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

APOSTILA
ENGENHARIA DE SOFTWARE 1

Prof. Walteno Martins Parreira Júnior
www.waltenomartins.com.br
waltenomartins@yahoo.com
2013

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

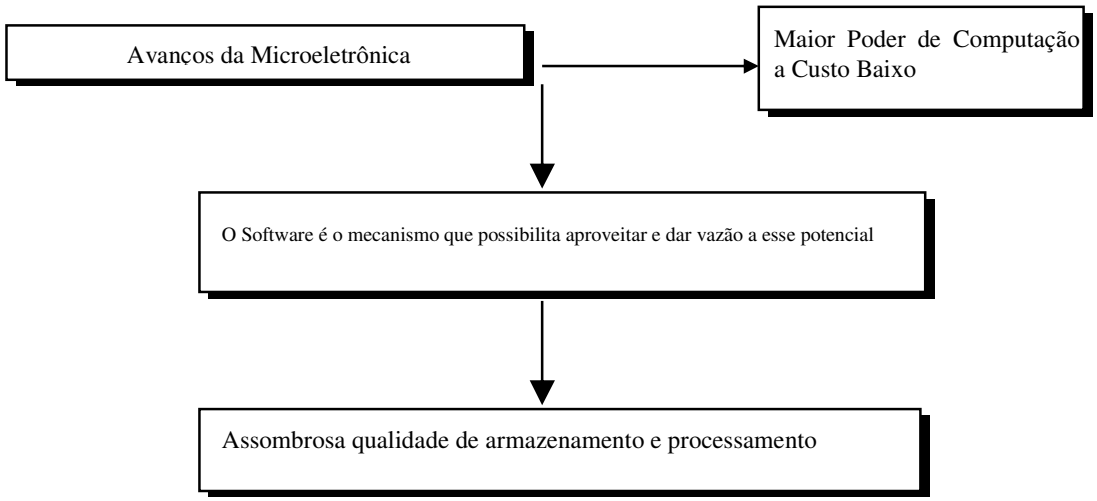
Existem um conjunto de problemas associados ao software que vários autores chamam de "crise do software". Este conjunto de problemas não se limitam ao software que não funciona adequadamente, mas ab range também prob lemas associados a forma de desenvolvimento destes softwares, a forma como é efetuada 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 sequência 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 consequência 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 sequência adequada e na combinação certa. Por vezes o analista poderá saber em que sequência 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 ferramenta 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 obtêm essas informações costuma 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 ANALISE DE REQUISITOS

3.1 Introdução

A compreensão dos requisitos de software é um ponto fundamental para o sucesso de um projeto de software. Independentemente da qualidade com que o software venha a ser projetado e implementado, ele certamente trará problemas para o cliente/usuário se a sua análise de requisitos foi mal realizada.

A análise de requisitos é uma tarefa que envolve, antes de tudo, um trabalho de entendimento, refinamento, modelagem e especificação das necessidades e desejos relativos ao software que deverá ser desenvolvido. Nesta tarefa, tanto o cliente quanto o desenvolvedor vão desempenhar um papel de grande importância, uma vez que caberá ao primeiro a formulação (de modo concreto) das necessidades em termos de funções e desempenho, enquanto o segundo atua como indagador, consultor e solucionador de problemas.

Logo, a engenharia de requisitos ajuda os engenheiros de software a compreender melhor o problema que eles vão trabalhar para resolver. Ela inclui o conjunto de tarefas que levam a um entendimento de qual será o impacto do software sobre o negócio, do que o cliente quer e de como os usuários vão interagir com o software.

Então, a etapa é de suma importância no processo de desenvolvimento de um software, principalmente porque ela estabelece o elo de ligação entre a alocação do software em nível de sistema (realizada na etapa de Engenharia de Sistema) e o projeto do software. Desta forma, esta etapa permite:

Ao engenheiro de software:

- especificar as necessidades do software em termos de funções e de desempenho;
- estabelecer as interfaces do software com os demais elementos do sistema;
- especificar as restrições de projeto.

Ao engenheiro de software (analista):

- uma alocação mais precisa do software no sistema;
- a construção de modelos do processo, dos dados e dos aspectos comportamentais que serão tratados pelo software.

Ao projetista:

- a obtenção de uma representação da informação e das funções que poderá ser traduzida em projeto procedimental, arquitetônico e de dados.

Além disso, é possível definir os critérios de avaliação da qualidade (teste) do software a serem verificados uma vez que o software esteja concluído.

3.2 O que são requisitos?

Como sistemas computacionais são construídos para serem utilizados no mundo real, é necessário que sejam modelados como uma parte do mundo real. O domínio de uma aplicação é uma atividade extremamente importante para se compreender a necessidade e a importância do sistema a ser construído e definir os requisitos que tornam o sistema útil.

Requisitos são objetivos ou restrições estabelecidas por clientes e usuários do sistema que definem as diversas propriedades do sistema. Os requisitos de software são, obviamente, aqueles dentre os requisitos de sistema que dizem respeito a propriedades do software.

Um conjunto de requisitos pode ser definido como uma condição ou capacidade necessária que o software deve possuir para que o usuário possa resolver um problema ou atingir um objetivo ou então para atender as necessidades ou restrições da organização ou dos outros componentes do sistema.

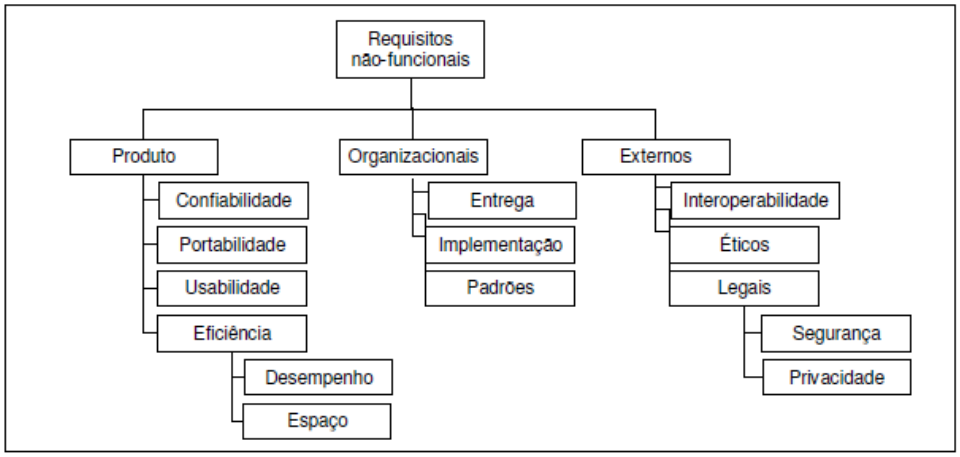
Tradicionalmente, os requisitos de software são classificados, segundo Sommerville, em requisitos funcionais, não-funcionais e de domínio. Os **requisitos funcionais** são a descrição das diversas funções que clientes e usuários querem ou precisam que o software ofereça. Eles definem a funcionalidade desejada do software. O termo função é usado no sentido genérico de operação que pode ser realizada pelo sistema, seja através comandos dos usuários ou seja pela ocorrência de eventos internos ou externos ao sistema. São exemplos de requisitos funcionais:

- "o software deve possibilitar o cálculo dos gastos diários, semanais, mensais e anuais com pessoal";
- "o software deve emitir relatórios de compras a cada quinze dias";
- "os usuários devem poder obter o número de aprovações, reprovações e trancamentos em todas as disciplinas por um determinado período de tempo".

A especificação de um **requisito funcional** deve determinar o que se espera que o software faça, sem a preocupação de como ele faz. É importante diferenciar a atividade de especificar requisitos da atividade de especificação que ocorre durante o design do software. No design do software deve-se tomar a decisão de quais a funções o sistema efetivamente terá para satisfazer àquilo que os usuários querem.

Requisitos não-funcionais são as qualidades globais de um software, como manutenibilidade, usabilidade, desempenho, custos e várias outras. Normalmente estes requisitos são descritos de maneira informal, de maneira controversa (por exemplo, o gerente quer segurança mas os usuários querem facilidade de uso) e são difíceis de validar. São exemplos de requisitos não-funcionais:

- "a base de dados deve ser protegida para acesso apenas de usuários autorizados";
- "o tempo de resposta do sistema não deve ultrapassar 30 segundo";
- "o software deve ser operacionalizado no sistema Linux";
- "o tempo de desenvolvimento não deve ultrapassar seis meses".



A necessidade de se estabelecer os requisitos de forma precisa é crítica na medida que o tamanho e a complexidade do software aumentam. Os requisitos exercem influência uns sobre os outros. Por exemplo, o requisito de que o software deve ter grande portabilidade (por exemplo, ser codificado em Java) pode implicar em que o requisito desempenho não seja satisfeito (programas em Java são, em geral, mais lentos).

Requisitos de domínio são os requisitos que se originam do domínio da aplicação do sistema e que refletem características desse domínio. Podem ser requisitos funcionais ou não-funcionais. Eles podem ser novos requisitos funcionais, podem também restringir requisitos funcionais existentes ou estabelecer como devem ser executados determinados cálculos específicos. São exemplos de requisitos de domínio:

- "deve haver uma interface padrão com o usuário para todos os bancos de dados, com base no padrão Z39.50";
- "em razão das restrições de direitos autorais, alguns documentos devem ser excluídos imediatamente após serem fornecidos".

Um exemplo de documentação para um requisito funcional e que contempla três requisitos não-funcionais:

Nome: RF4 – Manter funcionário			**Oculto ( ) Evidente** (x)	
Descrição: O sistema permite a consulta, alteração e exclusão do funcionário.				
Requisitos Dominio				
Nome	**Restrição**	**Categoria**	**Obrigatoriedade**	**Permanência**
RD4.1 – Atualizar campos;	O sistema deverá permitir a alteração dos campos do funcionário que já está cadastrado como: a função, a data de admissão, CPF, RG, nome, endereço, data de nascimento, n° da CTPS, telefone.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )
RD4.2 – Validar CPF;	O sistema permitirá a consulta do funcionário e alteração dos campos mediante a validação do CPF caso seja alterado.	Segurança	**O**(X) **D**( )	**P**(X) **T**( )
RD4.3 – Armazenar os dados.	O sistema permite armazenar os dados mediante a conclusão da alteração.	Especificação	**O**(X) **D**( )	**P**(X) **T**( )

Legenda: O – Obrigatório **D** – Desejável **P** – Permanente **T** - Transitório

3.3 Etapas da Análise de Requisitos

O analista executa ou coordena cada uma das tarefas associadas à análise de requisitos de software. Durante as tarefas de identificação, ele comunica-se com o pessoal do usuário/cliente com a finalidade de conhecer as características existentes no ambiente. O analista convoca o pessoal de desenvolvimento durante as tarefas de avaliação e síntese, de forma que as características do software sejam corretamente definidas. O analista geralmente é o responsável pelo desenvolvimento de uma Especificação de Requisitos de Software e participa de todas as revisões. Assim, a Análise de Requisitos é dividida em várias etapas.

3.3.1 Analise do Problema

Nesta fase inicial, o analista estuda os documentos de Especificação do Sistema e o Plano de Software, como forma de entender o posicionamento do software no sistema e revisar o escopo do software utilizado para definir as estimativas de projeto. Um elo de comunicação

entre o analista e o cliente deve ser estabelecido. A meta do analista nesta fase é identificar os principais fatores do problema a ser resolvido, pela ótica do cliente.

Logo, reconhecer o problema a ser resolvido. Nesta fase encontra-se a especificação do sistema, o planejamento, o contato do analista com o cliente com a intenção de entender a visão do cliente com relação ao problema.

3.3.2 Avaliação do problema e sintetização da solução

Esta fase envolve a análise dos fluxos de informação e a elaboração de todas as funções de tratamento e os aspectos de comportamento do software. Ainda, é importante que uma definição de todas as questões relacionadas à interface com o sistema, além de uma especificação das restrições do projeto.

Terminada a análise, o analista pode iniciar a síntese de uma ou mais soluções para o problema. Na síntese das eventuais soluções, o analista deve levar em conta as estimativas e as restrições de projeto. Este processo de avaliação e síntese prossegue até que o analista e o cliente estejam de acordo sobre a adequação das especificações realizadas para a continuidade do processo.

Nesta fase tem-se o entendimento do problema, e faz-se a identificação das informações que serão necessárias ao usuário, identificação das informações que serão necessárias ao sistema e a seleção da melhor solução possível dentro das soluções propostas.

3.3.3 Modelagem

A partir da tarefa de avaliação e síntese, o analista pode estabelecer um modelo do sistema, o qual permitirá uma melhor compreensão dos fluxos de informação e de controle, assim como dos aspectos funcionais e de comportamento. Este modelo, ainda distante de um projeto detalhado, servirá de referência às atividades de projeto, assim como para a criação da especificação de requisitos.

Em muitas situações como forma de reforçar o conhecimento sobre a viabilidade do software a ser desenvolvido, pode ser necessário o desenvolvimento de um protótipo de software como alternativa ou como trabalho paralelo à análise de requisitos.

Logo, modelar é um recurso usado para o suporte da síntese da solução, o modelo vai apresentar ferramentas que facilitarão o entendimento do sistema, como as funcionalidades, informações e comportamento do sistema.

3.3.4 Especificação dos Requisitos

A etapa de Análise de requisitos culmina com a produção de um documento de Especificação de Requisitos de Software, que registra os resultados das tarefas realizadas. Eventualmente, pode ser produzido como documento adicional um Manual Preliminar do Usuário. Embora pareça estranho, a produção deste manual permite que o analista passe a olhar para o software da ótica do cliente/usuário, o que pode ser bastante interessante, principalmente em sistemas interativos.

Então a etapa de especificar os requisitos consolida funções, interfaces, desempenho, o contexto e as restrições do sistema.

3.3.5 Revisão

De posse de um Manual do Usuário, mesmo em estágio preliminar, vai permite ao cliente uma revisão dos requisitos (de interface, pelo menos) ainda num estágio bastante prematuro do desenvolvimento de software. Desta forma, algumas indagações resultantes de uma má definição de alguns aspectos do software podem ser evitadas.

Portanto, juntos, cliente e analista, avaliarão o objetivo do projeto com o intuito de eliminar possíveis redundâncias, inconsistências e omissões do sistema, obtendo uma mesma visão.

Então, a revisão é observada primeiramente em nível macroscópico. Neste nível, os revisores tentam garantir que a especificação seja completa, consistente e precisa. As seguintes questões são consideradas:

- As metas e os objetivos do software permanecem consistentes com as metas e os objetivos do sistema?
- Interfaces importantes para todos os elementos do sistema foram descritas?
- O fluxo e a estrutura de informação são adequadamente definidos para o domínio da informação? Etc.

Para desenvolver respostas a muitas das questões apresentadas, a revisão pode focalizar um nível detalhado. Diretrizes para uma revisão detalhada:

- Estar alerta para perceber conectivos persuasivos (por exemplo, "certamente","claramente") e perguntar "por que eles estão presentes?".
- Procurar termos vagos (por exemplo, "usualmente", "às vezes"); peça esclarecimentos.
- Quando forem fornecidas listas que não estejam completas, certificar-se de que todos os itens sejam entendidos.
- Estar certo de que os limites declarados não contenham pressuposições não declaradas (por exemplo: "Os códigos válidos variam de 0 a 100". Inteiro? Reais?).

Logo que a revisão for concluída, a Especificação de Requisitos de Software é "assinada" pelo cliente e pelo desenvolvedor. A especificação torna-se um "contrato" de desenvolvimento de software.

3.4 Processos de comunicação

Na maioria das vezes o desenvolvimento de um software é motivado pelas necessidades de um cliente que tem a necessidade de automatizar um sistema existente ou obter um novo sistema completamente automatizado. E o software é desenvolvido por um desenvolvedor ou por uma equipe de desenvolvedores que normalmente não conhecem profundamente o negocio. Uma vez desenvolvido, o software será provavelmente utilizado por usuários finais, os quais não são necessariamente os clientes que originaram o sistema.

