1

Visão Geral

NO DECORRER DA HISTÓRIA, DIVERSOS TIPOS DE BENS SERVIRAM DE base para o desenvolvimento da economia. Propriedade, mão-de-obra, máquinas e capital são exemplos desses bens. Atualmente, está surgindo um novo tipo de bem econômico: a informação. Nos dias de hoje, a empresa que dispõe de mais informações sobre seu processo de negócio está em vantagem em relação a suas competidoras.

Há um ditado que diz que "a necessidade é a mãe das invenções". Em conseqüência do crescimento da importância da informação, surgiu a necessidade de gerenciar informações de uma forma adequada e eficiente e, dessa necessidade, surgiram os denominados *sistemas de informações*.

Um sistema de informações é uma combinação de pessoas, dados, processos, interfaces, redes de comunicação e tecnologia que interagem com o objetivo de dar suporte e melhorar o processo de negócio de uma organização empresarial com relação às informações que nela fluem. Considerando o caráter estratégico da informação nos dias de hoje, pode-se dizer também que os sistemas de informações têm o objetivo de prover vantagens para uma organização do ponto de vista competitivo.

O objetivo principal e final da construção de um sistema de informações é a *adição de valor* à empresa ou organização na qual esse sistema será utilizado. O termo "adição de valor" implica em que a produtividade nos processos da empresa na qual o sistema de informações será utilizado deve aumentar de uma forma significativa, de tal forma a compensar os recursos utilizados na construção do sistema. Para que um sistema de informações adicione valor a uma organização, tal sistema deve ser economicamente justificável.

O desenvolvimento de um sistema de informações é uma tarefa das mais complexas. Um dos seus componentes é denominado ***sistema de software***. Esse componente compreende os módulos funcionais computadorizados que interagem entre si para proporcionar ao(s) usuário(s) do sistema a automatização de diversas tarefas.

1.1 Modelagem de sistemas de software

Uma característica intrínseca de sistemas de software é a complexidade de seu desenvolvimento, que cresce à medida que cresce o tamanho do sistema. Para se ter uma idéia da complexidade envolvida na construção de alguns sistemas, considere o tempo e os recursos materiais necessários para se construir uma casa de cachorro. Para construir essa casa, provavelmente tudo de que se precisa é de algumas ripas de madeira, alguns pregos, uma caixa de ferramentas e uma certa dose de amor por seu cachorro. Depois de alguns dias, a casa estaria pronta. O que dizer da construção de uma casa para sua família? Certamente tal empreitada não seria realizada tão facilmente. O tempo e os recursos necessários seriam uma ou duas ordens de grandeza maiores do que o necessário para a construção da casa de cachorro. O que dizer, então, da construção de um edifício? Certamente, para se construir habitações mais complexas (casas e edifícios), algum planejamento adicional é necessário. Engenheiros e arquitetos constroem plantas dos diversos elementos da habitação antes do início da construção propriamente dita. Na terminologia da construção civil, plantas hidráulicas, elétricas, de fundação etc. são projetadas e devem manter consistência entre si. Provavelmente, uma equipe de profissionais estaria envolvida na construção e aos membros dessa equipe seriam delegadas diversas tarefas, no tempo adequado para cada uma delas.

Na construção de sistemas de software, assim como na construção de sistemas habitacionais, também há uma gradação de complexidade. Para a construção de sistemas de software mais complexos, também é necessário um planejamento inicial anterior. O equivalente ao projeto das plantas da engenharia civil também deve ser realizado. Essa necessidade leva ao conceito de ***modelo***, tão importante no desenvolvimento de sistemas. De uma perspectiva mais ampla, um modelo pode ser visto como uma representação idealizada de um sistema a ser construído. Maquetes de edifícios e de aviões, e plantas de circuitos eletrônicos são apenas alguns exemplos de modelos. São várias as razões da utilização de modelos para a construção de sistemas. Abaixo são enumeradas algumas dessas razões.

1. **Gerenciamento da complexidade:** uma das principais razões pelas quais modelos são utilizados é que há limitações no ser humano em lidar com a

complexidade. Pode haver diversos modelos de um mesmo sistema, cada modelo descrevendo uma perspectiva do sistema a ser construído. Por exemplo, um avião pode ter um modelo para representar sua parte elétrica, outro modelo para representar sua parte aerodinâmica etc. Através de modelos de um sistema, os indivíduos envolvidos no seu desenvolvimento podem fazer estudos e predizer comportamentos do sistema em desenvolvimento. Como cada modelo representa uma perspectiva do sistema, detalhes irrelevantes que podem dificultar o entendimento do sistema podem ser ignorados por um momento estudando-se separadamente cada um dos modelos. Além disso, modelos se baseiam no denominado *Princípio da Abstração* (ver Seção 1.2.3), o qual advoga que só as características relevantes à resolução de um problema devem ser consideradas. Modelos revelam as características essenciais de um sistema; detalhes não-relevantes e que só aumentariam a complexidade do problema podem ser ignorados.

