Do Diagrama de Fluxo de Dados ao Use Case

Alessandro Botelho Bovo[1,2], Renato Balancieri[1,2], Sandra Ferrari[2]

[1]Grupo Stela – Universidade Federal de Santa Catarina (UFSC)
Rua Lauro Linhares, 2123, bloco B, 2º andar – 88036-002 – Florianópolis – SC

[2]Programa de Pós-Graduação em Engenharia de Produção
Universidade Federal de Santa Catarina (UFSC)

{abbovo,renato}@stela.ufsc.br, sandraferrari@terra.com.br

Abstract. *This article has the objective of giving a brief view about two of the most used methodologies of development of systems: the structured method for analysis and design of software development and the objects oriented method. The main features of the tools used to the modeling of systems that use of these methods for its development are presented. Inside of the possible, it's intended to make a matching between them, aiming at, inside of the defined target, to show the advantages and disadvantages of its use, given the existing conditions. Finally, some points of view that can contribute for extend the debate concerning the application of modeling tools, considering the analysis method used, are presented.*

Keywords: *diagrams, modeling, tools, use cases, DFD, UML, objects oriented.*

Resumo*. Este artigo tem como objetivo dar uma rápida visão sobre duas das mais utilizadas metodologias de desenvolvimento de sistemas: o método estruturado para análise e projeto de desenvolvimento de software e o método orientado a objetos. São apresentadas as principais características das ferramentas utilizadas para a modelagem de sistemas que fazem uso desses métodos para o seu desenvolvimento. Pretende-se, dentro do possível, dadas às condições existentes, fazer uma comparação entre elas, visando, dentro do escopo definido, mostrar as vantagens e desvantagens do seu uso. Por fim, são apresentados alguns pontos de vistas que podem contribuir para a ampliação do debate acerca da aplicação de ferramentas de modelagem, considerando o método de análise utilizado.*

Palavras-chave: *diagramas, modelagem, ferramentas, use cases, DFD, UML, orientado a objetos.*

1. Introdução

Este trabalho contém uma pequena abordagem sobre dois dos mais utilizados métodos para análise utilizados no desenvolvimento de sistemas.

Visando atender ao propósito estabelecido, os assuntos são abordados de modo a possibilitar a contextualização do assunto a ser desenvolvido, bem como a obtenção do objetivo pretendido.

Na Seção 2 é feita uma breve apresentação do método de análise estruturada para o desenvolvimento de sistemas; a Seção 2.1 são enfatizadas as características das ferramentas utilizadas para a modelagem de sistemas que utilizam a análise estruturada para o seu desenvolvimento e, na Seção 2.1.1 há uma descrição sobre o diagrama de fluxo de dados, uma das mais utilizadas ferramentas de modelagem. A Seção 3 apresenta uma visão do método de análise orientado a objetos e seus componentes fundamentais, enquanto que a Seção 3.1 as ferramentas de modelagem orientadas a objeto são abordadas de maneira resumida e com pouca profundidade. A Seção 4 apresenta a UML, uma linguagem unificada de modelagem, considerada um padrão e reconhecida mundialmente. Na Seção 5 é apresentada uma comparação entre as ferramentas de modelagem utilizadas nos métodos de análise estruturada e orientado a objetos, visando, dentro do escopo definido, mostrar os pontos fortes e fracos de sua utilização e, finalmente, Seção 6 contém as observações resultantes da realização deste trabalho e aponta os problemas resultantes da proliferação de ferramentas de modelagem que dificultam o estabelecimento de um padrão único de modelagem, independente do processo de desenvolvimento utilizado e, conseqüentemente, a proposição de ferramentas CASE, para dar suporte aos desenvolvedores, em todas as atividades do processo de desenvolvimento de *software*.

2. Método de Análise Estruturada

Até o final da década de 70, a grande maioria dos projetos de desenvolvimento de sistemas começava pela criação de um documento elaborado pelo analista de sistemas, contendo uma descrição textual sobre o que ele entendia ser os requisitos do usuário. Esses documentos muitas vezes denominados de especificação funcional padeciam de alguns problemas: eram monolíticos e, portanto, de difícil compreensão, uma vez que era necessária toda a especificação funcional, para poder compreendê-la; eram redundantes, dado que a mesma informação era muitas vezes repetida em diversas partes diferentes do documento, tornando difícil a sua atualização e revisão, conduzindo a um problema grave de inconsistência; eram ambíguos, pois o detalhamento dos requisitos do usuário podia ser interpretado de modo distinto pelo analista de sistemas, pelo projetista, pelo programador e pelo próprio usuário; eram de difícil manutenção, posto que a especificação funcional quase sempre era obsoleta no final do processo de desenvolvimento do sistema, o que muitas vezes ocorria no final da fase de análise, devido aos motivos anteriormente explicitados.

