergrupos
- Engenharia de Produto de Software
- Gerenciamento de Software Integrado
- Programa de Treinamento
- Definição do Processo da Organização
- Foco no Processo da Organização

d) Nível 4: Gerenciado

Utilizando métricas precisas, o gerenciamento pode efetivamente controlar os esforços para desenvolvimento de software. Em particular, o gerenciamento pode identificar caminhos para ajustar e adaptar o processo a projetos particulares, sem perda de métricas de qualidade ou desvios das especificações.

Organizações neste nível conseguem metas quantitativas para o processo de desenvolvimento de software e de manutenção.

Subprocessos são selecionados conforme a importância no desempenho total do processo. Esses subprocessos selecionados são controlados usando técnicas estatísticas e quantitativas.

Uma distinção crítica entre o nível de maturidade 3 e 4 é a previsibilidade do desempenho do processo. No nível 4, o desempenho do processo é controlado usando técnicas estatísticas e quantitativas, e é previsível quantitativamente. No nível 3, os processos são somente previsíveis qualitativamente.

d) Nível 5: Otimizado

No nível 5, uma organização adquiriu todas as metas específicas das áreas de processo dos níveis 2, 3, 4, e 5 e as metas genéricas dos níveis 2 e 3. Processos são continuamente melhorados com base no entendimento quantitativo das causas comuns de variação inerente aos processos.

No nível de maturidade 5, o foco é o contínuo progresso do desempenho dos processos, através da introdução de melhorias de inovação tecnológica e incremental. Objetivos de melhoria quantitativa dos processos para a organização são estabelecidos, continuamente revisados, refletindo as mudanças nos objetivos da organização, e usando critérios de melhoria na gerência de processos.

Os efeitos da melhoria da revisão dos processos são medidos e acompanhados, utilizando-se processos de melhoria de qualidade. Os processos definidos e o conjunto de processos padrões da organização são alvos de melhoria de métricas.

Uma distinção crítica entre os níveis 4 e 5 é o tipo de variação do processo. No nível 4, o interesse é verificar as causas especiais de variação de processo e fornecer resultados estatísticos. No nível 5, o foco são as causas comuns de variação de processo e a introdução de mudanças de modo a melhorar a performance do processo, atingindo objetivos quantitativos.

10.5 Total Quality Control (TQC)

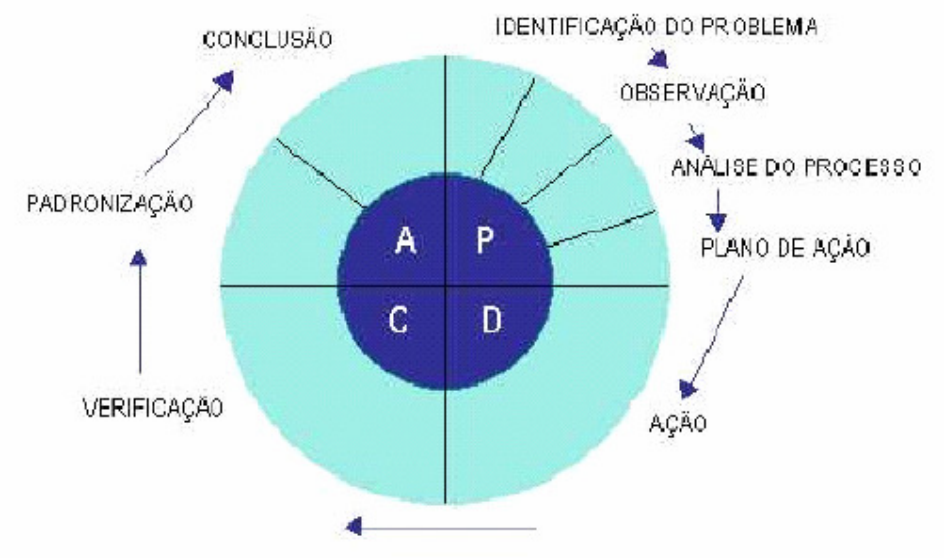
Total Quality Control, ou TQC, é um sistema desenvolvido no Japão, montado pelo Grupo de Pesquisa do Controle da Qualidade da Union of Japanese Scientists and Engineers (JUSE), para implementar a melhoria contínua da qualidade. Há mais de 40 anos, o TQC vem sendo aperfeiçoado por estudiosos na área de qualidade. Conforme praticado no Japão, o TQC se baseia na participação de todos os setores da empresa e de todos os empregados no estudo e condução do controle da qualidade. O TQC é regido pelos seguintes princípios básicos:

- Produzir e fornecer produtos e/ou serviços que atendam concretamente às necessidades do cliente;
- Garantir a sobrevivência da empresa através do lucro contínuo, adquirido pelo domínio da qualidade;
- Identificar o problema mais crítico e solucioná-lo pela mais alta prioridade;
- Falar, raciocinar e decidir com dados e com base em fatos;
- Gerenciar a organização ao longo do processo e não por resultados;
- Reduzir metodicamente as distorções através do isolamento de suas causas fundamentais;
- Não permitir a venda de produtos defeituosos;
- Não repetir erros;
- Definir e garantir a execução da visão e estratégia da alta direção da empresa.