2. **Comunicação entre as pessoas envolvidas:** certamente, o desenvolvimento de um sistema envolve a execução de uma quantidade significativa de atividades. Essas atividades se traduzem em informações sobre o sistema em desenvolvimento. Grande parte dessas informações corresponde aos modelos criados para representar o sistema. Nesse sentido, os modelos de um sistema servem também para promover a difusão de informações relativas ao sistema entre os indivíduos envolvidos em sua construção. Além disso, diferentes expectativas em relação ao sistema geralmente aparecem quando da construção dos seus modelos, já que estes servem como um ponto de referência comum.
3. **Redução dos custos no desenvolvimento:** no desenvolvimento de sistemas, seres humanos estão invariavelmente sujeitos a cometerem erros, que podem ser tanto erros individuais quanto erros de comunicação entre os membros da equipe. Certamente, a correção desses erros é menos custosa quando detectada e realizada ainda no(s) modelo(s) do sistema (por exemplo, é muito mais fácil corrigir uma maquete do que pôr abaixo uma parte de um edifício). Lembre-se: modelos de sistemas são mais baratos de construir do que sistemas. Conseqüentemente, erros identificados sobre modelos têm um impacto menos desastroso.
4. **Predição do comportamento futuro do sistema:** o comportamento do sistema pode ser discutido através da análise dos seus modelos. Os modelos servem como um "laboratório", onde diferentes soluções para um problema relacionado à construção do sistema podem ser experimentadas.

Uma outra questão importante é sobre como são os modelos de sistemas de software. Em construções civis, freqüentemente há profissionais analisando as

plantas da construção sendo realizada. A partir dessas, que podem ser vistas como modelos, os homens tomam decisões sobre o andamento da obra. Modelos de sistemas de software não são muito diferentes dos modelos de sistemas da construção civil. Nos próximos capítulos, apresentaremos diversos modelos cujos componentes são desenhos gráficos que seguem algum padrão lógico. Esses desenhos são normalmente denominados ***diagramas***. Um diagrama é uma apresentação de uma coleção de ***elementos gráficos*** que possuem um significado predefinido.

Talvez o fato de os modelos serem representados por diagramas possa ser explicado pelo ditado que diz que "uma figura vale por mil palavras". Através de desenhos gráficos que modelam o sistema, os desenvolvedores têm uma representação concisa do sistema. No entanto, modelos de sistemas de software também são compostos de informações textuais. Embora um diagrama consiga expressar diversas informações de forma gráfica, em diversos momentos há a necessidade de adicionar informações na forma de texto, com o objetivo de explicar ou definir certas partes desse diagrama. Dado um modelo de uma das perspectivas de um sistema, diz-se que o seu diagrama, juntamente com a informação textual associada, formam a *documentação* desse modelo.

> A modelagem de sistemas de software consiste na utilização de notações gráficas e textuais com o objetivo de construir modelos que representam as partes essenciais de um sistema, considerando-se diversas perspectivas diferentes e complementares.

1.2 O paradigma da orientação a objetos

Essencial ao desenvolvimento atual de sistemas de software é o *paradigma da orientação a objetos*. Esta seção descreve o que esse termo significa e justifica por que a orientação a objetos é importante para a modelagem de sistemas.

Pode-se começar pela definição da palavra *paradigma*. Essa palavra possui diversos significados distintos, mas o que mais se aproxima do significado aqui utilizado pode ser encontrado no dicionário Aurélio Século XXI:

> *paradigma . [Do gr. parádeigma, pelo lat. tard. paradigma.] S. m. Termo com o qual Thomas Kuhn designou as realizações científicas (p. ex., a dinâmica de Newton ou a química de Lavoisier) que geram modelos que, por período mais ou menos longo e de modo mais ou menos explícito, orientam o desenvolvimento posterior das pesquisas exclusivamente na busca da solução para os problemas por elas suscitados.*

Para o leitor que ainda não se sentiu satisfeito com essa definição, temos aqui uma outra definição, mais restrita e mais apropriada ao contexto deste li-

vro: *um paradigma é uma forma de abordar um problema*. Como exemplo, considere a famosa história da maçã caindo sobre a cabeça de Isaac Newton, citado na definição anterior.[1] Em vez de pensar que somente a maçã estava caindo sobre a Terra, Newton também considerou a hipótese de o próprio planeta também estar caindo sobre a maçã! Essa outra maneira de abordar o problema pode ser vista como um paradigma.