Esses problemas provocaram uma reação da comunidade de engenharia de *software* que levou à percepção de que a atividade de desenvolvimento de sistema compreendesse especificações funcionais que fossem gráficas, particionadas e de redundância mínima conforme visto em Yourdon (1992).

Para DeMarco (1989) a análise tradicional de sistemas, caracterizada por especificações maciças e pesadas começou a se modificar no final da década de 70, dando lugar a uma nova abordagem denominada análise estruturada, que dizia respeito fundamentalmente a um novo tipo de especificação funcional, a especificação estruturada.

O termo “análise estruturada” foi popularizado por DeMarco (1989) quando então introduziu e nomeou símbolos gráficos que possibilitariam ao analista criar modelos de fluxo de informação, sugeriu uma heurística para o uso desses símbolos e

sugeriu que um dicionário de dados e narrativas de processamento pudesse ser usado como complemento aos modelos de fluxo de informação.

Segundo DeMarco (1989) a análise estruturada é a utilização das seguintes ferramentas: diagrama de fluxo de dados, dicionário de dados, português estruturado, tabelas e árvores de decisão. A utilização dessas ferramentas de maneira efetiva contribuiria grandemente para a solução ou minimização dos problemas inerentes à fase de análise (problemas de comunicação, a natureza variável dos requisitos do sistema, falta de ferramentas adequadas, problemas no documento de especificação funcional, dentre outros).

A análise estruturada, como todos os métodos de análise de requisitos de *software*, é uma atividade de construção de modelos. Utilizando uma notação que é própria ao método de análise estruturada, é possível criar modelos que retratem o fluxo e o conteúdo da informação (dado e controle), divide-se o sistema em partições funcionais e comportamentais e descreve-se a essência do que deve ser construído conforme pode ser verificado em Pressman (1995).

Dentre os muitos benefícios obtidos com a utilização de uma metodologia estruturada para análise e projeto de sistemas, pode-se destacar:

- os usuários obtêm uma idéia mais clara do sistema proposto, por meio dos diagramas de fluxo de dados lógico do que a obtida através de narrativa de texto e fluxogramas de sistemas físicos;
- a apresentação do sistema em termos de fluxo de dados lógicos evidencia mal-entendidos e pontos controversos mais cedo do que normalmente é o caso;
- as interfaces entre o sistema a ser desenvolvido e os sistemas automatizados e/ou rotinas manuais, já existentes, são mostradas de modo bem claro pelo diagrama de fluxo de dados e, a necessidade de documentar os detalhes dos fluxos de dados do dicionário de dados força uma definição clara dessa interface nos estágios iniciais;
- supressão de duplicação de esforço, pelo uso do modelo lógico; e
- a utilização de dicionário de dados para guardar os itens do glossário, economiza tempo na resolução de problemas relacionados à atribuição de nomes iguais para designar diferentes coisas ou diferentes nomes para designar a mesma coisa.

2.1. Características das Ferramentas de Modelagem

As características que segundo Yourdon (1992), as ferramentas de modelagem devem ter são as seguintes: ser gráficas, com adequado detalhamento textual de apoio; permitir que os sistemas sejam visualizados de forma subdividida, na modalidade *top-dow*; ter redundâncias mínimas; ajudar no prognóstico do comportamento do sistema; e ser transparente para o leitor.

A maioria das ferramentas de modelagem apóia-se em gráficos. O uso de gráficos não é obrigatório em um modelo de sistema, mas é justificável a utilização de gráficos ao invés de narrativas de texto, porque uma figura pode englobar uma imensa quantidade de informações de forma concisa e compacta, o que não significa que uma figura possa descrever, necessariamente, tudo sobre um sistema.

Para Yourdon (1992), normalmente, os gráficos são usados para identificar os componentes de um sistema e as interfaces entre eles, as demais informações são apresentadas em documentos textuais de apoio.

Outro aspecto importante de uma boa ferramenta de modelagem é a aptidão para retratar um sistema de forma subdividida *top-dow*, o que é extremamente importante quando se trata de modelar sistemas grandes e complexos, uma vez que essa abordagem permite fornecer uma visão geral de alto nível de uma parte do modelo (principais componentes de alto nível e interfaces do sistema) bem como uma visão das partes subseqüentes do modelo, enfocando informações sobre componentes detalhados de baixo nível do sistema.