Assim, existem três partes envolvidas no processo de desenvolvimento de um produto de software: o cliente, o desenvolvedor e os usuários. Para que o processo de desenvolvimento seja conduzido com sucesso, é necessário que os desejos do cliente e as expectativas dos eventuais usuários finais do sistema sejam precisamente transmitidos ao desenvolvedor.

Este processo de transmissão de requisitos, do cliente e dos usuários ao desenvolvedor, invariavelmente apresenta relativa dificuldade, considerando alguns aspectos:

- geralmente os clientes não entendem de software ou do processo de desenvolvimento de um programa;
- o desenvolvedor, usualmente, não entende do sistema no qual o software vai executar.

A boa comunicação entre a equipe de desenvolvimento e o cliente é fundamental para a realização de sucesso da atividade de análise de requisitos. Nem sempre a boa comunicação é alcançada devido aos motivos já citados.

Uma Análise de Requisitos bem sucedida deve representar corretamente as necessidades do cliente e dos usuários, satisfazendo, porém às três partes envolvidas (incluindo o desenvolvedor). O que é verificado, em boa parte dos projetos de software é que o cliente nem sempre entende perfeitamente quais são as suas reais necessidades, assim como os usuários têm dificuldades para exprimir as suas expectativas com relação ao que está sendo desenvolvido. Um primeiro resultado desta etapa deve ser, sem dúvida, o esclarecimento a respeito do que são estas necessidades e expectativas.

3.5 Princípios da análise

Pesquisadores desenvolveram vários métodos de análise e especificação de software identificando problemas e suas causas. Cada método tem suas características que são únicas mas todos compartilham alguns princípios fundamentais: a) O domínio da informação, b) Modelos e c) Partições.

3.5.1 O domínio da informação

Composto por dados e eventos. Para entender o domínio da informação cada um destes três pontos de vista deve ser considerado para dados e eventos:

a) fluxo da informação: representa a maneira pela qual os dados e eventos se modificam à medida que cada um se movimenta pelo sistema. Ao longo deste caminho, novas informações podem ser introduzidas a partir de um depósito. Uma entrada pode ser transformada em informações intermediárias até alcançar a saída.

b) conteúdo da informação: representa os dados e os itens de controle que compõem um determinado item de informação mais amplo.

c) estrutura da informação: representa a organização interna dos dados que compõe um item de informação.

3.5.2 Modelagem

Obter maior compreensão do que deve ser construído. Os modelos devem ser capazes de modelar a informação que o software transforma, as funções que possibilitam as transformações e o comportamento do sistema quando a transformação está ocorrendo. Durante a análise de requisitos construímos modelos que se concentram naquilo que o software deve fazer e não como ele deve fazer. Papéis dos modelos criados durante a análise dos requisitos:

a) Ajuda o analista a entender a informação, função e o comportamento.

b) Torna-se ponto principal para revisão.

c) Torna-se base para o projeto a qual pode ser “mapeada” para um contexto de implementação.

3.5.3 Particionamento

Muitas vezes, os problemas a serem resolvidos são excessivamente complexos, apresentando grande dificuldade para a sua compreensão (e consequente resolução) como um todo. Para dominar de forma completa os problemas em análise, um princípio fundamental é a decomposição do problema em parte menores, denominado de particionamento.

A partir do particionamento do problema e da análise de cada parte, o entendimento fica facilitado. E é possível estabelecer as interfaces de cada parte do problema de modo que a função global do software seja realizada.

O procedimento básico de particionamento é o estabelecimento de uma estrutura hierarquizada de representação da função ou da informação, dividindo em partições o elemento superior, podendo esta divisão ser efetuada segundo uma abordagem vertical (deslocando-se verticalmente na hierarquia) ou horizontal.

3.6 Especificação

O modo de especificação reflete na qualidade da solução. Ela representa o que deve ser implementado. Princípios da especificação:

1) a separação entre funcionalidade e implementação;
2) utilização de uma linguagem de especificação orientada ao processo;
3) a especificação deve levar em conta o sistema do qual o software faz parte e o ambiente no qual o sistema vai operar;
4) a especificação deve representar a visão que os usuários terão do sistema;
5) uma especificação deve ser operacional, no sentido de que ela deve permitir, mesmo num nível de abstração elevado, algum tipo de validação;
6) uma especificação deve ser localizada e fracamente acoplada, o que são requisitos fundamentais para permitir a realização de modificações durante a

A figura a seguir apresenta uma proposta de estrutura para o documento de Especificação dos Requisitos do Software. O documento inicia com uma Introdução, que apresenta as principais metas do software, descrevendo o seu papel no contexto do sistema global. As informações prestadas nesta seção podem ser, inclusive, inteiramente extraídas do documento de Plano de Software.

O documento que apresenta a Especificação de Requisitos de Software tem o seguinte esboço, mas não exclusivamente:

I. Introdução
   1. Referências do sistema.
   2. Descrição geral.
   3. Restrições de projeto do software.

II. Descrição da Informação.
   1. Representação do fluxo de informação
      a. Fluxo de dados.
      b. Fluxo de controle.
   2. Representação do conteúdo de informação.
   3. Descrição da interface com o sistema.

III. Descrição Funcional
   1. Divisão funcional em partições
   2. Descrição funcional
      a. Narrativa de processamento.
      b. Restrições/limitações.
      c. Exigências de desempenho.
      d. Restrições de projeto.
      e. Diagrama de apoio.
   3. Descrição do controle.

a. Especificação do controle.
b. Restrições de projeto.

IV. Descrição Comportamental.
1. Estados do sistema.
2. Eventos e ações.

V. Critérios de Validação
1. Limites de desempenho.
2. Classes de testes.
3. Reação esperada do software.
4. Considerações especiais.

VI. Bibliografia

VII. Apêndice.

Onde, cada seção pode ser entendida como sendo:

a) **Introdução:** declara as metas e os objetivos do software, decervendo-o no contexto do sistema baseado em computador.

b) Seção **Descrição da Informação**: apresenta uma descrição detalhada do problema que o software deve resolver.

c) Seção **Descrição Funcional**: apresenta uma descrição de cada função exigida para resolver o problema.

d) Seção **Descrição Comportamental**: examina a operação do software como uma consequencia de eventos externos e características de controle internamente geradas.

e) Seção **Critérios de Validação**: "Como reconhecer uma implementação bem-sucedida?", "Quais classes de testes devem ser levadas a efeito para validar a função, o desempenho e as restrições?". A especificação dos critérios de validação age como uma revisão implícita de todas as demais exigências.

f) **Bibliografia** e **Apêndice**: A bibliografia contém referências a todos os documentos que se relacionam com o software. O apêndice traz informações que complementam a especificação. Dados de tabelas, descrição detalhada de algoritmos, quadros, gráficos e outros materiais são apresentados como apêndice.

4 GERENCIAMENTO DE PROJETOS

O fracasso de muitos projetos de softwares grandes nas décadas de 60 e 70 foi uma indicação 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.

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.

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

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

4.1 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.

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.

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.

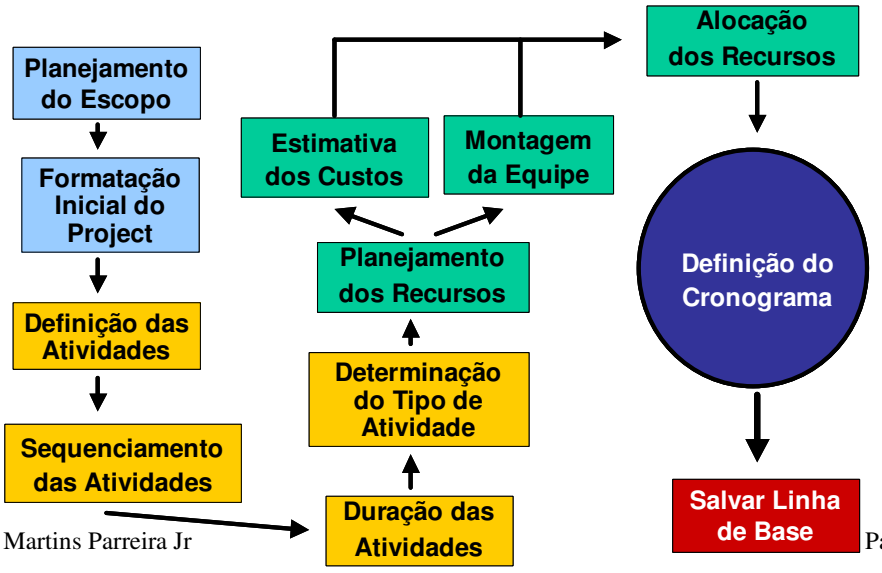
4.2 Atividades de Gerenciamento

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.

4.3 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.



4.4 Especificação de Requisitos de Software

A etapa de levantamento de requisitos é composta por diversas técnicas que visam obter do cliente as informações necessárias para desenvolver o projeto do sistema de informação. Essas técnicas podem ser:

- Entrevistas não estruturadas: Informal ou sem agenda pré-definida;
- Entrevistas estruturadas: Com uma agenda pré-definida;
- Observação do comportamento: Observar os usuários em seu ambiente de trabalho;
- Aprendizagem com o usuário: Analisa e discute com o usuário a maneira como é feito o trabalho;
- Prototipagem: Desenvolvimento de um modelo que simulará o sistema real;
- Brainstorming: Reunião com várias pessoas onde todos discutem um tema central;
- Análise de textos: O usuário descreve as necessidades textualmente. (técnica muito usada atualmente);
- Reutilização de requisitos: Reaproveitamento de padrões ou requisitos de outros sistemas.

Os requisitos podem ser funcionais e não-funcionais.

4.4.1 Requisitos funcionais

Os requisitos funcionais são aqueles que fazem parte do sistema, como um relatório específico, um campo a mais em um cadastro, etc. Ele normalmente tem a finalidade de agregar valor ao usuário ou facilitar o trabalho que ele desenvolve. São exemplos de requisitos funcionais:

- "o software deve possibilitar o cálculo dos gastos diários, semanais, mensais e anuais com pessoal".
- "o software deve emitir relatórios de compras a cada quinze dias"
- "os usuários devem poder obter o número de aprovações, reprovações e trancamentos em todas as disciplinas por um determinado período de tempo.

A especificação de um requisito funcional deve determinar o que se espera que o software faça, sem a preocupação de como ele faz. É importante diferenciar a atividade de especificar requisitos da atividade de especificação que ocorre durante o design do software. No design do software deve-se tomar a decisão de quais a funções o sistema efetivamente terá para satisfazer àquilo que os usuários querem.

4.4.2 Requisitos não-funcionais

Requisitos não-funcionais são aqueles relacionados ao ambiente onde o sistema está inserido. Um servidor mais robusto, um firewall, ou um usuário especializado em determinado procedimento pode ser visto como requisitos não-funcionais. São exemplos de requisitos não-funcionais:

- "a base de dados deve ser protegida para acesso apenas de usuários autorizados".
- "o tempo de resposta do sistema não deve ultrapassar 30 segundo".
- "o software deve ser operacionalizado no sistema Linux"
- "o tempo de desenvolvimento não deve ultrapassar seis meses".

A necessidade de se estabelecer os requisitos de maneira de forma precisa é crítica na medida em que o tamanho e a complexidade do software aumentam. Os requisitos exercem influência uns sobre os outros. Por exemplo, o requisito de que o software de ter grande portabilidade (por exemplo, ser implementado em Java) pode implicar em que o requisito desempenho não seja satisfeito (programas em Java são, em geral, mais lentos).

Os requisitos podem ser classificados também pelo seu tipo:

- Requisitos operacionais;
- Requisitos de segurança;
- Requisitos de desempenho;
- Especificações de Hardware e software.
- Entre outros.

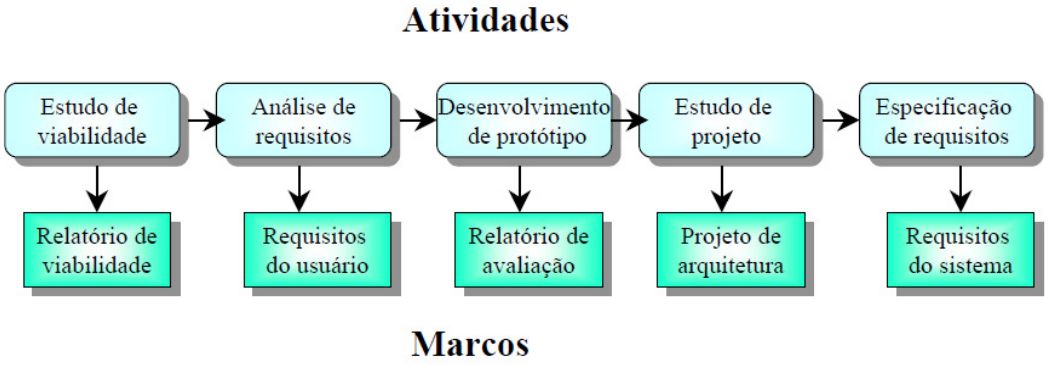
4.5 Organização de atividades

As atividades em um projeto devem ser organizadas de modo a produzir saídas tangíveis para os gerentes julgar o progresso.

4.5.1 Marco de Projeto

Marco de projeto é a conclusão de uma etapa do projeto, o ponto-final de uma atividade de processo de software.

O processo com prototipação, por exemplo, permite uma definição direta de marcos de progresso.

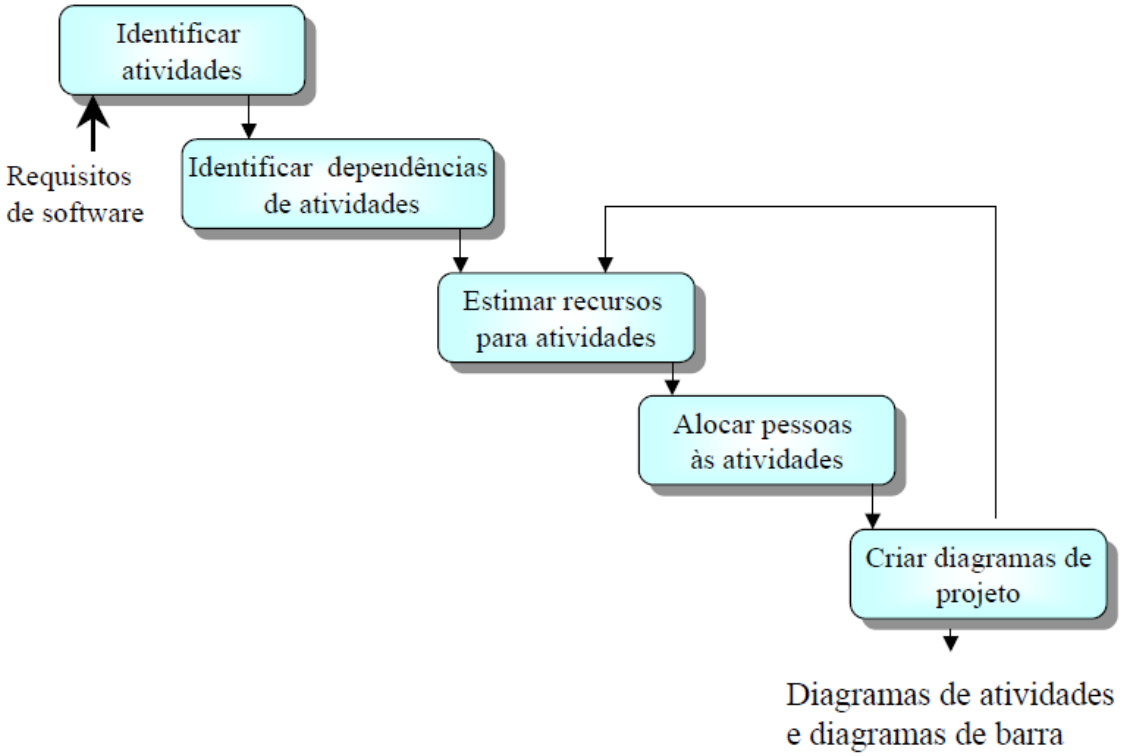


Um marco de projeto é o resultado previsto de uma atividade em que algum relatório formal de progresso deve ser apresentado à gerência.

4.5.2 Programação de projeto

Envolve a divisão do trabalho total de um projeto em atividades distintas e avaliação do tempo necessário para completar cada uma destas atividades.