Pode-se dizer, então, que o termo "paradigma da orientação a objetos" é uma forma de abordar um problema. Há alguns anos, Alan Kay, um dos pais do paradigma da orientação a objetos, formulou a chamada "analogia biológica". Nessa analogia, ele imaginou como seria um sistema de software que funcionasse como um ser vivo. Nesse sistema, cada "célula" interagiria com outras células através do envio de mensagens para realizar um objetivo comum. Adicionalmente, cada célula se comportaria como uma unidade autônoma.

De uma forma mais geral, Kay pensou em como construir um sistema de software a partir de agentes autônomos que interagem entre si. Ele, então, estabeleceu os seguintes princípios da orientação a objetos:

1. Qualquer coisa é um objeto.
2. Objetos realizam tarefas através da requisição de serviços a outros objetos.
3. Cada objeto pertence a uma determinada *classe*. Uma classe agrupa objetos similares.
4. A classe é um repositório para comportamento associado ao objeto.
5. Classes são organizadas em hierarquias.

Vamos ilustrar esses princípios com a seguinte história: suponha que alguém queira comprar uma pizza. Chame este alguém de João. João está muito ocupado em casa e resolve pedir a sua pizza por telefone. João liga para a pizzaria e realiza o seu pedido. João informa ao atendente (digamos, o José) seu nome, as características da pizza desejada e o seu endereço. José, que só realiza a função de atendente, então comunica à Maria, funcionária da pizzaria responsável por fazer as pizzas, qual pizza deve ser feita. Quando Maria termina de fazer a pizza, José chama Antônio, o entregador. Finalmente, João recebe a pizza desejada das mãos de Antônio meia hora depois de tê-la pedido.

Pode-se observar que o objetivo de João foi atingido através da colaboração de diversos agentes, que são denominados *objetos*. Há diversos objetos na história (1º princípio): João, Maria, José, Antônio. Todos colaboram com uma parte, e o objetivo é alcançado quando todos trabalham juntos (2º princípio). Além

[1] Talvez tal história não seja verídica, mas ilustra bem o conceito que quero passar.

disso, o comportamento esperado de Antônio é o mesmo esperado de qualquer entregador. Diz-se que Antônio é um objeto da classe Entregador (3º princípio). Um comportamento comum a todo entregador, não somente ao Antônio, é o de entregar a mercadoria no endereço especificado (4º princípio). Finalmente, José, o atendente, é também um ser humano, também mamífero, também um animal etc (5º princípio).

Mas o que o paradigma da orientação a objetos tem a ver com a modelagem de sistemas? Antes da orientação a objetos, um outro paradigma era utilizado na modelagem de sistemas[2]: o paradigma estruturado. Nesse paradigma, os elementos são *dados* e *processos*. Processos agem sobre dados para que um objetivo seja alcançado. Por outro lado, no paradigma da orientação a objetos, há um elemento, o objeto, uma unidade autônoma que contém seus próprios dados que são manipulados pelos processos definidos para o objeto e que interage com outros objetos para alcançar um objetivo. É o paradigma da orientação a objetos que os seres humanos utilizam no cotidiano para a resolução de problemas. Uma pessoa atende a mensagens (requisições) para realizar um serviço; essa mesma pessoa envia mensagens a outras para que estas realizem serviços. Por que não aplicar essa mesma forma de pensar à modelagem de sistemas?

O paradigma da orientação a objetos visualiza um sistema de software como uma coleção de agentes interconectados chamados *objetos*. Cada objeto é responsável por realizar tarefas específicas. É através da interação entre objetos que uma tarefa computacional é realizada.

Pode-se concluir que a orientação a objetos, como técnica para modelagem de sistemas, diminui a diferença semântica entre a realidade sendo modelada e os modelos construídos. Este livro descreve o papel importante da orientação a objetos na modelagem de sistemas de software atualmente. Explícita ou implicitamente, as técnicas de modelagem de sistemas aqui descritas utilizam os princípios que Alan Kay estabeleceu há mais de 30 anos.

As seções a seguir continuam descrevendo os conceitos principais da orientação a objetos.

[2] Na verdade, tanto o paradigma estruturado quanto o paradigma orientado a objetos surgiram nas linguagens de programação, para depois serem aplicados à modelagem de sistemas. De fato, as idéias de Alan Kay foram aplicadas na construção de uma das primeiras linguagens de programação orientadas a objetos: o SmallTalk.

Um sistema de software orientado a objetos consiste de objetos em colaboração com o objetivo de realizar as funcionalidades desse sistema. Cada objeto é responsável por tarefas específicas. É através da cooperação entre objetos que a computação do sistema se desenvolve.

1.2.1 Classes e objetos

O mundo real é formado de coisas. Como exemplos de coisas pode-se citar um cliente, uma loja, uma venda, um pedido de compra, um fornecedor, este livro etc. Na terminologia de orientação a objetos, essas coisas do mundo real são denominadas *objetos*.