2.1.1 O Diagrama de Fluxo de Dados

O diagrama de fluxo de dados (DFD) é a principal ferramenta de planejamento para um sistema de informação conforme Gane (1988), e é amplamente utilizado no método de análise e projeto estruturado, pois ele mostra a fronteira do sistema e, é muito importante porque é o único documento que mostra todas as relações entre os dados (armazéns e fluxos de dados) e os processos e funções que transformam esses dados.

O diagrama de fluxo de dados é uma ferramenta de modelagem que permite que um sistema seja visto como uma rede de processos assíncronos e funcionais, interligados por fluxos de dados e repositórios de armazenamento de dados. É uma das mais utilizadas ferramentas de modelagem de sistemas, principalmente para sistemas nos quais as funções do sistema sejam mais importantes e mais complexas que os dados manipulados pelo sistema. Em um sistema no qual os relacionamentos entre os dado sejam mais importantes que as funções, pode-se dar menos importância aos DFD's e concentrar esforços no desenvolvimento de diagramas de entidades-relacionamento. Como alternativa, se o comportamento tempo-dependente do sistema suplantar todos os outros aspectos, os diagramas de transição de estado são a melhor opção. Deve-se, porém, entender que as ferramentas acima se apresentam como alternativas, podendo ser utilizadas como opções complementares durante o processo de modelagem, não sendo, portanto, mutuamente exclusivas.

O primeiro componente de um DFD é denominado processo, também conhecido com bolha, função e transformação. O processo mostra como uma ou mais entradas são transformadas em saída. Através dos processos é possível representar todas as funções de um sistema. Na maioria dos modelos de DFD, o nome do processo descreve o **que** o processo faz, em alguns casos, conterá o nome de uma pessoa ou de um grupo de pessoas (ex: um departamento, ou divisão de uma empresa), ou um computador, ou um dispositivo mecânico, para descrever **quem** ou o **que** executa o processo, ao invés de descrever acerca do processo.

O fluxo de dados, outro componente do DFD, é utilizado para mostrar o movimento de fragmentos ou pacotes de informações de um ponto a outro do sistema. O fluxo de dados representa uma estrutura de dados dinâmica, enquanto que os depósitos de dados, ou armazéns de dados, representam uma estrutura de dados estática.

O exame de fluxos de dados que entram e saem de um depósito de dados, provoca questionamentos, a saber, (dentre outros): o fluxo representa um único pacote, múltiplos pacotes, partes de pacotes, partes de um ou de vários pacotes? O fluxo de

dados não informa sobre *prompts* de entradas e nem responde perguntas de saída. Essas e outras questões envolvem detalhes procedimentais e o DFD não trata desses aspectos.

Embora algumas das perguntas procedimentais possam ser respondidas pelo exame dos rótulos dos fluxos, nem todos os detalhes são evidentes. Para saber tudo o que se quer sobre um fluxo que parte de um depósito, tem-se que examinar os detalhes – a especificação de processos – do processo ao qual o fluxo está interligado.

Há um detalhe procedimental do qual se pode ter certeza: o depósito é passivo e os dados não transitam a partir dele pelos fluxos a menos que um processo os solicite explicitamente. Existe outro detalhe procedimental que é pressuposto, por convenção: um depósito não se altera quando um pacote de informações se movimenta a partir dele por um fluxo descrito em Yourdon (1992).

Outro componente do DFD é o terminador. Os terminadores representam as entidades externas com as quais o sistema se comunica. Existem três aspectos importantes a serem considerados sobre terminadores: 1) eles são externos ao sistema que está sendo modelado; os fluxos que interligam os terminadores aos diversos processos (ou depósitos) representam a interface entre o sistema e o mundo externo; 2) como conseqüência, nem o analista e nem o projetista de sistemas estão em condições de modificar o conteúdo de um terminador ou o modo como o terminador funciona; e 3) qualquer relacionamento existente entre terminadores não é mostrado no DFD, caso haja relacionamentos entre terminadores e for essencial que o analista de sistema os modele para documentar de forma correta os requisitos do sistema, então, por definição, os terminadores são realmente parte do sistema e devem ser modelados como processo na visão de Yourdon (1992).