Coordenar as atividades paralelas de modo que a força de trabalho seja otimizada. Minimizar dependências entre tarefas para evitar atrasos causados por uma tarefa ter que esperar que outra seja encerrada. Depende da intuição e experiência do gerente de projeto.



4.5.3 Diagramas de barras e redes de atividades

Notações gráficas utilizadas para ilustrar a programação de projeto. Apresentam a divisão do projeto em atividades. As atividades não devem ser muito pequenas. Devem durar pelo menos uma semana, a não em situações bem específicas.

As redes de atividades mostram dependências entre atividades e o caminho crítico.

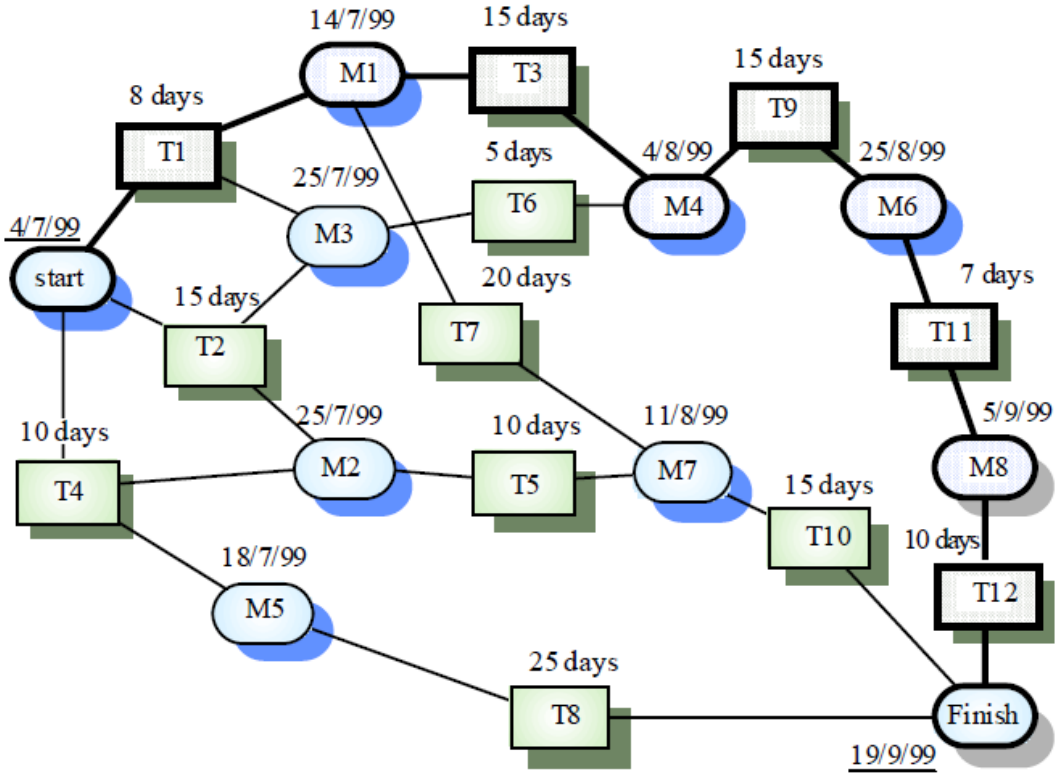
Os diagramas de barra mostram para quando está programado o início e término da atividade, bem como os responsáveis por cada atividade.

Um exemplo de programação de um projeto, apresentando as atividades (T1 a T12), o tempo estimado para cada atividade (em dias), as dependências para iniciar as atividades e os seus respectivos marcos (M1 a M8).

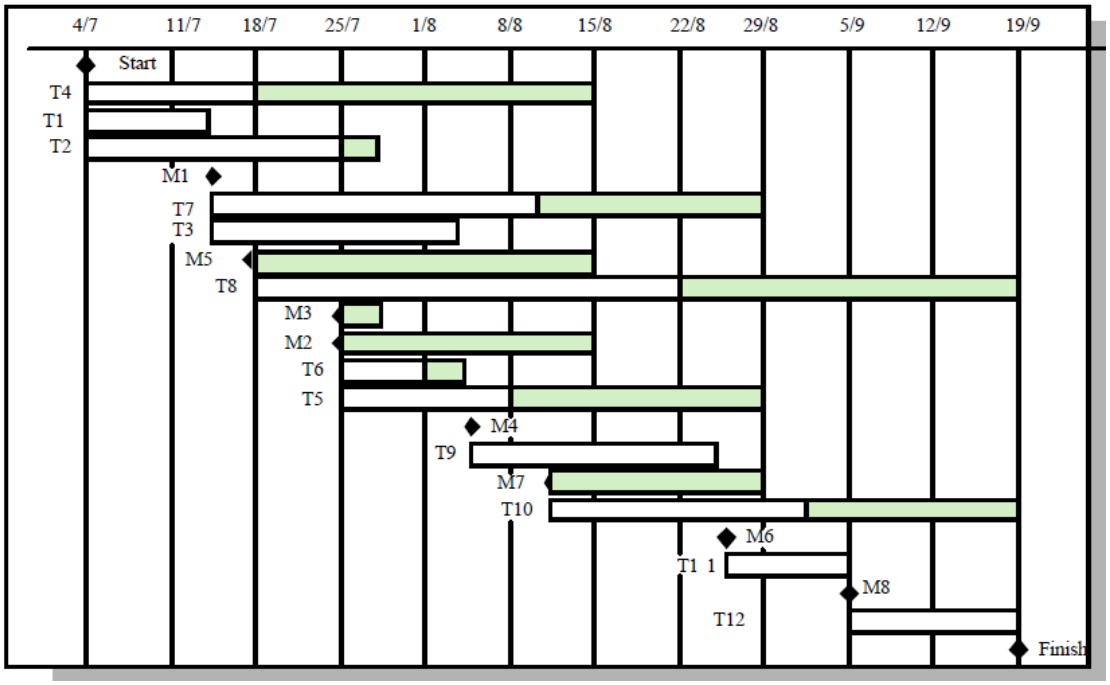
A partir do quadro de atividades é possível desenvolver os diagramas.

Atividade	Duração (dias)	Dependências
T1	8	
T2	15	
T3	15	T1 (M1)
T4	10	
T5	10	T2, T4 (M2)
T6	5	T1, T2 (M3)
T7	20	T1 (M1)
T8	25	T4 (M5)
T9	15	T3, T6 (M4)
T10	15	T5, T7 (M7)
T11	7	T9 (M6)
T12	10	T11 (M8)

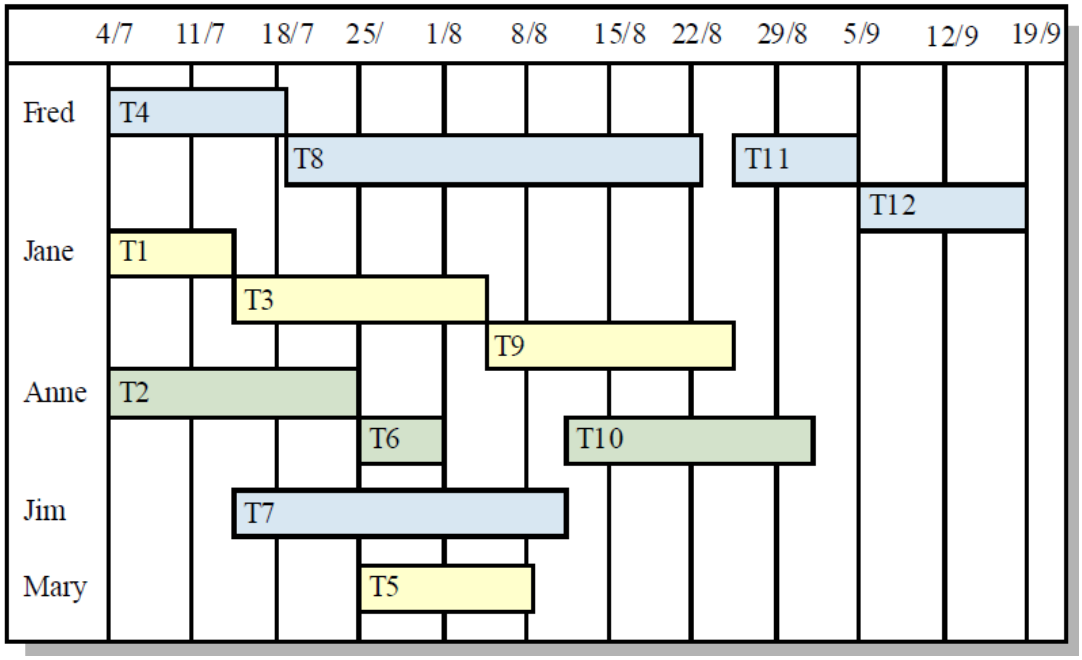
A Rede de Atividades desenvolvida a partir do Quadro de atividades, observando os Marcos e o Caminho Principal (caminho em negrito).



O Diagrama de Barra de atividades, considerando as datas iniciais de cada atividade, os Marcos definidos e a Margem de atraso para cada atividade (quando isto é possível) em função das dependências das atividades anteriores.



O Diagrama de alocação de pessoas no projeto:



Os principais riscos de projeto devem ser identificados, avaliados e monitorados, de forma a elaborar planos preventivos, gerenciamento de riscos, ou planos para administrar os riscos.

4.6 PMBOK -Project Management Body of Knowledge

Um dos principais difusores do gerenciamento de projetos e da profissionalização do gerente de projetos é o Instituto de Gerenciamento de Projetos (PMI - Project Management Institute). Fundado nos Estados Unidos em 1969, o PMI é uma associação profissional mundialmente difundida, atualmente com meio milhão de membros em mais de 180 países. O PMI é distribuído geograficamente pelo mundo em Capítulos.

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.

Editado na forma de livro, o Guia PMBOK está atualmente na quarta edição de 2008 e traduzido oficialmente para diversos idiomas, inclusive o português do Brasil. As edições anteriores foram publicadas nos anos de 1996, 2000 e 2004.

O PMBOK formaliza diversos conceitos em gerenciamento de projetos, como a própria definição de projeto e do seu ciclo de vida. Também identifica na comunidade de gerenciamento de projetos um conjunto de conhecimentos amplamente reconhecido como boa prática, aplicáveis à maioria dos projetos na maior parte do tempo. Estes conhecimentos estão categorizados em nove áreas e os processos relacionados são organizados em cinco grupos ao longo do ciclo de vida do projeto.

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.

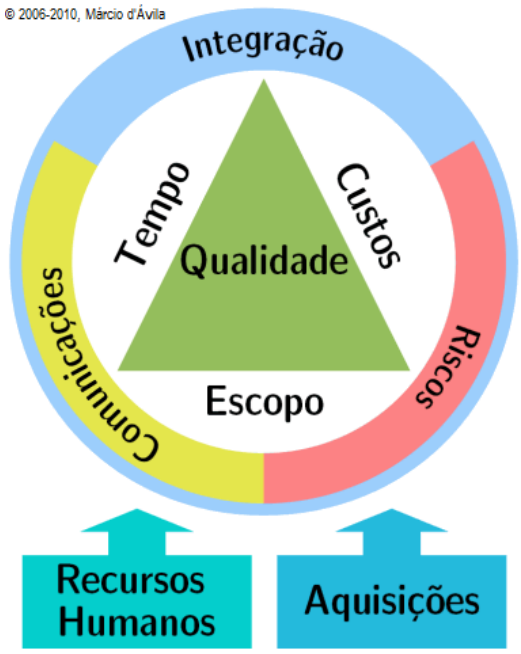
4.6.1 Projeto e seu gerenciamento

Um projeto é um esforço temporário empreendido para criar um produto, serviço ou resultado exclusivo. Temporário não significa necessariamente de curta duração, mas sim que um projeto possui um início e um término definidos. Isso distingue o projeto dos trabalhos operacionais de natureza contínua. E exclusivo indica a singularidade da natureza de cada projeto, pois mesmo que elementos repetitivos ou similares possam estar presentes em algumas entregas do projeto, o resultado de cada projeto é obtido sob uma combinação exclusiva de objetivos, circunstâncias, condições, contextos, fornecedores etc.

O gerenciamento de projetos consiste na aplicação de conhecimentos, habilidades, ferramentas e técnicas adequadas às atividades do projeto, a fim de atender aos seus requisitos.

São definidas nove áreas de conhecimento caracterizam os principais aspectos envolvidos em um projeto e no seu gerenciamento:

- Integração
- Escopo
- Tempo
- Custos
- Qualidade
- Recursos humanos
- Comunicações
- Riscos
- Aquisições



Escopo, Tempo, Custos e Qualidade são os principais determinantes para o objetivo de um projeto: entregar um resultado de acordo com o escopo, no prazo e no custo definidos, com qualidade adequada; em outras palavras, o que, quando, quanto e como. Recursos Humanos e Aquisições são os insumos para produzir o trabalho do projeto. Comunicações e Riscos devem ser continuamente abordados para manter as expectativas e as incertezas sob controle, assim como o projeto no rumo certo. E Integração abrange a orquestração de todos estes aspectos.

Um projeto consiste nisso: pessoas (e máquinas) que utilizam tempo, materiais e dinheiro realizando trabalho coordenado para atingir determinado objetivo.

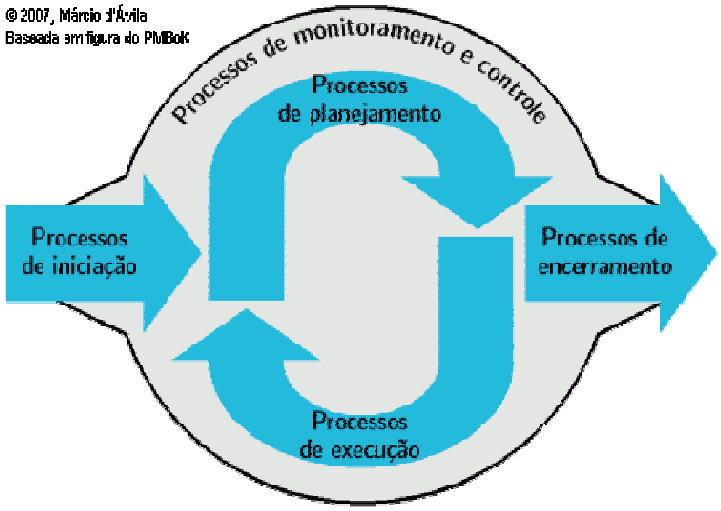
4.6.2 Processos do gerenciamento de projetos

A aplicação dos conhecimentos requer a adoção eficaz de processos apropriados. Cada área de conhecimento abrange diversos processos no gerenciamento de projetos.

Um processo é um conjunto de ações e atividades interrelacionadas que são executadas para alcançar um objetivo. Cada processo é caracterizado por suas entradas, as ferramentas e as técnicas que podem ser aplicadas, e as saídas resultantes.

Os cinco grupos de processos de gerenciamento de projetos são:

- Iniciação
- Planejamento
- Execução
- Monitoramento e Controle
- Encerramento



Além de conceituar os aspectos fundamentais do gerenciamento de projetos, de forma a promover um vocabulário comum dentro dessa profissão, o Guia PMBOK documenta (define e descreve) processos de gerenciamento de projetos e os apresenta didaticamente, organizados em um capítulo por área de conhecimento. Em cada processo, são abordadas suas entradas e saídas, suas características, bem como os artefatos, técnicas e ferramentas envolvidas.

O diagrama com fluxo proposto por Mauro Sotille, relaciona de forma gráfica e sintética todos os 42 processos de gerenciamento de um projeto descritos no PMBOK 4ª Edição, indicando também os cinco grupos em que os processos se distribuem e as respectivas áreas de conhecimento associadas a cada um dos processos.