Seres humanos costumam agrupar os objetos. Provavelmente, os seres humanos realizam esse processo mental de agrupamento para tentar gerenciar a complexidade de entender as coisas do mundo real. Realmente, é bem mais fácil entender a idéia *cavalo* do que entender todos os cavalos que existem. Na terminologia da orientação a objetos, cada idéia é denominada *classe de objetos*, ou simplesmente *classe*. Uma classe é uma descrição dos atributos e serviços comuns a um grupo de objetos. Sendo assim, pode-se entender uma classe como sendo um *molde* a partir do qual objetos são construídos. Ainda sobre terminologia, diz-se que um objeto é uma *instância* de uma classe.

Por exemplo, quando se pensa em um cavalo, logo vem à mente um animal de quatro patas, cauda, crina etc. Pode ser que algum dia você veja dois cavalos, um mais baixo que o outro, um com cauda maior que o outro, ou mesmo, por um infeliz acaso, um cavalo com menos patas que o outro. No entanto, você ainda terá certeza de estar diante de dois cavalos. Isso porque a *idéia* (classe) cavalo está formada na mente dos seres humanos, independentemente das pequenas diferenças que possa haver entre os *exemplares* (objetos) da idéia cavalo.

É importante notar que uma classe é uma *abstração* das características de um grupo de coisas do mundo real. Na maioria das vezes, as coisas do mundo real são muito complexas para que *todas* as suas características sejam representadas em uma classe. Além disso, para fins de modelagem de um sistema, somente um subconjunto de características pode ser relevante. Portanto, uma classe representa uma abstração das características *relevantes* do mundo real.

Finalmente, é preciso atentar para o fato de que alguns textos sobre orientação a objetos (inclusive este livro!) utilizam os termos *classe* e *objeto* equivalentemente para denotar uma classe de objetos.

1.2.2 Mensagens

Objetos não executam suas operações aleatoriamente. Para que uma operação em um objeto seja executada, deve haver um estímulo enviado a esse objeto. Se um objeto for visto como uma entidade ativa que representa uma abstração de algo do mundo real, então faz sentido dizer que tal objeto pode responder a estímulos a ele enviados (assim como faz sentido dizer que seres vivos reagem a estímulos que eles recebem). Independentemente da origem do estímulo, quando ele ocorre, diz-se que o objeto em questão está recebendo uma ***mensagem*** requisitando que ele realize alguma operação.

Quando se diz na terminologia de orientação a objetos que *objetos de um sistema estão trocando mensagens* significa que esses objetos estão enviando mensagens uns aos outros com o objetivo de realizar alguma tarefa dentro do sistema no qual eles estão inseridos.

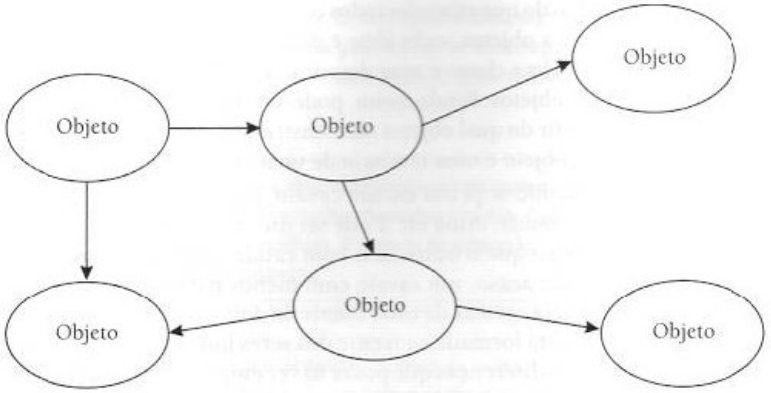

Figura 1-1 Objetos interagem através do envio de mensagens.

1.2.3 O papel da abstração na orientação a objetos

Nesta seção, apresentamos os princípios do paradigma da orientação a objetos. Discutiremos também o argumento de que todos esses princípios são, na verdade, a aplicação de um único princípio mais básico, o *Princípio da Abstração*.

Uma abstração é qualquer modelo que inclui os aspectos mais importantes, essenciais de alguma coisa, ao mesmo tempo em que ignora os detalhes menos importantes. Abstrações permitem gerenciar a complexidade e concentrar a atenção nas características essenciais de um objeto. Note que uma abstração é dependente da perspectiva: o que é importante em um contexto pode não ser importante em outro.