Uma maneira prática de referenciar os processos de um DFD é numerar cada bolha. O importante quando se utilizar numeração, é ser consistente no modo como se atribui os números e ter claro que a numeração dos processos não presume seqüência de execução, ainda que uma certa seqüência possa ser deduzida pela presença ou ausência de dados. A numeração é útil na identificação dos processos e servem como base para o esquema de numeração hierárquico, quando do processo de decomposição do DFD em níveis inferiores.

O propósito de um DFD é modelar corretamente as funções que um sistema deve executar e as interações entre elas. Uma das maneiras de tornar um DFD geral mais compreensivo e diminuir sua complexidade é modelar um DFD geral em uma série de níveis, de modo que cada nível ofereça sucessivamente mais detalhes sobre uma parte do nível que lhe seja superior.

Para Yourdon (1992), é um erro considerar o DFD como única ferramenta de modelagem. Está claro que um DFD, sem outras ferramentas adicionais de modelagem, praticamente não tem valor.

Segundo Pressman (1995), a análise estruturada, o mais utilizado dos métodos de modelagem de requisitos, faz uso de modelos como o primeiro elemento de uma representação gráfica para um sistema automatizado. Usando como base os diagramas de fluxo de dados e de controle, o analista particiona as funções que transformam os fluxos de dados. A seguir, cria um modelo comportamental, usando o diagrama de transição de estado, e desenvolve um modelo de conteúdo de dados com um dicionário de requisitos (também chamado dicionário de dados), ferramenta proposta como uma

gramática quase formal para descrever o conteúdo de objetos definidos durante a análise estruturada.

A modelagem de dados, cuja notação é o diagrama de entidade-relacionamento, é usada em aplicações que envolvem grande volume de dados e pode ser aplicada como uma ferramenta complementar durante a análise estruturada.

3. Método de Análise Orientado a Objetos

No final da década de 1980, o "paradigma orientado a objeto" começava a amadurecer numa abordagem poderosa e prática para desenvolvimento de *software*. A partir daí a análise orientada a objeto vem fazendo um firme progresso como método de análise de requisitos como complemento a outros métodos de análise. Ao invés de examinar um problema usando o clássico modelo de fluxo de informação ou um modelo derivado exclusivamente de estruturas de informação hierárquicas, a análise orientada a objetos introduz uma série de novos conceitos na visão de Pressman (1995).

Os métodos de análise de requisitos de *software* orientados a objetos possibilitam que o analista modele um problema ao representar classes, objetos, atributos e operações como os componentes primordiais de modelagem. A abordagem orientada a objetos combina classificação de objetos, herança de atributos e comunicação de mensagens no contexto de uma notação de modelagem.

A essência da análise e do projeto orientados a objetos é enfatizar a consideração de um domínio de problema e uma solução lógica, segundo a perspectiva de objetos.

De acordo com Pressman (1995) é importante considerar que o uso de análise orientada a objeto pode levar a uma prototipagem efetiva e a técnicas de "engenharia de *software* evolucionárias". Os objetos especificados e implementados em um projeto em desenvolvimento podem ser catalogados em uma biblioteca. Os objetos são inerentemente reusáveis e, com o passar do tempo, a biblioteca de objetos reusáveis crescerá. Ao utilizar a análise orientada a objetos para o desenvolvimento de novos projetos, o analista poderá utilizar-se dos objetos já existentes durante a análise, o que reduzirá substancialmente o tempo dedicado à especificação e, um protótipo do sistema especificado pode ser criado e revisado pelo cliente/usuário.

Os objetos modelam quase todos os objetos identificáveis do domínio de problemas: entidades externas, coisas, ocorrências, papéis, unidades organizacionais, lugares e estruturas. Outro aspecto importante é que objetos encapsulam tanto dados como processo.

Segundo Thiry (2001), conceitualmente, uma **classe** é entendida como um conjunto de objetos que possuem características similares (atributos), comportamentos comuns (operações), relacionamentos comuns com outros objetos (agregações e associações) e semântica comum.

Um **objeto** é um componente do mundo real que é mapeado para o domínio de *software*. No contexto de sistemas computadorizados, um objeto (instância de uma classe) é tipicamente um produtor ou consumidor de informações ou um item de informação. Um membro (objeto) de uma classe **herda** todos os atributos definidos para a classe. Todo objeto de uma classe pode ser modificado por **operações** (ações aplicadas para mudar os atributos de um objeto ou realizar ações sobre um objeto). Um

objeto (e todos os objetos em geral) **encapsula** dados (valores dos atributos que identificam um objeto). O encapsulamento representa o empacotamento do objeto, ocultando seu estado e a implementação de suas operações. De acordo com Thiry (2001), o estado de um objeto representa uma das prováveis condições em que o objeto pode existir bem como representa os valores das propriedades de um determinado objeto em um dado momento. Encapsulamento possibilita que toda informação empacotada sob um determinado nome, pode ser reusada como uma especificação ou componente de programa.