Uma das principais formas de implementação do controle de processos e adotada pelo TQC é a utilização do Ciclo PDCA (Plan-Do-Check-Action), que consiste em 4 fases: planejar, executar, verificar e atuar corretamente. Figura abaixo.

Na fase Plan, o foco está na identificação do problema, análise do processo atual e definição do plano de ação para melhoria do processo em questão. Na fase Do, o plano de ação definido deve ser executado e controlado. Em Check, verificações devem ser realizadas, a fim de subsidiar ajustes e tirar lições de aprendizagem. Finalmente, em Action, deve-se atuar corretivamente para fundamentar um novo ciclo, garantindo a melhoria contínua.

O PDCA é considerado um ciclo clássico, adotado, até hoje, por organizações no mundo inteiro. Ele deve ser utilizado para todos os processos da organização, em todos os níveis hierárquicos. Ele pode ser utilizado como base para qualquer organização, na definição de uma metodologia de controle ou melhoria de qualquer tipo de processo.



Bases do TQC.

10.6 Total Quality Management (TQM)

Total Quality Management (TQM) é uma abordagem para sucesso a longo prazo, que é medida através da satisfação do cliente e baseada na participação de todos os membros da organização. TQM foca a melhoria de processos, produtos, serviços e cultura organizacional, e considera qualidade de um processo como responsabilidade do "dono" do processo. Nesse contexto, descreve-se os seguintes elementos-chave do TQM:

- **Foco no cliente** - consiste em analisar os desejos e necessidades do cliente, definir requisitos do cliente, além de medir e gerenciar a satisfação do cliente.
- **Melhoria de Processo** - o objetivo é reduzir a variação do processo e atingir
- **A melhoria contínua do processo**. Processos incluem tanto processos de negócio quanto processo de desenvolvimento de produtos.
- **Aspecto Humano** - nesse contexto, as áreas-chave incluem liderança, gerência, compromisso, participação total e outros fatores sociais, psicológicos e humanos.
- **Medição e Análise** - o objetivo é gerenciar a melhoria contínua em todos os parâmetros de qualidade por um sistema de medição orientado a metas.

11 BIBLIOGRAFIA

Aprimorando o conhecimento sobre gestão de qualidade total. Disponível em: <http://www.ucg.br/site_docente/eng/ximena/pdf/aula2.pdf> acesso em 19/10/2004

BRAZ JÚNIOR., Osmar de Oliveira. **Técnicas de Análise de Sistemas** (apostila). Tubarão: UNISUL. 1997.

CARNEIRO, Margareth Fabíola dos Santos; **Gerenciamento de Projetos** (apostila), ENAP, 2000

Conhecendo mais sobre TQC. Disponível em: <http://www.furnas.com.br/portug/institucional/glossario.asp?letra=t> acesso em 19/10/2004

DeMARCO, Tom. **Análise Estruturada e Especificação de Sistema** – 2ª reimpressão Rio de Janeiro: Editora Campus. 1989.

FOURNIER, Roger. **Guia Prático para Desenvolvimento e Manutenção de Sistemas Estruturados** – 1ª Edição. São Paulo: Makron Book. 1994.

KELLER, Robert. **Análise Estruturada na Prática** – 1ª edição. São Paulo: McGraw-Hill Editora. 1990.

KIMURA, Marcos. **Curso básico de MS Project** (apostila). Brasília: ENAP, 2002.

PAULA FILHO, Wilson de Pádua. **Engenharia de Software** – Fundamentos, Métodos e Padrões – 1ª edição. Rio de Janeiro: LTC Editora. 2001.

PARREIRA JÚNIOR, Walteno Martins & TEODORO NETO, Euclides Martins & GOMES, Gina Marcia dos Santos. **Análise e Programação Orientada ao Objeto** – in I Encontro de Iniciação Científica e III Encontro de Pesquisa - ILES/ULBRA – Itumbiara-GO - 2002

PRESSMAN, Roger S. **Engenharia de Software** – 3ª edição. São Paulo: Makron Books. 1995.

SOMMERVILLE, Ian. **Engenharia de Software** – 6ª edição. São Paulo: Addison Wesley. 2003.

YOURDON, Edward. **Administrando o ciclo de vida do sistema** – 2ª edição. Rio de Janeiro: Editora Campus. 1989.

_____. **Análise Estruturada Moderna**. 3ª Edição. Rio de Janeiro: Editora Campus. 1992.