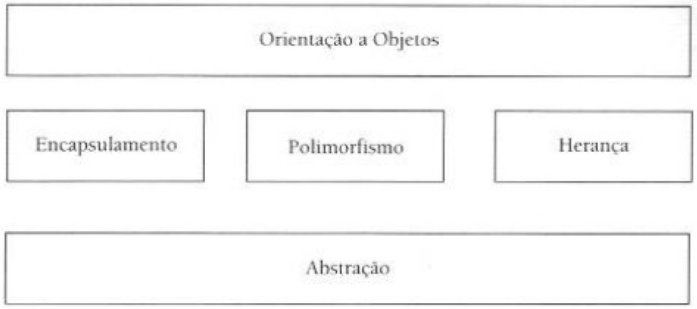

Figura 1-2 Princípios da orientação a objetos como aplicações do Princípio da Abstração.

1.2.3.1 Encapsulamento

Objetos possuem ***comportamento***. O termo comportamento diz respeito a operações realizadas por um objeto e também ao modo pelo qual essas operações são executadas. O mecanismo de *encapsulamento* é uma forma de restringir o acesso ao comportamento interno de um objeto. Um objeto que precise da colaboração de outro objeto para realizar alguma tarefa simplesmente envia uma mensagem a este último. O método que o objeto requisitado usa para realizar a tarefa não é conhecido dos objetos requisitantes.

Certamente, o objeto requisitante precisa conhecer quais as tarefas que um outro objeto sabe fazer ou que informação ele conhece. Para tanto, a classe de um objeto descreve o seu comportamento. Na terminologia da orientação a objetos, diz-se que um objeto possui uma ***interface*** (ver Figura 1-3). Em termos bastante simples, a interface de um objeto é o que ele conhece e o que ele sabe fazer, sem descrever *como* o objeto conhece o faz. A interface de um objeto define os serviços que ele pode realizar e conseqüentemente as mensagens que ele recebe. Uma interface pode ter várias formas de ***implementação***. Mas, pelo Princípio do Encapsulamento, a implementação de um objeto requisitado não importa para um objeto requisitante.

Através do encapsulamento, a única coisa que um objeto precisa saber para pedir a colaboração de outro objeto é conhecer a sua interface. Nada mais. Isso contribui para a autonomia dos objetos. Cada objeto envia mensagens a outros objetos para realizar certas tarefas, sem se preocupar em *como* as tarefas são realizadas. A aplicação da abstração, neste caso, está em esconder os detalhes de funcionamento interno de um objeto.

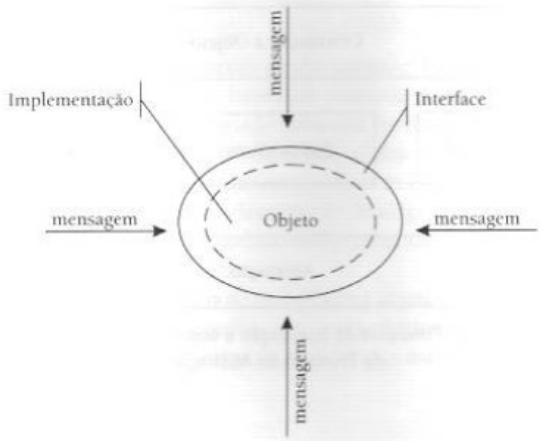

Figura 1-3 Princípio do Encapsulamento: visto externamente, o objeto é a sua interface.

1.2.3.2 Polimorfismo

O polimorfismo indica a capacidade de abstrair várias implementações diferentes em uma única interface. Há algum tempo atrás, o controle remoto de meu televisor quebrou. (Era realmente enfadonho ter de levantar para desligar o aparelho ou para trocar de canal). Um tempo depois, comprei um videocassete do mesmo fabricante de meu televisor. Para minha surpresa, o controle remoto do videocassete também funcionava para o televisor!

Esse é um exemplo de aplicação do ***Princípio do Polimorfismo***. Note mais uma vez que a abstração também é aplicada aqui: um objeto pode enviar a *mesma* mensagem para objetos semelhantes, mas que implementam a sua interface de formas diferentes.

1.2.3.3 Herança

A herança é outra forma de abstração utilizada na orientação a objetos. A Seção 1.2.1 declara que as características e o comportamento comuns a um conjunto de objetos podem ser abstraídos em uma classe. A herança pode ser vista como um nível de abstração acima da encontrada entre classes e objetos.

Na herança, classes semelhantes são agrupadas em hierarquias (ver Figura 1-4). Cada nível de uma hierarquia pode ser visto como um nível de abstração. Cada classe em um nível da hierarquia herda as características das classes nos níveis acima. Esse mecanismo facilita o compartilhamento de comportamento comum entre um conjunto de classes semelhantes. Além disso, as diferenças ou variações de uma classe em particular podem ser organizadas de forma mais clara.