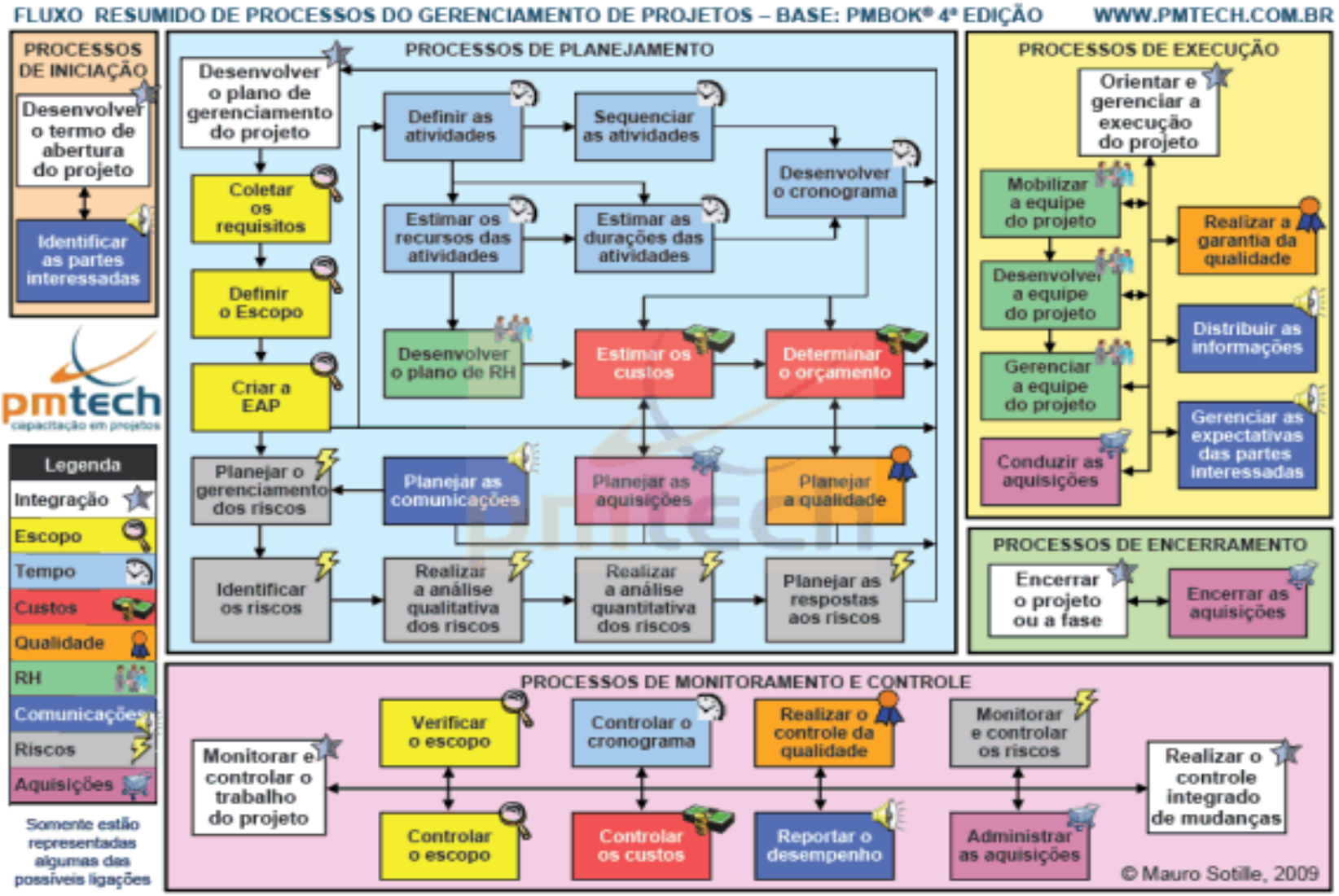
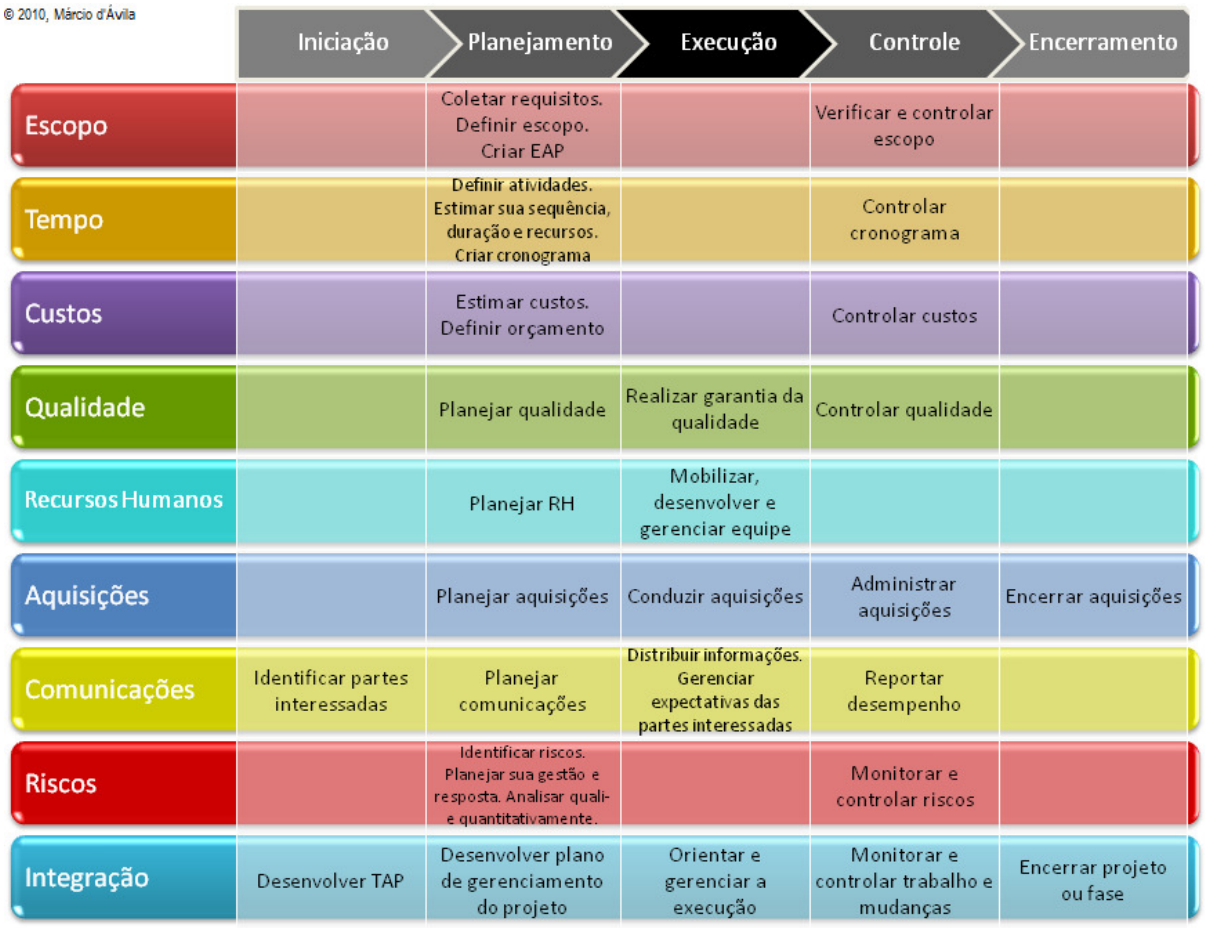


Diagrama com fluxo proposto por Mauro Sotille, disponível na seção de Artigos sobre Gerenciamento de Projetos do portal da empresa PM Tech.

Para estes mesmos 42 processos de gerenciamento de projetos do PMBOK, a matriz a seguir provê uma visão quantitativa de sua distribuição pelas áreas de conhecimento e pelos grupos de processos..



Pelo diagrama é fácil perceber algumas características lógicas dos processos de gerenciamento de um projeto:

- Praticamente todas as áreas de conhecimento são abordadas nas atividades de Planejamento (definir, estimar e planejar cada aspecto) e de Monitoramento e Controle (controlar) — no PMBOK 4ª edição, o processo de Gerenciar a equipe passou ao grupo de Execução, deixando apenas a área de RH sem processos no grupo de Controle;
- quanto a Execução, os aspectos envolvidos mais ativamente são a equipe (RH), as comunicações, as aquisições, e a garantia da qualidade;
- a integração se faz presente em todos os momentos do projeto.
- na figura com as descrições, os grupos de processos representam os tipos de atividades, as áreas de conhecimento caracterizam os assuntos, e seu cruzamento induz, de forma bastante intuitiva, os respectivos verbos — definir, planejar, estimar, gerenciar, monitorar, controlar, encerrar etc. — e substantivos que descrevem os processos de gerenciamento relacionados.

Isso mostra que os conceitos e melhores práticas que o PMBOK reúne, organiza e formaliza estão naturalmente presentes na essência do gerenciamento de qualquer bom projeto.

4.6.3 Gerencia do projeto

O gerente do projeto é a pessoa designada pela organização responsável pela condução do projeto, com a missão de zelar para que os objetivos do projeto sejam atingidos. O gerente de projetos tem sido caracterizado por um perfil profissional com domínio e experiência especializados nos processos e nas áreas de conhecimento do gerenciamento de projetos. O trabalho do gerente de um projeto pode ser sintetizado em dois grandes elementos:

- Planejar (antes) e Controlar (durante) as atividades do projeto e seu gerenciamento, conforme se pode constatar pela concentração de processos de gerenciamento de um projeto abrangendo todas os aspectos envolvidos.
- Comunicar: os gerentes de projetos passam a maior parte do seu tempo se comunicando com os membros da equipe e outras partes interessadas do projeto.

Além disso, os gerentes de projetos devem dominar diversas habilidades interpessoais que utilizam com frequencia, dentre as quais se pode destacar:

- Comprometimento, responsabilidade, ética e honestidade;
- Transparência, franqueza, clareza e objetividade;
- Liderança, agregação, motivação e entusiasmo;
- Solução de conflitos e problemas;
- Negociação, influência e persuasão;
- Decisão, iniciativa e proatividade;
- Organização e disciplina;
- Autocontrole, equilíbrio e resiliência;
- Empreendedorismo;
- Eficácia.

O PMI mantém um Código de Ética e Conduta Profissional (*Project Management Institute Code of Ethics and Professional Conduct*), criado para incutir confiança à profissão de gerenciamento de projetos e auxiliar os praticantes a se tornarem melhores profissionais. Para isso, o código descreve as expectativas que os profissionais de gerenciamento de projetos tem de si e de seus colegas. Ele exige que os profissionais demonstrem compromisso com a conduta ética e profissional, sendo específico quanto à obrigação básica de responsabilidade, respeito, justiça e honestidade. Isso inclui respeitar as leis, regulamentos e políticas organizacionais e profissionais.

Mais que ser um facilitador, o gerente de projetos deve fazer a diferença no bom andamento e no sucesso dos projetos.

4.6.4 Partes interessadas no projeto

Partes interessadas, intervenientes ou do termo em inglês (stakeholders) são todas as pessoas e organizações envolvidas no projeto, ou cujos interesses podem ser positiva ou

negativamente afetados pela realização ou pelos resultados do projeto. As partes interessadas também podem exercer influência sobre o projeto e sobre os membros da equipe do projeto.

Desde a iniciação do projeto, a equipe de gerenciamento precisa identificar as partes interessadas internas e externas. Ao longo do planejamento e da execução do projeto, o gerente do projeto e sua equipe devem gerenciar as diferentes necessidades, preocupações e expectativas das partes interessadas, bem como a influência destas no projeto, para garantir um resultado bem sucedido. Alguns exemplos de possíveis partes interessadas podem incluir:

- Patrocinador (Sponsor): pessoa ou grupo que fornece os recursos financeiros para a realização do projeto, e que também provê o aval estratégico e político que viabiliza e promove o projeto e o defende;
- A equipe do projeto, que inclui o gerente do projeto, a equipe de gerenciamento do projeto, e outros membros da equipe que executam trabalho no projeto mas não estão necessariamente envolvidos com o gerenciamento;
- Clientes e usuários;
- Presidente, donos e executivos;
- Acionistas e investidores;
- Gerentes funcionais;
- Escritório de projetos (Project Management Office - PMO), gerentes e comitês de portfólios e de programas;
- Fornecedores e parceiros comerciais;
- Concorrentes;
- Governo, em suas diversas esferas e poderes;
- Organismos de regulação e fiscalização internos e externos, incluindo auditorias, agências, conselhos, sindicatos e associações institucionais, profissionais e oficiais;
- Organizações não governamentais (ONG);
- Comunidades, vizinhança e população abrangida pelas ações e resultados do projeto.

Outros elementos importantes que influenciam projetos são as culturas e estilos organizacionais, bem como os fatores ambientais da empresa, do mercado, da sociedade e da localização geopolítica onde o projeto acontece.

5 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.

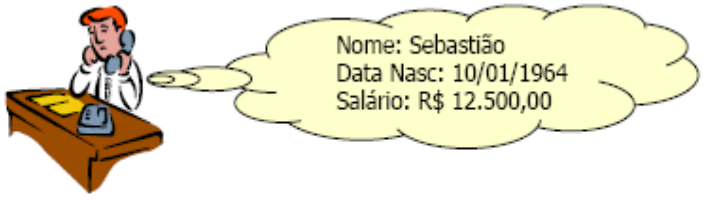
A UML define uma notação e um meta-modelo. A notação representa a sintaxe da linguagem composta por elementos gráficos. Um meta-modelo é um diagrama de classe. A UML foi desenvolvida por Grady Booch, James Rumbaugh, e Ivar Jacobson que são conhecidos como "os três amigos". A UML é a junção do que havia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

5.1 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)