Conforme consta em Pressman (1995), quando um objeto é representado graficamente, ele se compõe de uma estrutura de dados e processos particulares, denominados **operações** que podem transformar a estrutura de dados. As operações contêm construções procedimentais e de controle que podem ser invocadas por uma **mensagem** – uma solicitação para que o objeto execute uma de suas operações.

4. UML

Uma série de diferentes ferramentas de modelagem tem sido sugerida para a análise orientada a objetos. Todas utilizam os conceitos de orientação a objetos, mas cada uma introduz sua própria notação, heurística e filosofia.

As metodologias para análise e projeto de sistemas provavelmente continuarão a definir métodos, modelos e processos de desenvolvimento visando a construção efetiva de sistemas de *software* e, atualmente, podem fazê-lo, usando uma linguagem comum – a UML.

Para Craig (2000), a Linguagem de Modelagem Unificada (UML), é uma notação (incluindo sua semântica) já aceita como padrão que utiliza conceitos orientados a objetos para modelagem de sistemas. É uma linguagem para especificar, visualizar e construir artefatos de sistemas.

A UML padroniza artefatos e notação, mas não define um processo-padrão de desenvolvimento de *software* e, dentre as razões para isso, tem-se que: a idéia é aumentar sua aceitação como uma notação de modelagem padrão sem se comprometer com um determinado processo de desenvolvimento, dado que existe uma variação significativa sobre o que constitui um processo apropriado.

A UML compreende um conjunto de diagramas, referenciados a seguir, que propiciam a sua aplicação e a obtenção dos resultados dentro do nível de qualidade a que se propõe: diagrama de classe, de atividade, de componentes, de *deployment*, de seqüência, de colaboração, de estados e de *use case*.

De acordo com Booch (2000) a UML é independente de processo, porém, para se obter o máximo proveito de suas potencialidades, é necessário que um processo seja: orientado a casos de usos, centrado em arquitetura, iterativo e incremental.

Para Craig (2000), um diagrama de classes ilustra as especificações para as classes de *software* e de interfaces de uma aplicação, associações e atributos; interfaces, com suas operações e constantes; método, informação sobre o tipo do atributo; navegabilidade e dependências.

Para Booch (2000) um diagrama de atividade é essencialmente um gráfico de fluxo, mostrando o fluxo de controle de uma atividade para outra e, um diagrama de

componentes é uma apresentação gráfica da visão estática de implementação de um sistema que mostra a organização e as dependências existentes entre um conjunto de componentes.

Na concepção de Fowler (2000) um diagrama de *deployment* mostra as relações físicas entre componentes de *hardware* e *software* no sistema implementado.

Segundo Thiry (2001) um diagrama de seqüência mostra a interação de objetos organizada no tempo; o diagrama de colaboração mostra a interação de objetos no contexto dos objetos e suas ligações; os diagramas de estados são utilizados para modelar o comportamento de um objeto e, os diagramas de *use case* são construídos para possibilitar a visualização dos relacionamentos entre atores e *use cases*.

Para o propósito deste trabalho, será enfatizado o diagrama de *use case*, o qual será abordado a seguir.

4.1. *Use Case*

Conforme Booch (2000) um sistema interage com atores humanos ou qualquer outro agente externo que se utilizam desse sistema para algum propósito e, esses atores esperam que o sistema se comporte conforme o esperado. Um caso de uso especifica o comportamento de um sistema ou de parte de um sistema e descreve um conjunto de seqüências de ações, incluindo variantes realizadas pelo sistema para produzir um resultado observável do valor de um ator.

Ainda segundo Booch (2000) os casos de uso podem ser aplicados para captar o comportamento desejado do sistema em desenvolvimento, sem a necessidade de especificar como esse comportamento é implementado. Os casos de uso fornecem uma maneira para que os desenvolvedores, os especialistas do domínio e os usuários finais tenham uma compreensão comum sobre o problema. Um caso de uso representa um requisito funcional do sistema como um todo. Serve também, para ajudar a validar a arquitetura e para realizar a verificação do sistema à medida que este evolui durante seu desenvolvimento.