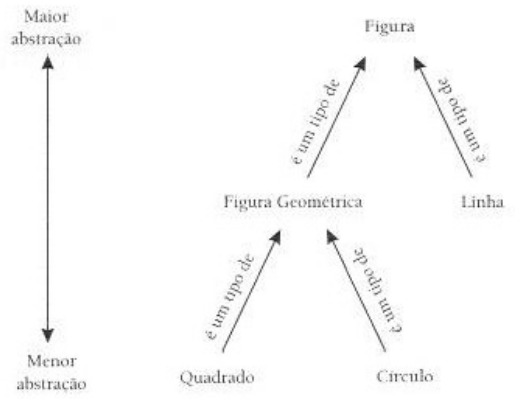

Figura 1-4 Princípio da Herança: classes podem ser organizadas em hierarquias.

1.3 Evolução histórica da modelagem de sistemas

A *Lei de Moore* é bastante conhecida na computação. Essa lei foi declarada em 1965 pelo engenheiro Gordon Moore, co-fundador da Intel, e estabelece que "a densidade de um transistor dobra em um período entre 18 e 24 meses". Isso significa que se um processador pode ser construído hoje com capacidade de processamento P, em 24 meses, pode-se construir um processador com capacidade 2P. Em 48 meses, um com capacidade 4P e assim por diante. Ou seja, a Lei de Moore implica em uma taxa de crescimento exponencial na capacidade de processamento dos computadores.[3]

O que a Lei de Moore tem a ver com a modelagem de sistemas? Bom, provavelmente o ditado "a necessidade é a mãe das invenções" também se aplique neste caso. O rápido crescimento da capacidade computacional das máquinas resultou na demanda por sistemas de software cada vez mais complexos, que tirassem proveito de tal capacidade. Por sua vez, o surgimento desses sistemas mais complexos resultou na necessidade de reavaliação da forma de se desenvolver sistemas. Conseqüentemente, desde o aparecimento do primeiro computador até os dias de hoje, as técnicas utilizadas para a construção de sistemas computacionais têm evoluído de forma impressionante, notavelmente no que tange à modelagem de sistemas.

[3] Desde que foi declarada, a Lei de Moore vem se verificando com uma precisão impressionante. No entanto, alguns especialistas, incluindo o próprio Moore, acreditam que a capacidade de dobrar a capacidade de processamento dos processadores a cada 18 meses não seja mais possível por volta de 2017.

A seguir, temos um breve resumo histórico da evolução das técnicas de desenvolvimento com o objetivo de esclarecer como se chegou às técnicas atualmente utilizadas.[4]

- **Décadas de 1950/60**: os sistemas de software eram bastante simples. O desenvolvimento desses sistemas era feito de forma "ad-hoc".[5] Os sistemas eram significativamente mais simples e, conseqüentemente, as técnicas de modelagem também eram mais simples: era a época dos *fluxogramas* e dos *diagramas de módulos*.
- **Década de 1970**: nesta época, computadores mais avançados e acessíveis começaram a surgir. Houve uma grande expansão do mercado computacional. Sistemas mais complexos começavam a surgir. Por conseguinte, modelos mais robustos foram propostos. Neste período, surgem a *programação estruturada* e o *projeto estruturado*. Os autores Larry Constantine e Edward Yourdon são grandes colaboradores nessas técnicas.
- **Década de 1980**: nesta fase, os computadores se tornaram ainda mais avançados e baratos. Surge a necessidade por interfaces mais sofisticadas, o que originou a produção de sistemas de softwares mais complexos. A *Análise Estruturada* surgiu no início deste período com os trabalhos de Edward Yourdon, Peter Coad, Tom DeMarco, James Martin e Chris Gane.
- **Início da Década de 1990**: este é o período em que surge um novo paradigma de modelagem, a *Análise Orientada a Objetos*. Grandes colaboradores deste paradigma são Sally Shlaer, Stephen Mellor, Rebecca Wirfs-Brock, James Rumbaugh, Grady Booch e Ivar Jacobson.
- **Fim da Década de 1990**: o paradigma da orientação a objetos atinge sua maturidade. Os conceitos de padrões de projeto, frameworks, componentes e qualidade começam a ganhar espaço. Surge a Linguagem de Modelagem Unificada (UML).

Um detalhamento da primeira metade da década de 1990 mostra que surgiram várias propostas de técnicas para modelagem de sistemas segundo o paradigma orientado a objetos. A Tabela 1-1 lista algumas das técnicas existentes durante esse período; nota-se uma grande proliferação de propostas para

[4] É importante notar que a divisão de períodos aqui apresentada é meramente didática. Na realidade, não há uma divisão tão clara das diversas propostas de modelagem. Por exemplo, propostas iniciais de modelagem orientadas a objetos podem ser encontradas já em meados da década de 1970.