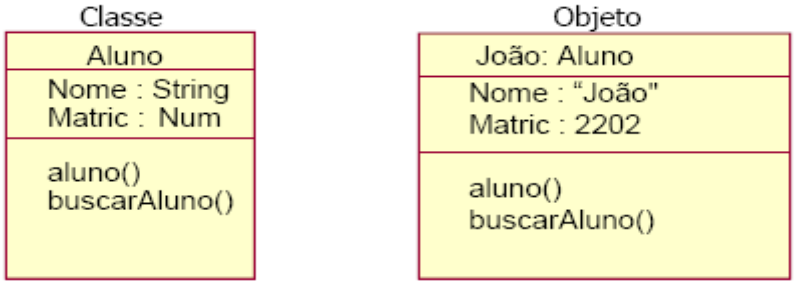
a) Classe x Objeto

Exemplo Classe x Objeto

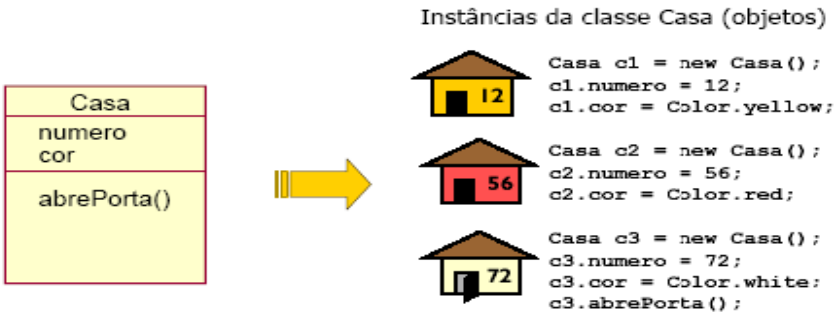
Para criar efetivamente um objeto, nós devemos instanciá-lo:

```
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

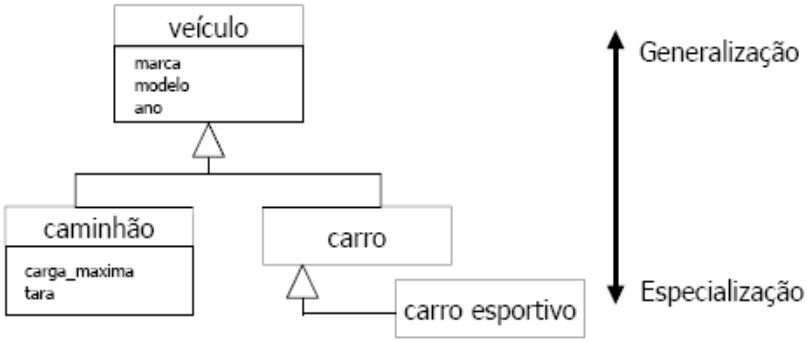


Outro Exemplo Classe x Objeto



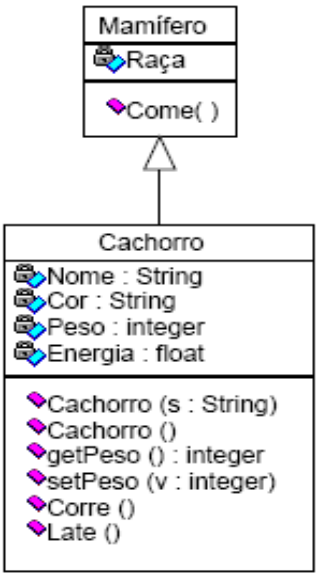
A **classe** casa é como uma forma que irá criar objetos semelhantes. Observe que cada objeto tem característica própria, a cor o número.

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.

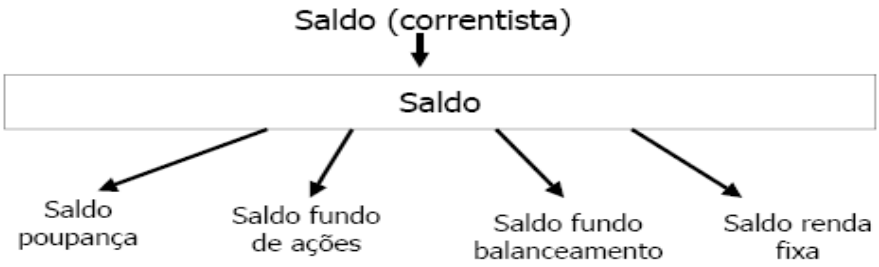


- **Herança - continuação**

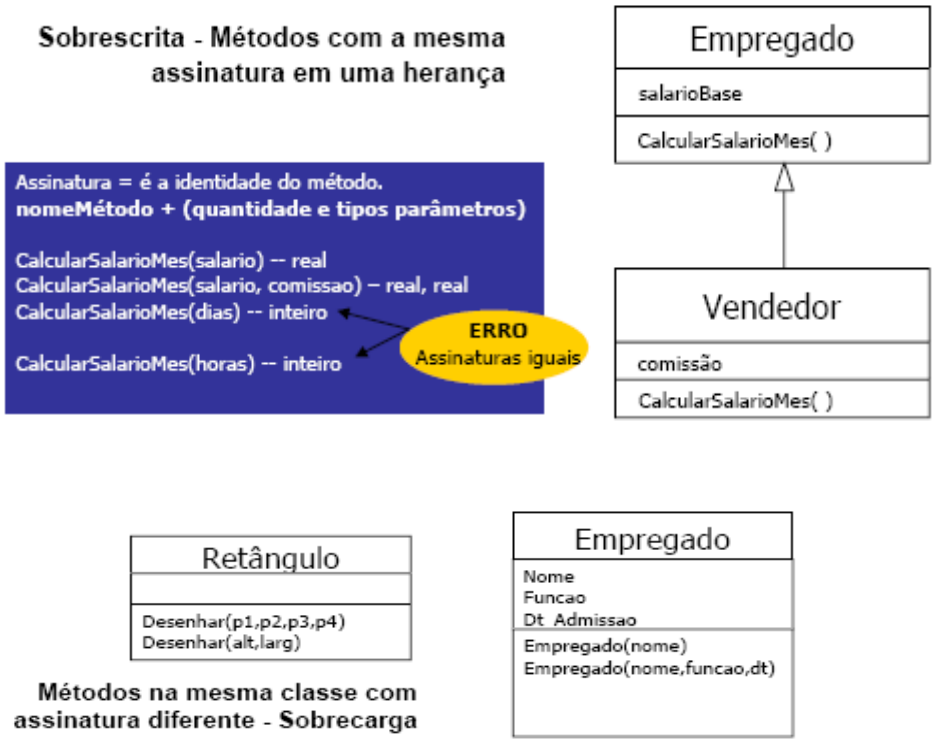
```
class Cachorro extends Mamífero {
    // Atributos dos objetos da classe
    private String nome;
    private String cor;
    private int    peso;
    private float  energia;

    // 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.



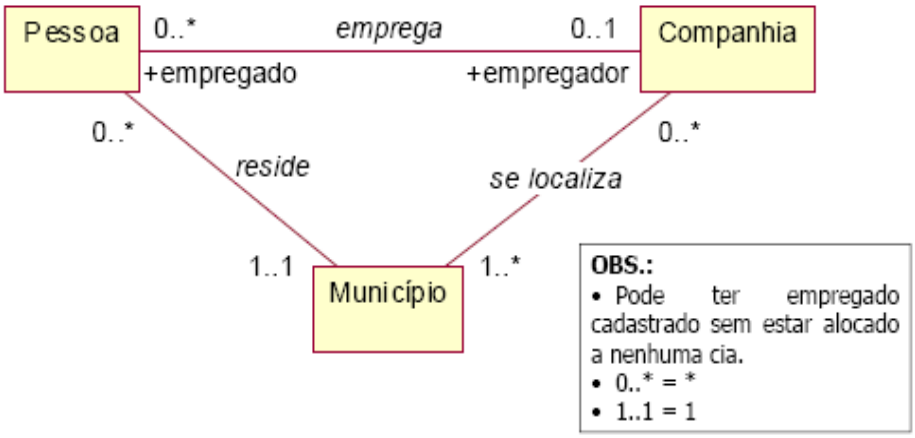
Mensagem - É a comunicação entre objetos. A execução de um método é ativada através do envio de uma mensagem.

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.

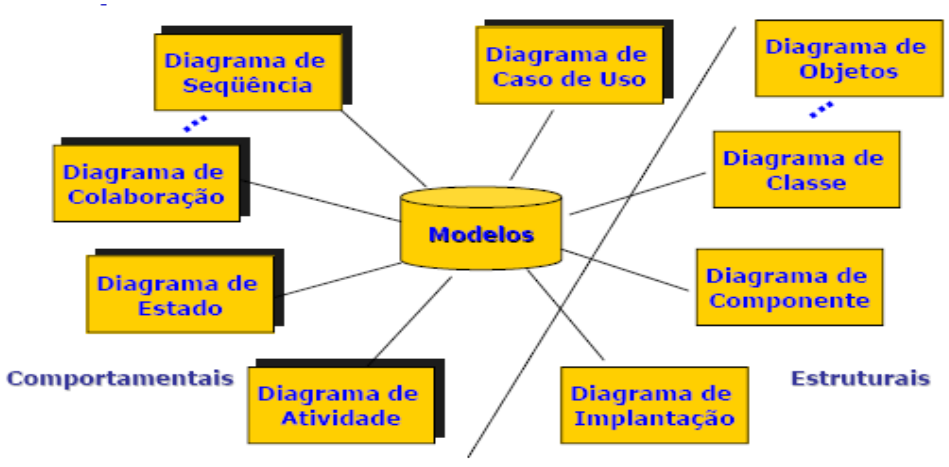
Casos de Uso – diagrama comportamental que mostra uma interação, dando ênfase à organização estrutural de objetos que enviam e recebem mensagens.

Interação (Sequência 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 pode-se 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:

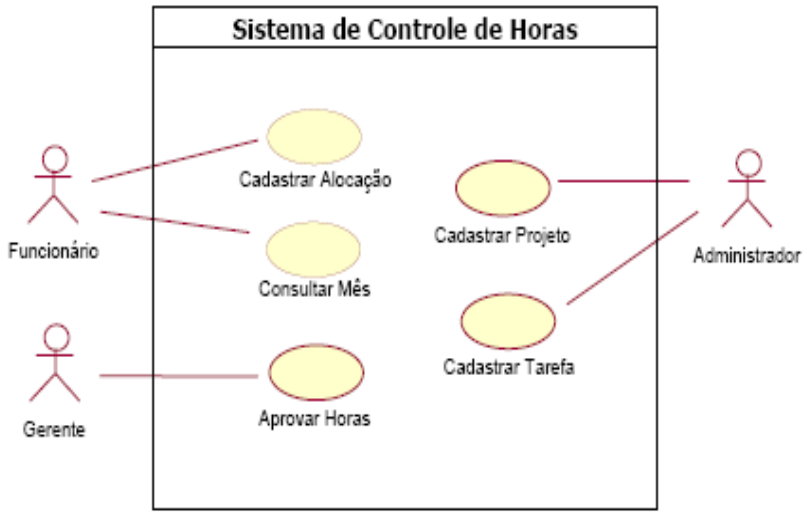


Diagrama de Caso de Uso: Alguns casos de uso que podem ser aplicados ao sistema de controle de horas são (este é um exemplo os casos de uso não estão levantados na totalidade do sistema):

- 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.

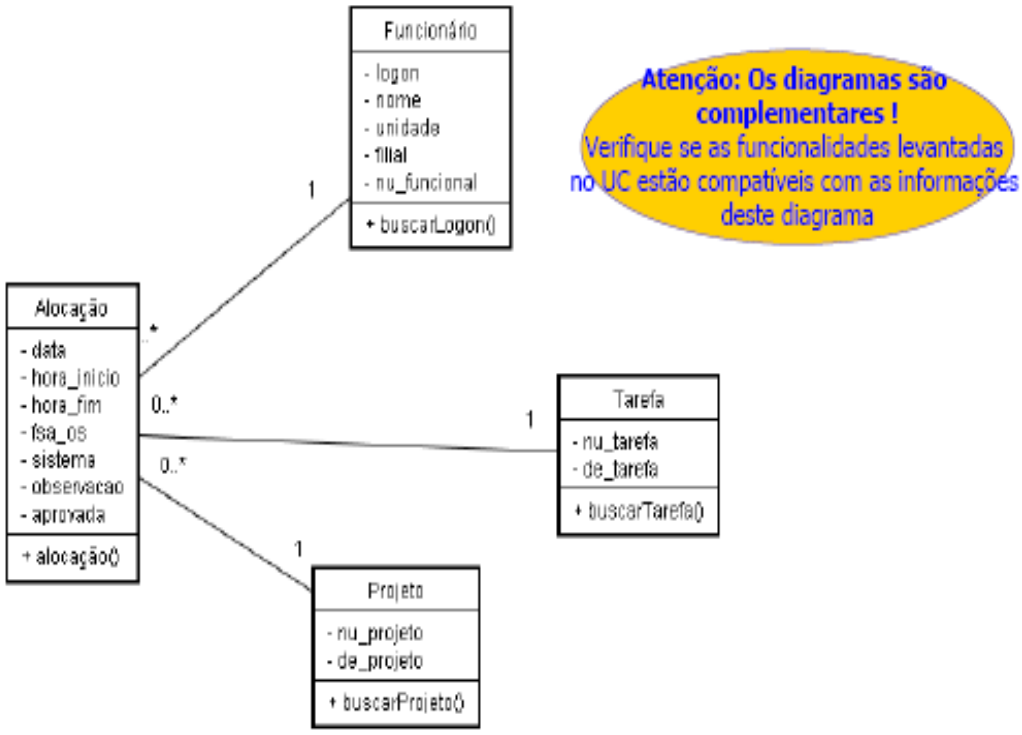


Diagrama de Sequência - Mostra uma interação organizada em forma de uma sequência, dentro de um determinado período de tempo. Os participantes são apresentados dentro do contexto das mensagens que transitam entre eles. O diagrama de sequência é um diagrama de interação.

Um Diagrama de Sequência oferece uma visão detalhada de um caso de uso. Ele mostra uma interação organizada em forma de uma sequência, 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 Sequência pode ser bastante detalhado, e pode incluir milhares de mensagens.

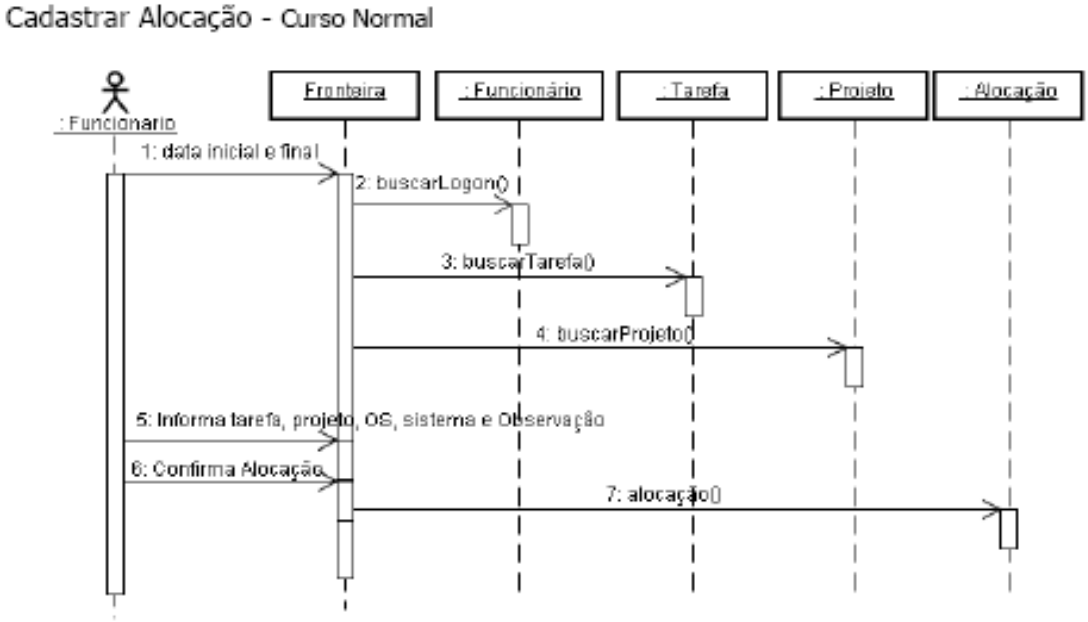
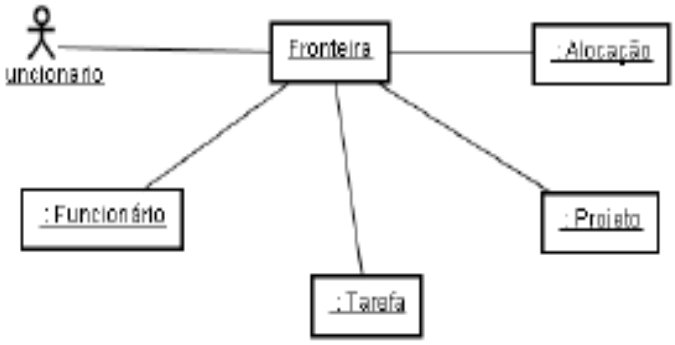


Diagrama de Colaboração - Mostra como um grupo de objetos num caso de uso interage com os demais. Cada mensagem é numerada para documentar a ordem na qual ela ocorre. O diagrama de colaboração também é um diagrama de interação.

Um Diagrama de Colaboração é outro tipo de diagrama de interação. Assim como no Diagrama de Sequência, 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 sequência são equivalentes a diferença é que o diagrama de sequência tem o tempo como participante fundamental.



O diagrama da figura é equivalente ao diagrama de sequência anterior. Podemos dizer que o diagrama de colaboração e o diagrama de sequência são isomórficos, ou seja, podemos obter uma visão a partir da outra. Enquanto o diagrama de sequência 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.

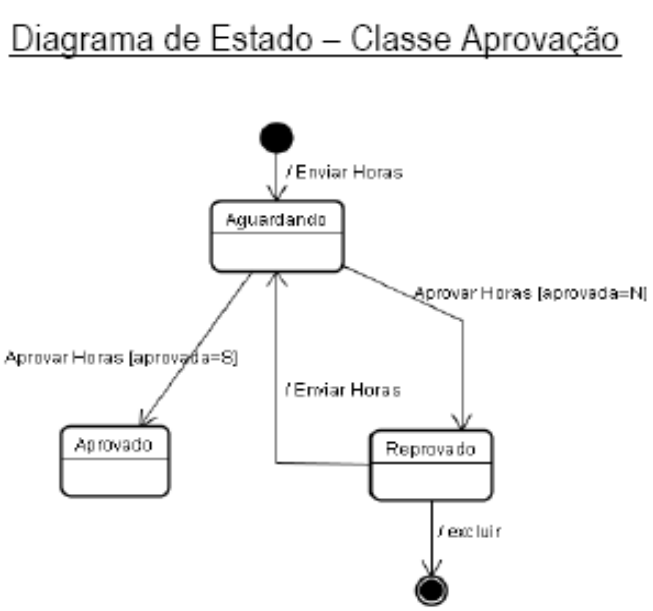


Diagrama de Atividade - Descreve a sequência de atividades, com suporte para comportamento condicional e paralelo. Componentes:

- 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.