Um caso de uso descreve um conjunto de seqüências, cada uma representando a interação de itens externos ao sistema (atores) com o próprio sistema. Esses comportamentos, na realidade são funções em nível de sistema utilizadas para visualizar, especificar, construir e documentar o comportamento previsto do sistema durante a elicitação e análise de requisitos.

Para Craig (2000) um caso de uso é uma descrição textual de um processo do domínio em uma empresa ou sistema em um formato estruturado. Constituindo-se, portanto, em um documento narrativo que descreve a seqüência de eventos de um ator (um agente externo) que usa um sistema para completar um processo. Na prática, o diagrama de caso de uso pode ser expresso em variados graus de detalhes, bem como de comprometimentos com as decisões de projeto, o que significa que o mesmo caso de uso pode ser escrito em diferentes formatos e em diferentes níveis de detalhes.

5. Comparação entre a Análise Estruturada e Orientada a Objetos

Um sistema (do mundo real ou de *software*) normalmente é muito complexo, portanto, é necessário decompô-lo em partes de forma a possibilitar sua compreensão e administrar sua complexidade. Essas partes podem ser representadas como modelos

que descrevem e abstraem aspectos essenciais do sistema. Assim, é importante criar, durante a construção de *software*, modelos que organizam e comunicam os detalhes do mundo real com o sistema ao qual está relacionado, bem como do sistema a ser construído. Esses modelos devem conter elementos coesos e fortemente relacionados.

Antes do advento do método de análise e projeto orientado a objetos, a abordagem mais utilizada para a decomposição de um problema era o método de análise e projeto estruturado, no qual o particionamento é feito através de funções e processos, tendo como resultado uma decomposição hierárquica de processos compostos por sub-processos. A análise e projeto orientados a objetos enfatizam a decomposição do espaço de um problema por objetos ao invés de funções. A Figura 3 mostra as diferentes visões de decomposição por meio dos métodos de análise e projeto estruturado e orientado a objetos segundo Craig (2000).

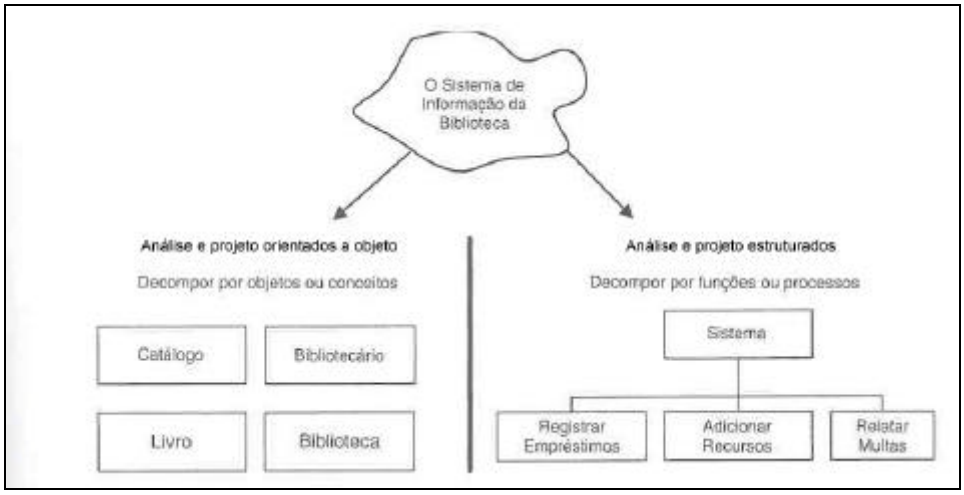

Figura 3 – Decomposição orientada a objetos versus orientada à função.

Existem diversas ferramentas de modelagem, que podem ser utilizadas para o desenvolvimento de sistemas, baseado nos métodos de análise acima referenciados.

Com o propósito de atingir o objetivo proposto neste trabalho, será feita uma comparação apenas das principais ferramentas de modelagem dos dois métodos de análise de requisitos abordados.

Na concepção de Craig (2000) a finalidade do diagrama de casos de uso é apresentar um tipo de diagrama de contexto, através do qual pode-se compreender rapidamente quais são os atores externos a um sistema e como eles o utilizam.

No caso do DFD, as entidades externas (categorias lógicas de coisas ou pessoas que representam a fonte ou destino de informação) são origem e/ou destino de informações estando, portanto, por definição, fora do sistema em consideração conforme visto em Gane (1988).