[5] Este termo significa "*direto ao assunto*" ou "*direto ao que interessa*". Talvez o uso deste termo denote a abordagem desta primeira fase do desenvolvimento de sistemas, na qual não havia um planejamento inicial. O código-fonte do programa a ser construído era, ele próprio, o modelo.

modelagem orientada a objetos. Era comum duas técnicas possuírem diferentes notações gráficas para modelar uma mesma perspectiva de um sistema. Ao mesmo tempo, cada técnica tinha seus pontos fortes e fracos em relação à notação utilizada.

A essa altura percebeu-se a necessidade da existência de uma linguagem que viesse a se tornar um padrão para a modelagem de sistemas, que fosse aceita e utilizada amplamente pela indústria e pelos ambientes acadêmicos. Surgiram, então, alguns esforços nesse sentido de padronização, o que resultou na definição da UML (Unified Modeling Language) em 1996 como a melhor candidata para ser a linguagem "unificadora" de notações, diagramas e formas de representação existentes em diferentes técnicas.

***Tabela 1-1* Principais propostas de técnicas de modelagem orientada a objetos durante a década de 1990**

Ano	Autor (Técnica)
1990	Shaler & Mellor
1991	Coad & Yourdon (OOAD – Object-Oriented Analysis and Design)
1993	Grady Booch (Booch Method)
1993	Ivar Jacobson (OOSE – Object-Oriented Software Engineering)
1995	James Rumbaugh et al (OMT – Object Modeling Technique)
1996	Wirfs-Brock (Responsibility Driven Design)
1996	(Fusion)

1.4 A Linguagem de Modelagem Unificada (UML)

A construção da UML teve muitos contribuintes, mas os principais atores no processo foram Grady Booch, James Rumbaugh e Ivar Jacobson. Esses três pesquisadores são freqüentemente chamados de "*os três amigos*". No processo de definição da UML, procurou-se aproveitar o melhor das características das notações preexistentes, principalmente das técnicas propostas anteriormente pelos três amigos (essas técnicas eram conhecidas pelos nomes *Booch Method*, *OMT* e *OOSE*). A notação definida para a UML é uma união de diversas notações preexistentes, com alguns elementos removidos e outros elementos adicionados com o objetivo de torná-la mais expressiva.

Finalmente, em 1997, a UML foi aprovada como padrão pelo OMG.[6] Desde então, a UML tem tido grande aceitação pela comunidade de desenvolvedores

[6] Sigla para Object Management Group. O OMG é um consórcio internacional de empresas que define e ratifica padrões na área da orientação a objetos (www.omg.org).

de sistemas. A sua definição ainda está em desenvolvimento e possui diversos colaboradores da área comercial.[7] Tanto que, desde o seu surgimento, várias atualizações foram feitas no sentido de torná-la mais clara e útil.

A UML é uma ***linguagem visual*** para modelar sistemas orientados a objetos. Isso quer dizer que a UML é uma linguagem constituída de elementos gráficos (visuais) utilizados na modelagem que permitem representar os conceitos do paradigma da orientação a objetos. Através dos elementos gráficos definidos nesta linguagem pode-se construir diagramas que representam diversas perspectivas de um sistema.

Cada elemento gráfico possui uma ***sintaxe*** (isto é, uma forma predeterminada de desenhar o elemento) e uma ***semântica*** que definem o que significa o elemento e para que ele deve ser utilizado. Além disso, conforme descrito mais adiante, tanto a sintaxe quanto a semântica da UML são ***extensíveis***. Essa extensibilidade permite que a UML seja adaptada às características específicas de cada projeto de desenvolvimento.

Pode-se fazer uma analogia da UML com uma caixa de ferramentas. Um construtor usa sua caixa de ferramentas para realizar suas tarefas. Da mesma forma, a UML pode ser vista como uma caixa de ferramentas utilizada pelos desenvolvedores de sistemas para realizar a construção de modelos.

A UML é ***independente tanto de linguagens de programação quanto de processos de desenvolvimento***. Isso quer dizer que a UML pode ser utilizada para a modelagem de sistemas, não importa qual a linguagem de programação será utilizada na implementação do sistema, ou qual a forma (processo) de desenvolvimento adotada. Esse é um fator importante para a utilização da UML, pois diferentes sistemas de software requerem diferentes abordagens de desenvolvimento.

A definição completa da UML está contida na ***Especificação da Linguagem de Modelagem Unificada da OMG***. Essa especificação pode ser obtida gratuitamente no site da OMG (www.omg.org). Essa documentação inclui diversas informações, dentre elas:

- *Sumário da UML (UML Summary)*: uma breve introdução aos conceitos básicos e à evolução histórica da UML;
- *Semântica da UML (UML Semantics)*: as regras e semânticas definidas para essa linguagem que contribuem para a criação de modelos consistentes;
- *Guia da Notação da UML (UML Notation Guide)*: definição das diversas notações gráficas fornecidas pela UML;

[7] Algumas das empresas que participam da definição da UML são: Digital, HP, IBM, Oracle, Microsoft, Unisys, IntelliCorp, i-Logix e Rational.