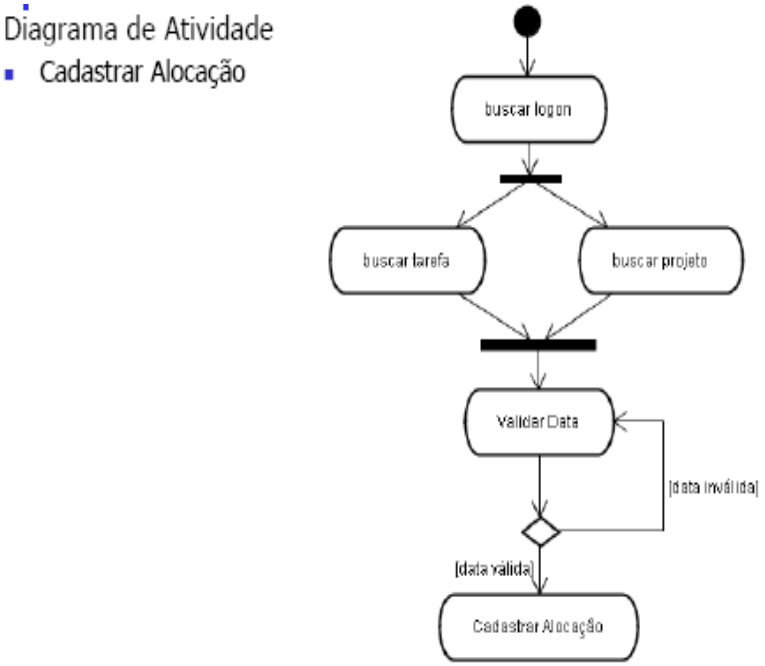
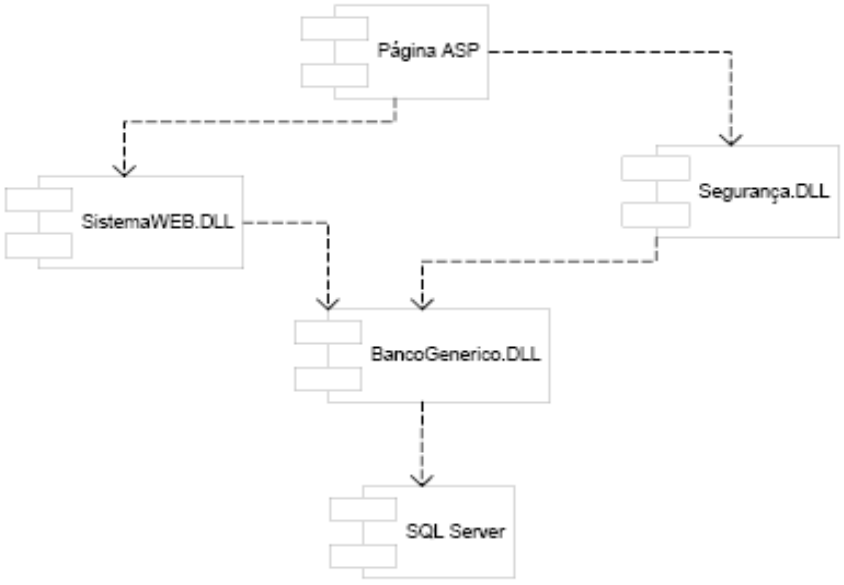


Diagrama de Componentes - Mostra como os diferentes subsistemas de software formam a estrutura total de um sistema.

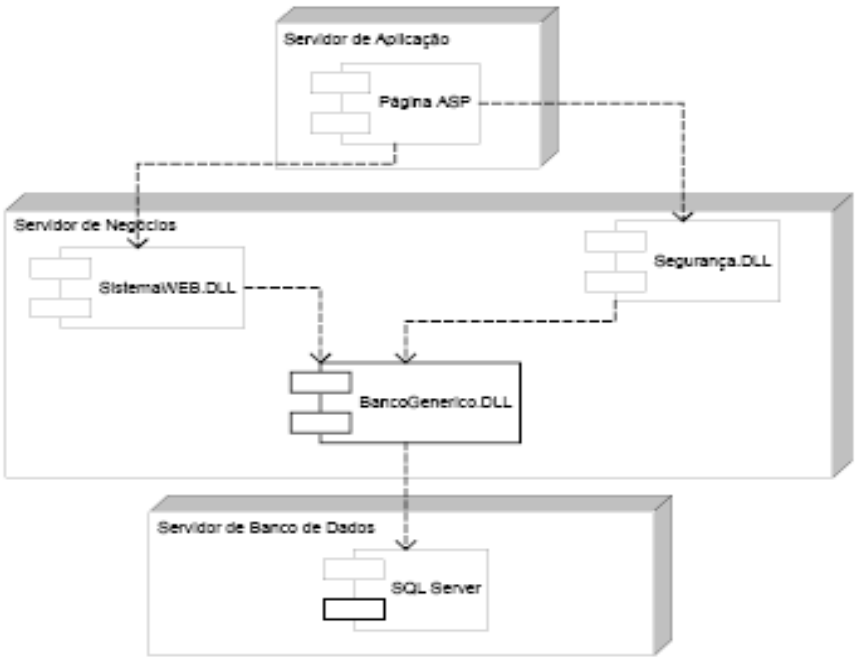
Um Diagrama de Componentes exibe a hierarquia de dependência entre os componentes do sistema. Um componente pode ser entendido como uma tela do sistema, uma biblioteca (dlls), um arquivo de críticas (arquivos js), um executável, etc.



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.

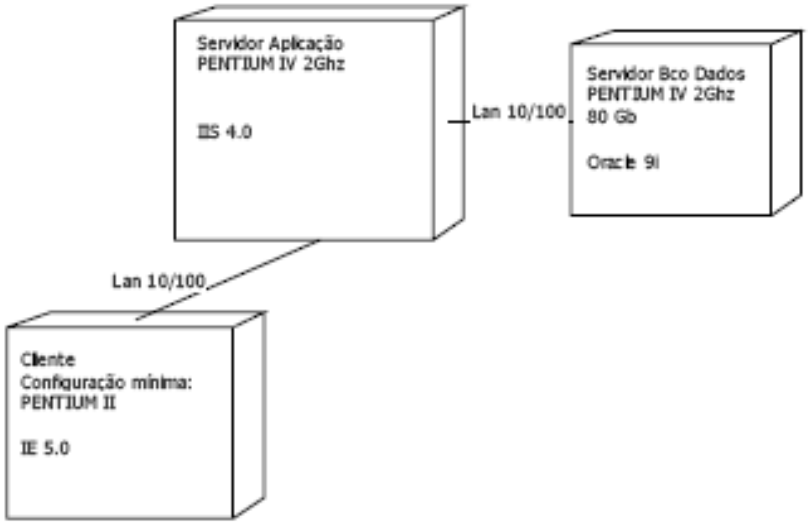
O Diagrama de Implantação mostra o layout físico da rede, onde cada nó representa uma máquina, podendo mostrar a configuração delas (hardware e software) e também onde cada componente irá ser instalado.



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.



5.2 Casos de Uso

Caso de Uso é: "Um conjunto de sequências 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.

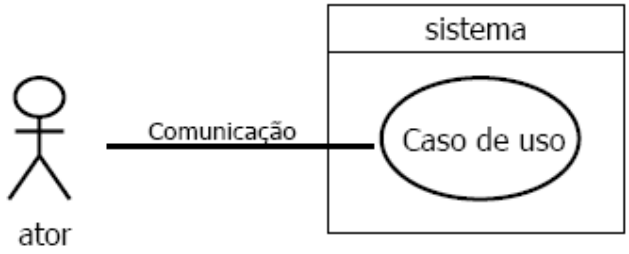
Requisito é uma característica do sistema ou a descrição de algo que o sistema é capaz de realizar para atingir seus objetivos. O requisito pode ser de dois tipos:

- **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.

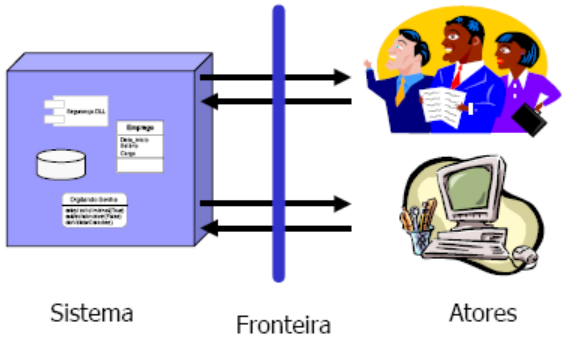
Cenário – descreve a interação entre o ator e o sistema. Um caso de uso pode ter várias terminações (sucessos e insucessos) cada enredo, cada instância é chamada de cenário.

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 sequência 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 sequência 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.

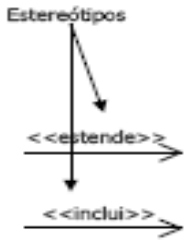
Sempre devolve um valor como resposta. Um caso de uso tem início, meio e fim. É completo, não terá terminado até que um valor tenha sido retornado;

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...

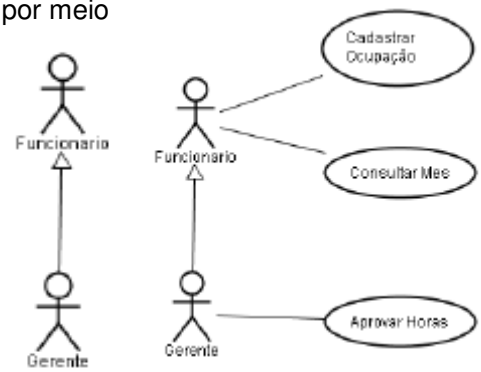
Tipos de Interações/Comunicações usadas (desde a versão 1.3):



- 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).

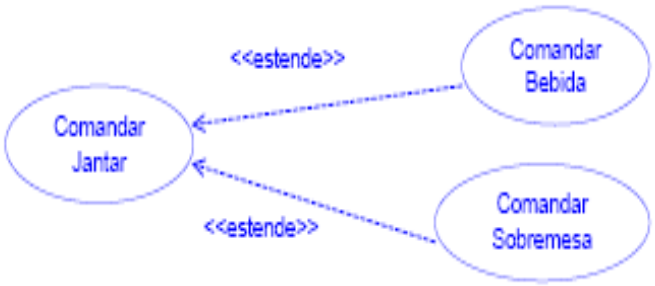
Exemplo: O desenho informa que o cadastro do cliente pode ser chamado diretamente da solicitação do serviço, mas esta é uma ação que pode ocorrer ou não. Observe que o caso de uso que estende tem uma relação de dependência com o caso de uso estendido (seta tracejada), ou seja, a o Cadastro do Cliente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

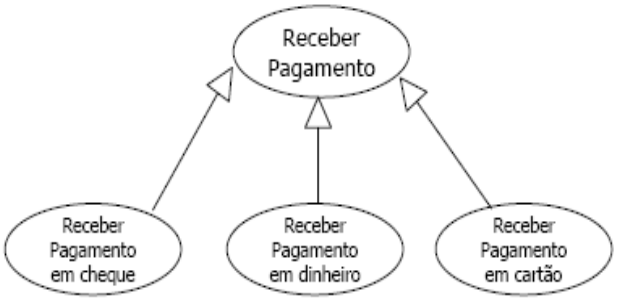
Na figura percebe-se que na funcionalidade Comandar Jantar pode-se ter um atalho para Comandar Bebida ou Comandar Sobremesa. Esta técnica é aplicada para facilitar o uso do sistema. Atenção o uso dos casos de uso estendidos são facultativos, ou seja, pode haver o pedido de jantar sem bebida e sem sobremesa.



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 sequências adicionais de ações em pontos arbitrários da sequência 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 frequência.



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 sequências 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.

5.2.1 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:

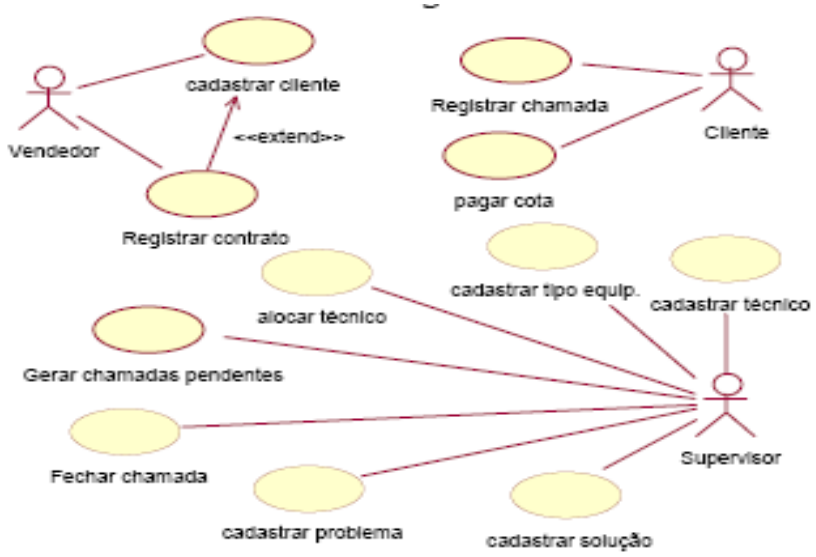
Nome : **Registrar contrato**
Ator : Vendedor
descrição : Este use case é responsável pelo cadastramento de novos contratos no sistema

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:

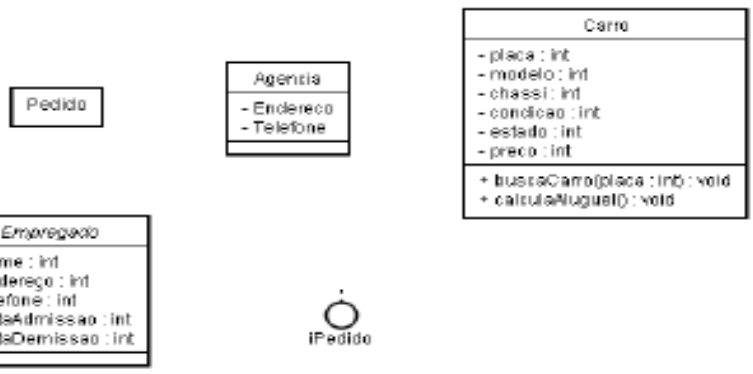
- Evitar na descrição, o caso de uso sem fim, aquele que está sempre retornando a algum passo (loop);
- Procurar sempre iniciar um caso de uso com uma ação do Ator;

- 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 ideia 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.