Assim, tanto a aplicação de DFD como de diagramas de casos de uso envolve a modelagem de contexto do sistema, pois ambos mostram a fronteira do sistema e as entidades externas (atores no caso de *use case*) que interagem com o mesmo.

Desse modo, entende-se que um determinado processo do diagrama de fluxo de dados (representa uma função) de um sistema pode ser visto como um diagrama de

casos de uso e que as entidades externas do diagrama de fluxo de dados podem ser mapeadas diretamente como atores em um diagrama de casos de uso.

Tanto o diagrama de fluxo de dados como de casos de uso possibilitam a visão de aspectos funcionais do sistema.

Para Pressman (1995) o DFD possibilita que o engenheiro de *software* desenvolva modelos do domínio da informação e do domínio funcional simultaneamente. Quando o DFD é refinado em níveis de detalhes, o analista executa uma decomposição funcional implícita. Ao mesmo tempo, o refinamento do DFD resulta em um refinamento dos dados à medida que eles se movimentam através dos processos.

A decomposição do DFD em níveis inferiores de detalhes é similar à criação de vários diagramas de casos de uso, dado que um diagrama de *use case* é utilizado para criar e refinar um conjunto de requisitos, o que em ambos os casos elimina a complexidade e problemas de inconsistência.

O DFD por si só não basta para realizar a modelagem de sistema baseados no método de análise estruturada e deve ser usada em conjunto com outras ferramentas, como o diagrama de entidade-relacionamento, para modelar os dados; o diagrama de estados, para modelar o comportamento do sistema, e o dicionário de dados, que contém o conteúdo dos dados.

Similarmente, um diagrama de casos de uso não é suficientemente poderoso para modelar um sistema orientado a objetos em sua totalidade, tendo estreita relação com os diagramas de seqüência e colaboração, dentre outros.

O DFD e o diagrama de casos de uso utilizam-se de narrativas de textos como apoio à especificação do processo.

Uma das críticas em relação ao DFD, diz respeito à abordagem *top-dow* exigida para a sua construção e manutenção.Para Pressman (1995) o DFD deve ser apresentado aos usuários na modalidade *top-dow*, entretanto, não é necessariamente verdadeiro que o analista de sistema deva desenvolver os diagramas de fluxo de dados segundo essa modalidade. A subdivisão de um DFD pode ser para cima (para criar DFD de níveis mais elevados) e para baixo (para criar DFD de níveis mais baixos).

Para Booch (2000) um caso de uso descreve o **que** um sistema (ou um subsistema, classe ou interface) faz, sem especificar **como** isso é feito. Ele captura o comportamento compreendido do sistema em desenvolvimento, sem precisar especificar como esse comportamento é implementado. Esse é um fator importante para "isolar" aspectos de análise (comportamento do sistema) de aspectos referentes à implementação (como esse comportamento é executado).

No DFD, embora alguns aspectos físicos de implementação possam ser visualizados, eles não são incentivados.

Kulak (2000) afirma que há uma pré-disposição na indústria para associar *use case* com sistemas orientados a objetos, mas acredita que os casos de uso podem ser usados facilmente para documentar requisitos de sistemas não orientados a objetos, entendendo-os, inclusive como um grande modo de documentar quaisquer tipos de sistemas. E defende também que os diagramas de fluxo de dados podem ser utilizados para modelar sistemas baseados nos métodos de análise orientados a objetos.

6. Conclusão

Constitui-se uma tarefa difícil e complexa a avaliação e, conseqüentemente a comparação, de ferramentas utilizadas na modelagem de sistemas que têm seu desenvolvimento baseado em métodos totalmente distintos.

A análise estruturada, como todos os métodos de análise de requisitos de *software*, é uma atividade de construção de modelos que usando uma notação que é própria ao método de análise estruturado, possibilita a construção de modelos que retratam o fluxo e o conteúdo da informação.

A análise orientada a objetos, por sua vez, ao invés de examinar um problema usando o conhecido modelo de fluxo de informação e, ao introduzir novos conceitos, possibilita que o analista modele um problema ao representar classes, objetos, atributos e operações como os componentes principais de modelagem.

Fica evidente que as duas abordagens acima referenciadas, examinam um problema a partir de distintas visões e, conseqüentemente, se utilizam de ferramentas que sejam compatíveis ao seu modo de "enxergar" o sistema a ser construído e possibilitem a criação de modelos que possam retratar o melhor possível o problema e sua solução.