- *Extensões da UML (UML Extensions)*: diversos elementos de extensão da linguagem de modelagem unificada.

Embora essa documentação seja bastante completa, ela está longe de fornecer uma leitura fácil, pois é direcionada a pesquisadores ou a desenvolvedores de ferramentas de suporte ao desenvolvimento de sistemas.

1.4.1 Visões de um sistema

O desenvolvimento de um sistema de software complexo demanda que seus desenvolvedores tenham a possibilidade de examinar e estudar esse sistema a partir de diversas perspectivas. Os autores da UML sugerem que um sistema pode ser descrito por cinco visões interdependentes desse sistema (Booch et al., 2000). Cada visão enfatiza aspectos diferentes do sistema. As visões propostas são as seguintes:

- *Visão de Casos de Uso*: descreve o sistema de um ponto de vista externo como um conjunto de interações entre o sistema e os agentes externos ao sistema. Esta visão é criada inicialmente e direciona o desenvolvimento das outras visões do sistema.
- *Visão de Projeto*: enfatiza as características do sistema que dão suporte, tanto estrutural quanto comportamental, às funcionalidades externamente visíveis do sistema.
- *Visão de Implementação*: abrange o gerenciamento de versões do sistema, construídas através do agrupamento de módulos (componentes) e subsistemas.
- *Visão de Implantação*: corresponde à distribuição física do sistema em seus subsistemas e à conexão entre essas partes.
- *Visão de Processo*: esta visão enfatiza as características de concorrência (paralelismo), sincronização e desempenho do sistema.

Dependendo das características e da complexidade do sistema, nem todas as visões precisam ser construídas. Por exemplo, se o sistema tiver de ser instalado em um ambiente computacional de processador único, não há necessidade da visão de implantação. Outro exemplo: se o sistema for constituído de um único processo, a visão de processo é irrelevante. De forma geral, dependendo do sistema, as visões podem ser ordenadas por grau de relevância.

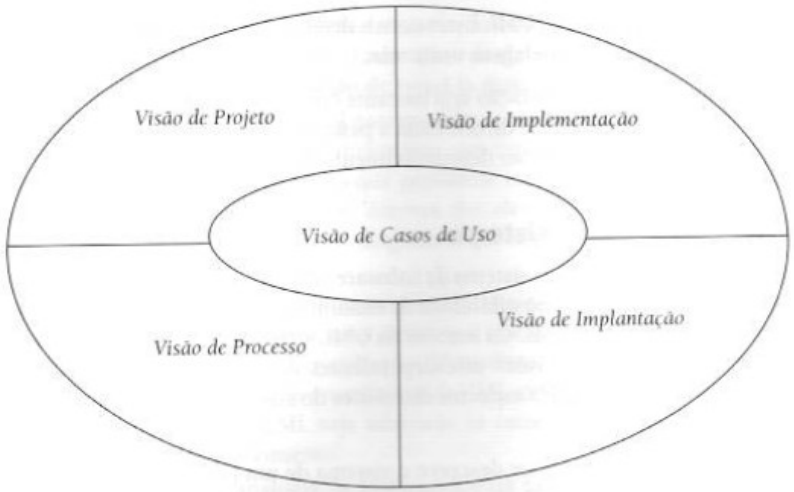

Figura 1-5 Visões (perspectivas) de um sistema de software.

1.4.2 Diagramas da UML

Um processo de desenvolvimento que utilize a UML como linguagem de suporte à modelagem envolve a criação de diversos documentos. Esses documentos podem ser textuais ou gráficos. Na terminologia da UML, esses documentos são denominados ***artefatos de software***, ou simplesmente *artefatos*. São os artefatos que compõem as visões do sistema.

Os artefatos gráficos produzidos durante o desenvolvimento de um sistema de software são definidos através da utilização dos diagramas da UML. Os diagramas da UML são listados na Figura 1-6. Nesta Figura, os retângulos com os cantos retos representam agrupamentos (tipos) de diagramas da UML. Já os retângulos com os cantos boleados representam os diagramas propriamente ditos.

O iniciante na modelagem de sistemas pode muito bem perguntar: "*Para que um número tão grande de diagramas para modelar um sistema? Será que um ou dois tipos de diagramas já não seriam suficientes?*". Para justificar a existência de vários diagramas, vamos utilizar novamente uma analogia. Carrinhos em miniatura são cópias fiéis de carros de verdade. Cada um dos carrinhos é um modelo físico tridimensional que pode ser analisado de diversas perspectivas (por cima, por baixo, por dentro etc.). O mesmo não ocorre com os diagramas, que são desenhos bidimensionais.