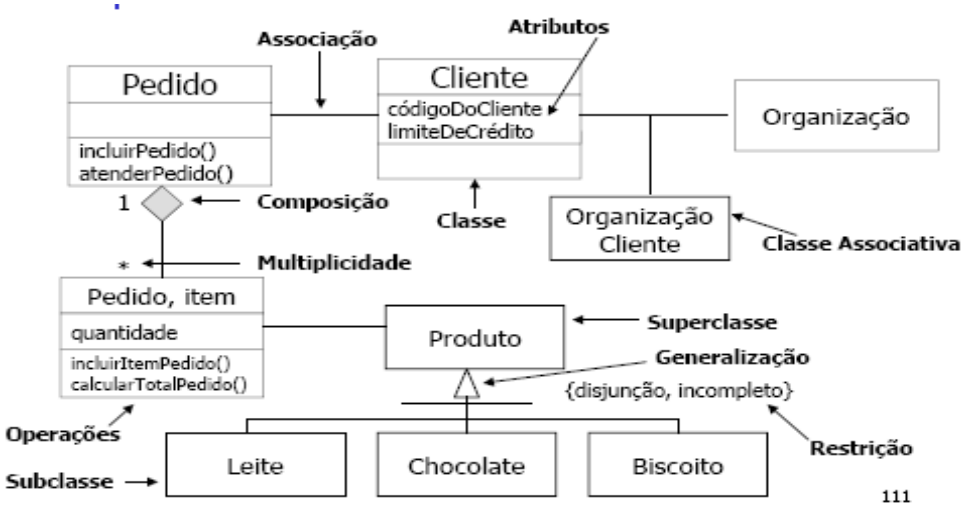
5.3 Diagrama de Classe

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:

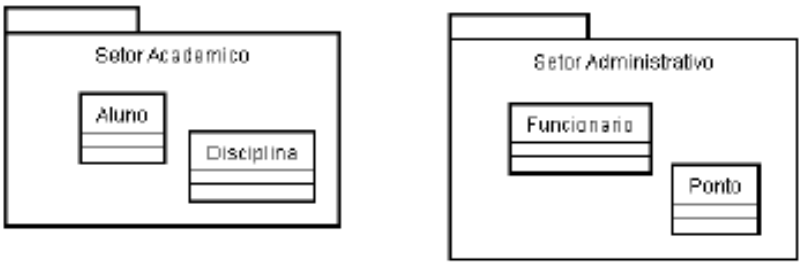


5.3.1 Pacotes

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.

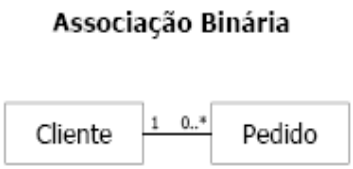


5.3.2 Associação

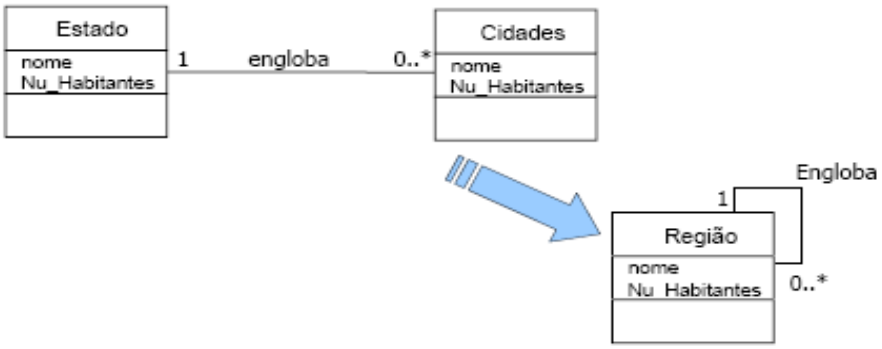
Associação é o relacionamento entre as classes, estabelece um vínculo entre objetos. Tipos de associação:

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

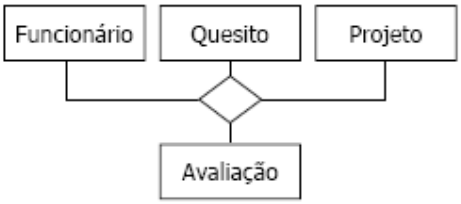
a) A associação **binária** é a mais comum nos diagramas de classe.



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



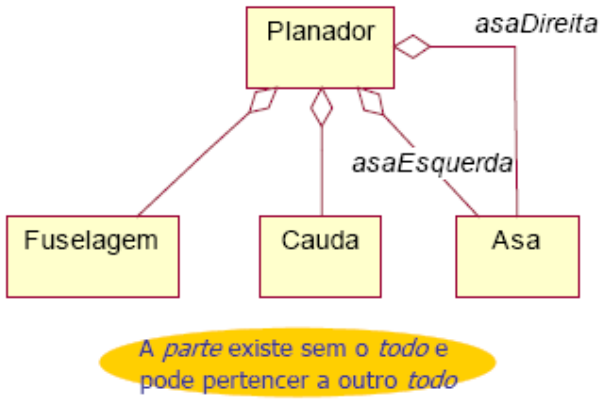
c) Associação **n-ária** - Representa o relacionamento com mais de duas classes.



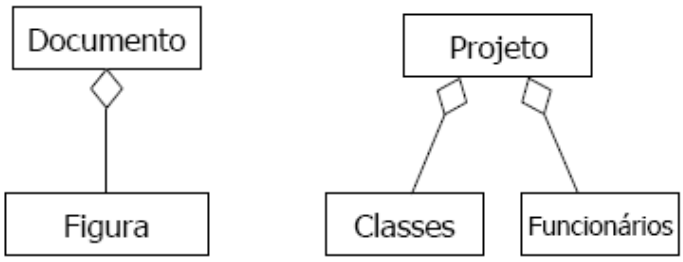
O exemplo acima está exibindo o relacionamento de avaliação de um funcionário em um projeto, e esta avaliação é para um quesito. Então, por exemplo, o funcionário João será avaliado em pontualidade no projeto A e terá uma nota de avaliação, quando ele for trabalhar no Projeto B ele poderá ter outra nota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

5.3.3 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.

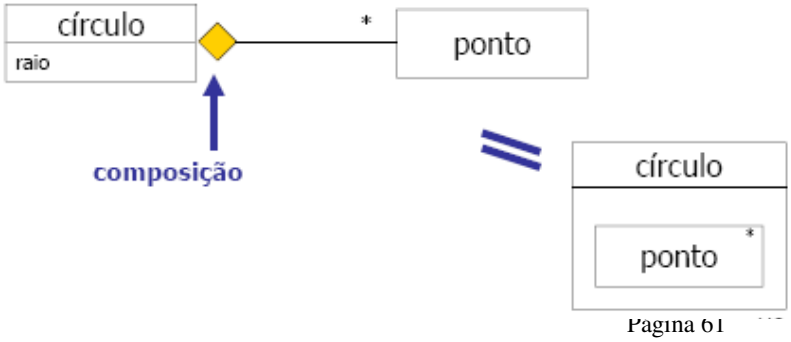


As figuras mostram dois exemplos de agregação: um documento pode ter figuras e parágrafos. Este serão membros do documento (todo-parte). Da mesma forma um projeto pode trabalhar com classes e funcionários, que poderão ser utilizados por outro projeto, significa que ao chegar ao fim de um projeto, as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



5.3.4 Composição

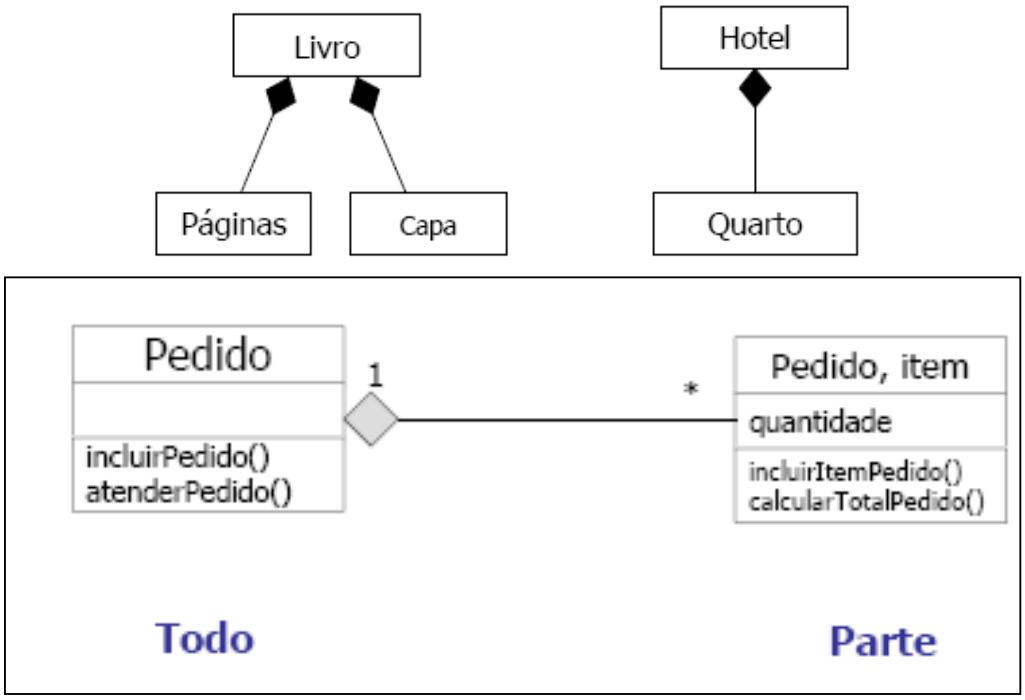
É uma variação da agregação, mas sendo que o objeto parte só pertence a um todo, e são extremamente dependentes do todo, podemos considerar que qualquer deleção do todo gera um efeito cascata nas partes.



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.

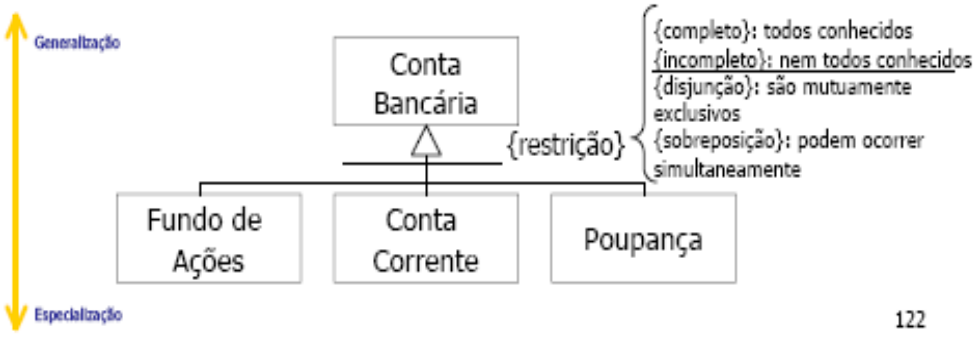
A composição impõe a cardinalidade de no mínimo 1. Os exemplos sugerem que as páginas e a capa do livro compõem um livro e não podem compor outro livro. Assim como os quartos do hotel somente pertencerão a ele, não faz sentido retirar o quarto do cadastro e aproveitar para outro hotel (a parte não existe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



5.3.5 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.

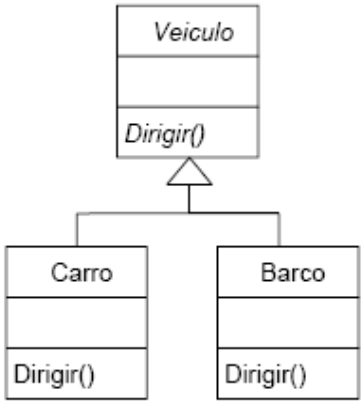


O uso das restrições é facultativo e serve para identificar o que está desenhado nas especializações da superclasse. Pode ser que estejam todos os subtipos desenhados no diagrama, se fosse colocado no exemplo acima poderíamos dizer que todas as contas são do tipo Fundo, Corrente ou Poupança, neste caso 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'.

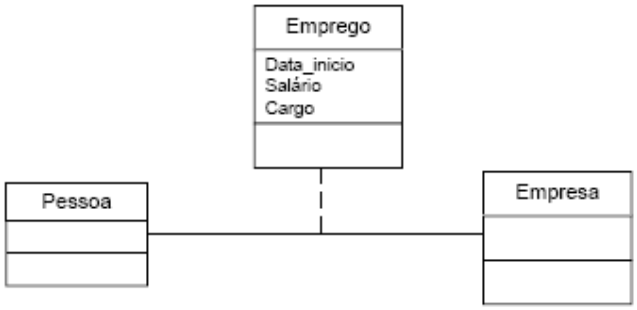
b) Classes Abstratas:

- 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.

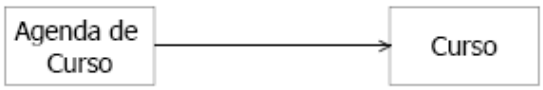


Dependência entre Classes:

- 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;

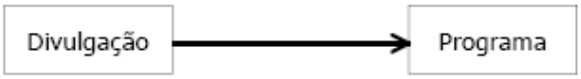
  ii) Uma classe chama uma operação que é de escopo de outra classe.



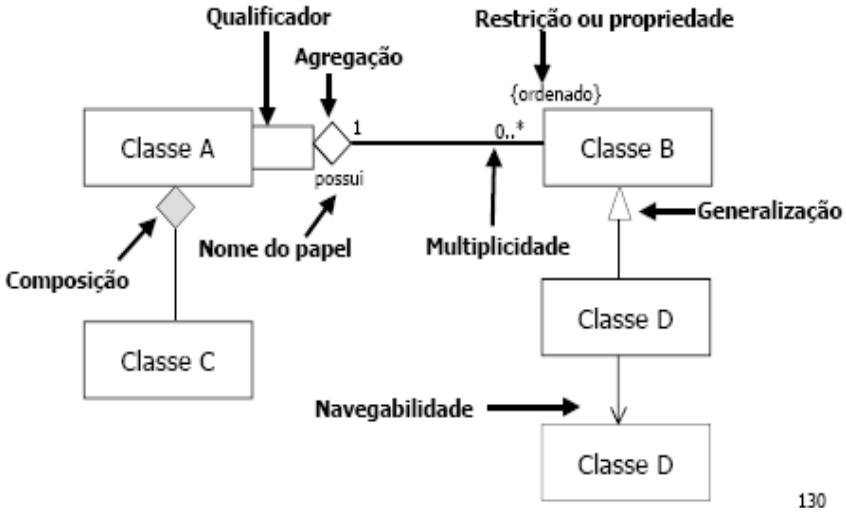
5.3.6 Navegabilidade

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.

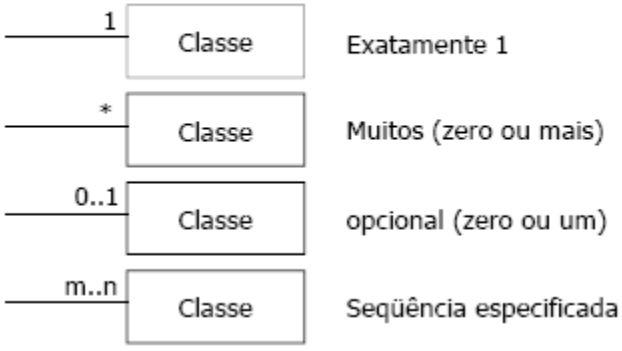
Como no diagrama de classe não tem a representação de chave estrangeira, a navegabilidade indica que a divulgação terá a chave estrangeira da entidade programa, pois divulgação terá a responsabilidade de saber qual programa faz referência.



Papeis em associação:

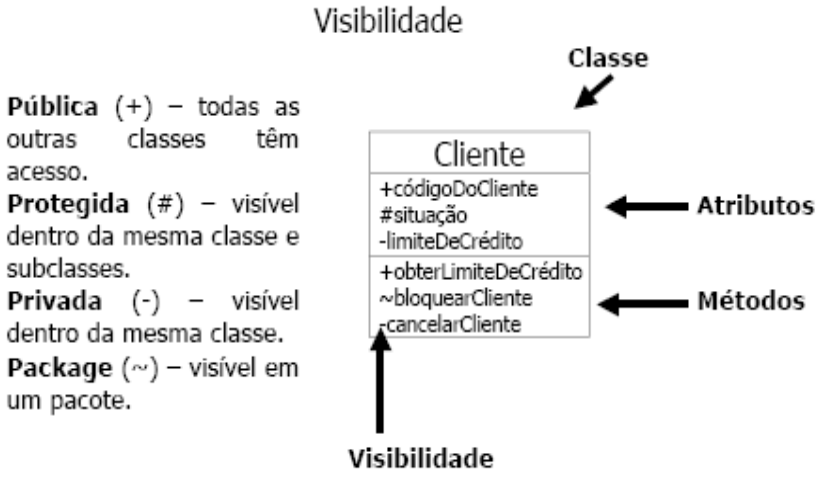


Multiplicidade



5.3.7 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 ideia dos métodos, mas o diagrama de sequência 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.