Acreditamos que muitos dos aspectos que dificultam a adoção de uma determinada ferramenta de modelagem estão relacionados principalmente com o fato de: 1) os analistas estarem acostumados a desenvolver sistemas utilizando um determinado método e, para a sua modelagem, utilizam ferramentas com características voltadas a esse método, resistindo a mudanças; 2) por não deterem conhecimento suficiente para amparar tal mudança ou por simples preconceito (decorrência de desconhecimento) em relação à adoção de um novo método de análise e a conseqüente aplicação de ferramentas que suportem as atividades previstas nesse método.

Os usuários finais nem sempre têm conhecimentos suficientes que lhes permitam opinar sobre a adoção (ou a migração) de um determinado método de análise de requisitos, ficando à mercê da opinião dos desenvolvedores (analistas e engenheiros de *software*) que, apegados a abordagens "antigas" devido ao despreparo ou por resistirem às inovações, da área de engenharia de software, contribuem para que as empresas permanecem estagnadas.

Muitas empresas têm potencial e anseiam por mudanças que impliquem em melhorias de seus produtos e do processo através dos quais seus produtos são construídos, mas, por falta de orientação, acabam se acomodando aos métodos e ferramentas costumeiros.

A resistência às mudanças, não é um mito, é uma realidade. Muitas vezes, o que determina a escolha do método de análise do sistema a ser desenvolvido e as ferramentas aplicadas para a sua modelagem, é a experiência do analista e não a natureza da aplicação ou os prováveis benefícios que podem ser obtidos com a utilização de um método que seja mais adequado ao problema para o qual se busca solução.

As contribuições dos profissionais e estudiosos da área de engenharia de software, no que tange aos métodos de análise de requisitos de software, trazem, normalmente, em seu bojo, uma nova notação, o que implica em criar novas

ferramentas que possam suportar essas notações. O esforço demandado no desenvolvimento de software poderia ser minimizado com um conjunto de ferramentas automatizadas para auxiliar as atividades de desenvolvimento, entretanto, dada à diversidade de notações, torna-se difícil à construção dessas ferramentas.

Ainda que os métodos de análise de requisitos estruturado e orientado a objetos abordem o desenvolvimento de sistemas a partir de visões totalmente distintas, entendemos que muitas das ferramentas de modelagem podem ser usadas por ambos, tais como, diagramas de fluxo de dados, diagramas de entidade-relacionamento, diagramas de estados e prototipagem, sem comprometer o resultado pretendido e o nível de qualidade desejado.

Como a área de engenharia de *software* está em constante evolução, acreditamos que esse cenário persistirá imutável, enquanto houver estudiosos buscando dar contribuições que visem a melhoria dos métodos de análise de requisitos bem como das ferramentas utilizadas para modelar os sistemas baseados nesses métodos.

Está claro que o uso de um determinado método de análise e ferramentas apropriadas de modelagem, não implica em garantia de elicitar correta, adequada e completamente os requisitos do usuário. A capacidade de capturar os requisitos e traduzi-los em uma representação que o usuário possa entender, depende significativamente da experiência do analista na tarefa de desenvolvimento de sistemas e de aplicação de ferramentas que possam dar suporte às suas atividades.

Referências

Booch, G. et al. (2000) UML – Guia do Usuário. Editora Campus, Rio de Janeiro.

DeMarco, T. (1989) Análise Estruturada e Especificação de Sistemas. Editora Campus, Série Yourdon Press, Rio de Janeiro.

Fowler, M. e Scott, K. (2000) UML Essencial. 2ª ed., Porto Alegre, Bookman.

Gane, C. (1988) Desenvolvimento Rápido de Sistemas. Livros Técnicos e Científicos Editora LTDA, Rio de Janeiro.

Gane, C. e Ssarson, T. (1983) Análise Estruturada de Sistemas. Livros Técnicos e Científicos Editora LTDA, Rio de Janeiro.

Larman, C. (2000) Utilizando UML e Padrões. Porto Alegre:Bookman.

Kulak, D. (2000) USE CASES – *Requirements in Contex*t. ADDISON-WESLEY, New York.

PRESSMAN, R. S. (1995) Engenharia de *Software*. MAKRON *Books* do Brasil, São Paulo.

Thiry, M. e Salm Jr, J. (2001) Processo de Desenvolvimento de *Software* com UML, 2º período/2001. [disponível on-line: http:// www.eps.ufsc.br/disc/procuml].

Yourdon, E. (1992) Análise Estruturada Moderna. Editora Campus, Série Yourdon Press, Rio de Janeiro.