5.4 Diagrama de Sequência

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

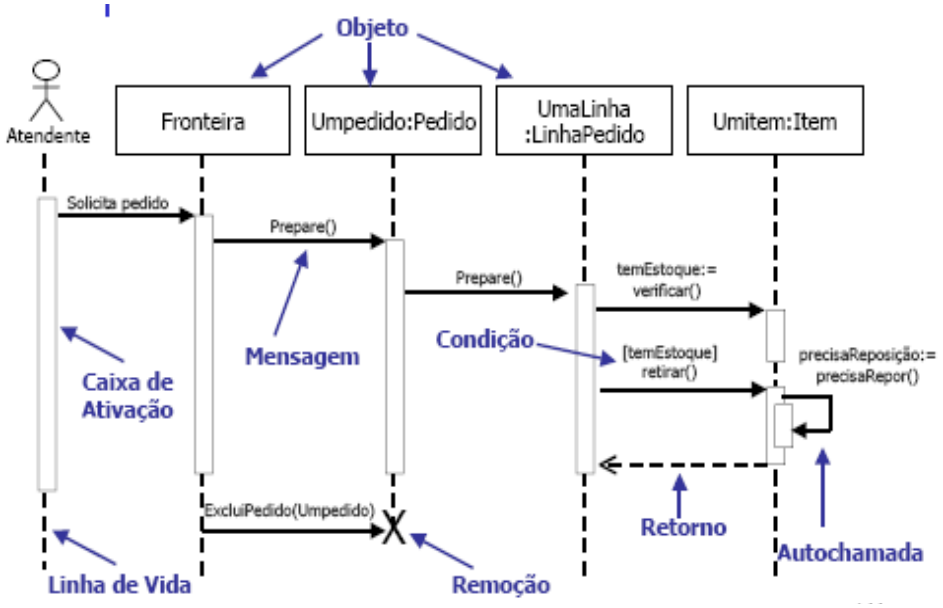
- Diagrama de Sequência – interação entre objetos ao longo do tempo, mostrando todos os objetos que participam do contexto e a sequência de mensagens trocadas.
- Diagrama de Colaboração – representa a troca de mensagens entre um conjunto de objetos.

5.4.1 O Que é o Diagrama de Sequência?

Os **diagramas de sequência** são orientados para exprimir o desenrolar temporal de sequências 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 Sequência: Objetos, Mensagem (retorno, auto-chamada, condição), Linha de Vida, Caixa de Ativação, Remoção.



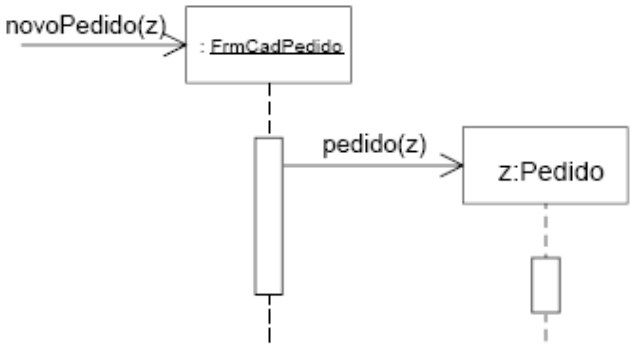
- Mensagem – pode ser um evento ou qualquer outra informação necessária, mas sempre com expectativa de ação a ser executada. Pode incluir parâmetros contendo algum dado adicional e a resposta do objeto destinatário depende do método associado à mensagem. Pode ser também etiquetada com uma expressão booleana (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 Sequência 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 Sequência colocando restrições de tempo nas execução das atividades.

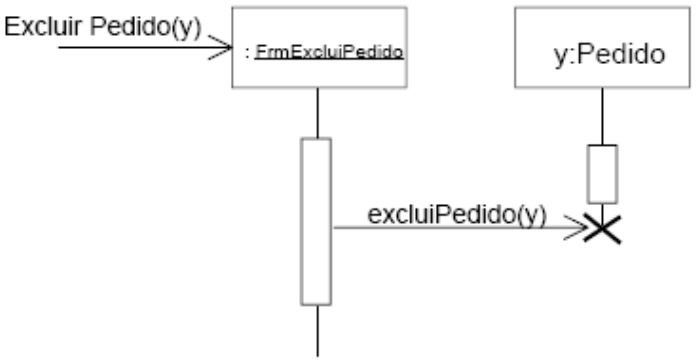
Criação e exclusão de Objetos:

- Criação e exclusão de objetos também são representados em um diagrama de sequência;
- 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.

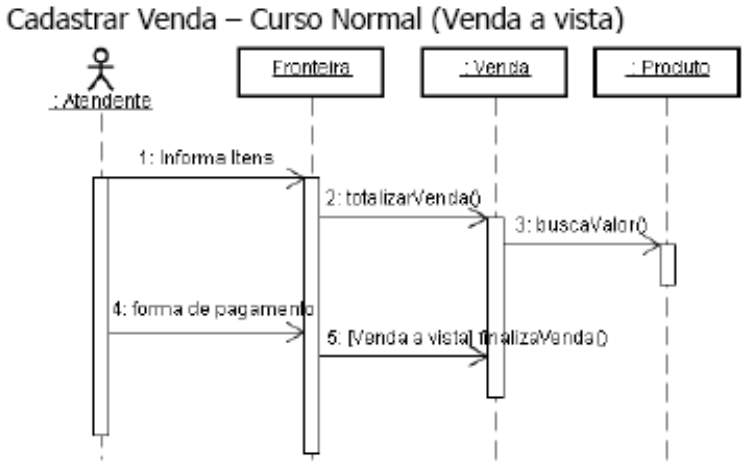
Exemplo: No diagrama pode-se observar que a instância de Cliente é criada a partir da Janela de Atendimento. Pode-se notar que a instância de Cliente aparece um pouco abaixo da instância da janela, indicando que ela foi criada depois.



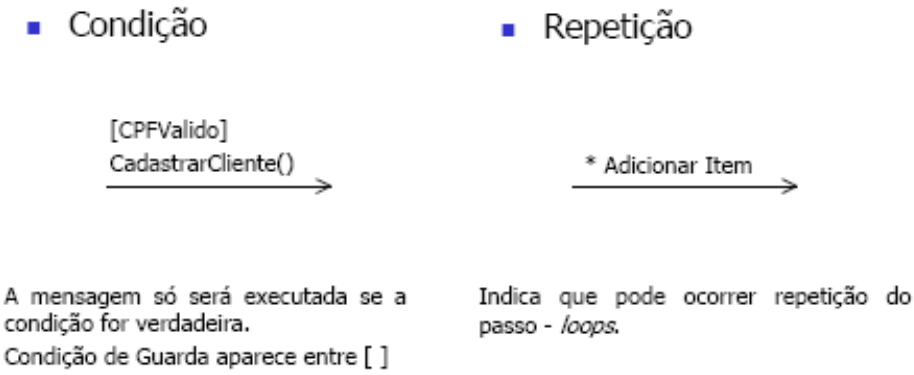
A figura mostra um 'X' na linha de vida da instância de cliente indicando a sua exclusão. Note que a linha de vida do objeto não segue adiante.



Exemplo de diagrama de sequência:



A **condição** no diagrama de sequência 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 sequência, teremos um diagrama de sequência para o curso principal do caso de uso estendido ou incluído e no diagrama de sequência do que inclui colocamos uma nota com a observação. Observações para o desenvolvimento do diagrama:

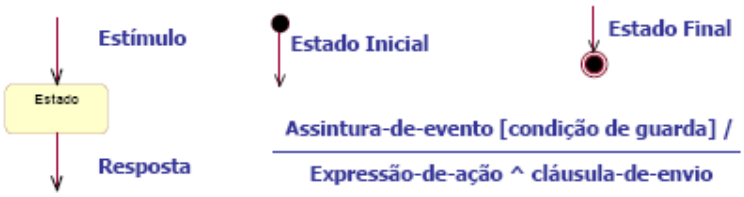
- É permitido o uso de Notas a qualquer momento, mas cuidado para não poluir o desenho.
- Existem autores que utilizam a fronteira como objeto de interface entre o sistema e o ator, representando as telas do sistema, outros autores preferem instanciar cada fronteira com o nome das telas, outros autores não utilizam fronteira de forma nenhuma. Ache sua forma de desenhar.
- O Diagrama de Sequência é um descobridor de métodos e um validador da sequência 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.

5.5 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.

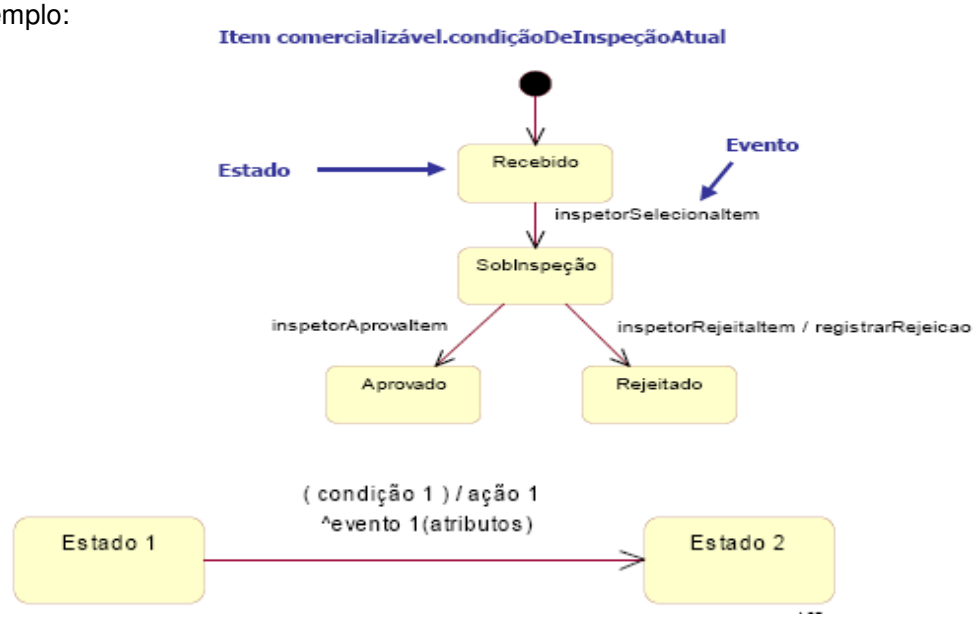
Em sistemas complexos o trabalho é muito extenso e difícil, a UML indica o uso do diagrama de estado por classe, mostrando os estados que os objetos dessa classe podem assumir e as transições que eles podem fazer entre estados, sendo ideal para descrever o comportamento de um único objeto. O sistema 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 subsequente 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:

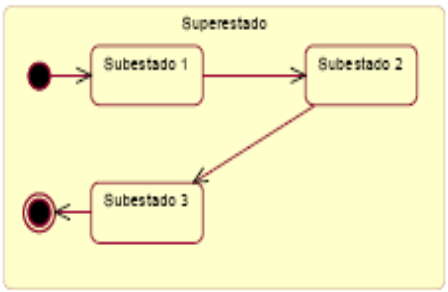
- Evento – é a especificação de uma ocorrência, o que irá ocasionar a alteração do estado de um objeto.
- Ação – atividade atômica que não pode ser interrompida.
- Condição de Guarda – Somente irá mudar para o estado se a condição for verdadeira. Aparece entre colchetes '[ ]' na transição.

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.

5.5.1 Máquina de Estados:

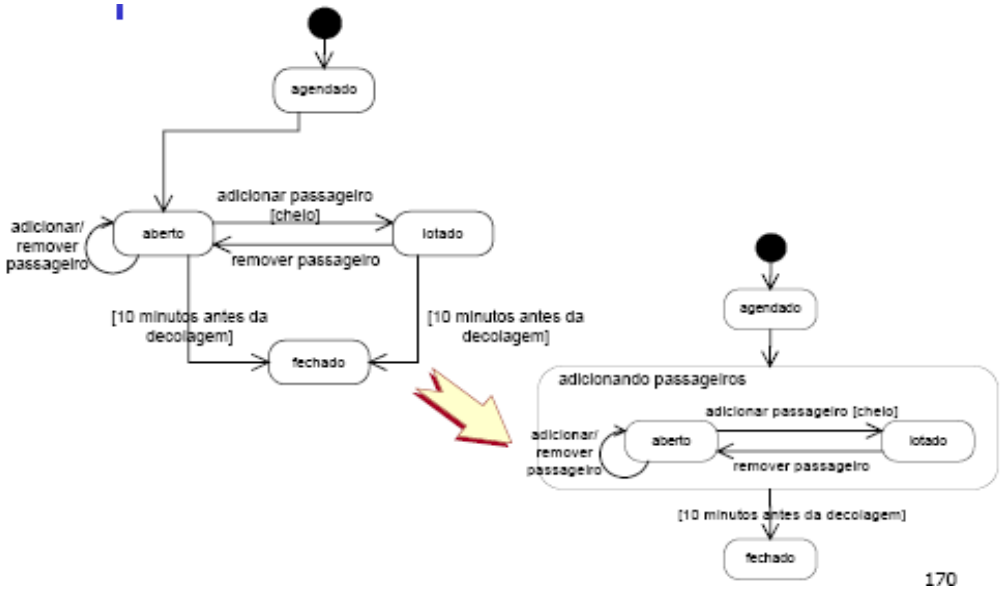


Os superestados são utilizados para minimizar a desorganização do diagrama de estados. Os superestados são compostos de dois ou mais subestados de uma mesma transição.

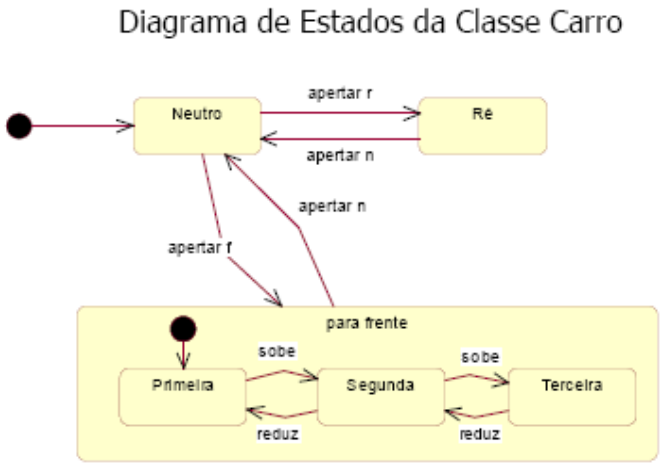
A máquina de estados (ou Superestado ou Estado Composto) foi criada para agrupar estados e facilitar a leitura do diagrama.

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 descobrir os estado do objeto (Pesquisa).

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

- Cadastrar pesquisa;

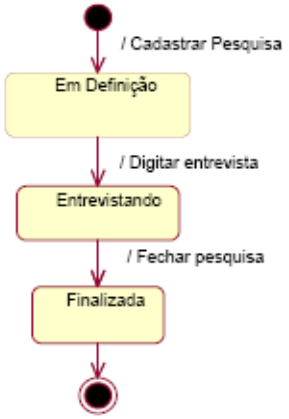
- Digitar entrevista;
- Fechar pesquisa.

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.

5.5.2 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.

6 PROJETO DE SISTEMAS

6.1 Aspectos Fundamentais do Projeto de Sistemas

6.2 Projeto Orientado a Objetos

7 FERRAMENTAS CASE

BIBLIOGRAFIA

CARNEIRO, Margareth Fabíola S. **Gerenciamento de Projetos** (apostila), ENAP, 2000

D´ÁVILA, Márcio. **PMBOK e gerenciamento de projetos**. 2001. Disponível em < http://www.mhavila.com.br/topicos/gestao/pmbok.html>, Acesso em 10 fev. 2012.

FORCHESATTO, André Luiz. **Apostila de Engenharia de Software**. Xanxere: Universidade do Oeste de Santa Catarina.

INPE. **Engenharia de Software**. Disponível em <http://www2.dem.inpe.br/ijar/AnalEstr.html>, acesso em 20 jul. 2013.

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 M.; TEODORO NETO, Euclides M. & GOMES, Gina Marcia S. **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.

Nota do Professor:

Este trabalho é um resumo do conteúdo da disciplina, para facilitar o desenvolvimento das aulas, devendo sempre ser complementado com estudos nos livros recomendados e o desenvolvimento dos exercícios indicados em sala de aula e a resolução das listas de exercícios propostas.