Uma Proposta de Evolução em Sistemas Legados

Luciana de Paiva Silva[1], Victor F.A. Santander[2]
[12]*Universidade Estadual do Oeste do Paraná*
Rua Universitária, 2.069 - Jd. Universitário
Cascavel /PR - CEP 85814-110
[1]*lucianapaiva@unipan.br*
[2]*vfasantander@unioeste.br*

Resumo - O processo de evolução de sistemas legados é um tópico de recentes pesquisas na área de engenharia de requisitos. Organizações vêm se defrontando continuamente com a necessidade de mudar e/ou melhorar seus sistemas computacionais. Neste processo de evolução, as maiores mudanças envolvem a transição do uso de metodologias tradicionais tais como Análise Estruturada e Essencial para metodologias Orientadas a Objetos tais como Catalysis e Processo Unificado (UP). É consensual que um processo de evolução de sistemas legados não deve se basear unicamente na avaliação do código fonte, mas também nos demais artefatos que possam facilitar o processo de elicitação, análise e documentação de requisitos. Neste contexto, uma das principais dificuldades reside na falta de diretrizes que permitam mapear informações de sistemas legados representadas em modelos tais como Diagramas de Fluxos de Dados (DFDs) para novos artefatos utilizados em metodologias orientadas a objetos, como Casos de Uso em UML. Neste artigo, apresentamos um breve estudo das principais metodologias de desenvolvimento de software utilizadas atualmente tanto no âmbito acadêmico quanto industrial, bem como propomos um conjunto de diretrizes para apoiar desenvolvedores no processo de evolução de sistemas legados mapeando informações contidas em DFDs e código fonte para Casos de Uso em UML.

Palavras-chave: Análise, Engenharia Reversa, Modelagem, Engenharia de Requisitos.

1. Introdução

O desenvolvimento de sistemas computacionais de pequeno e médio porte é apoiado por inúmeras técnicas, métodos e processos, a fim de facilitar e padronizar o seu desenvolvimento. Processos de desenvolvimento de software, comumente, não são seguidos e compreendidos adequadamente, sendo negligenciados em pontos fundamentais para o sucesso do projeto de software.
Escolher e executar adequadamente processos de desenvolvimento de software, são aspectos fundamentais para desenvolver produtos de software de qualidade [1].

Gerentes de projeto deparam-se com a difícil tarefa de escolher a melhor abordagem a ser aplicada no desenvolvimento de um determinado projeto de software.
Mais recentemente, a Engenharia de Requisitos [2] tem surgido para melhorar o processo de desenvolvimento de software em um dos pontos cruciais para a obtenção de um produto de qualidade: a correta elicitação, análise, especificação e validação de requisitos. É de consenso tanto da comunidade acadêmica quanto industrial que um produto de software de qualidade é aquele que é capaz de atender os requisitos funcionais e não-funcionais dos respectivos usuários [3,4].
Assim, verifica-se neste contexto, que a qualidade de um produto de software é fortemente influenciada pela qualidade do processo utilizado. Melhorar o processo de desenvolvimento é um aspecto fundamental para a melhoria da qualidade do produto. Fornecer software de qualidade não é apenas uma questão de funcionalidade visual. É necessário atentar para a documentação do mesmo, a facilidade para rastrear e avaliar impactos de mudanças nos requisitos, verificar a satisfação dos requisitos não-funcionais, entre outros aspectos críticos. Este sentimento é também expresso em [5], no qual são identificados os problemas mais comuns em projetos de software como falhas na elicitação e análise de requisitos e impossibilidade completa de manutenção, devido principalmente à falta ou inadequada documentação.
Estes problemas agravam-se ao observarmos que, em muitos casos, desenvolvedores de software encontram-se em contextos de trabalho, nos quais há a necessidade de atualizar e/ou reconstruir sistemas legados, juntamente com a difícil tarefa de evoluir de metodologias tradicionais tais como abordagem estruturada [6] e essencial [10] para metodologias mais modernas e atuais, como UP (Processo Unificado) [2,7]e Catalysis[8]. As dificuldades neste processo são inúmeras. É importante considerar todas as informações úteis e que possam ser aproveitadas a partir da documentação existente do sistema legado. Propomos neste artigo, um conjunto de diretrizes que visam facilitar o processo de transição em sistemas legados, e mais especificamente, diretrizes que permitem mapear importantes informações modeladas em Diagramas de Fluxos de dados (DFDs) para Casos de uso em UML.
A definição destas diretrizes é motivada pela grande quantidade de empresas que ainda utilizam metodologias tradicionais e necessitam migrar para abordagens orientadas a objetos, sem saber como nem quais informações podem ser mapeadas neste processo de evolução.
Desta forma, este artigo discute, inicialmente, os principais processos e metodologias de desenvolvimento de software existentes atualmente. Em seguida, propõe-se um conjunto de diretrizes para apoiar a evolução natural que muitas empresas se defrontam em relação à transição do uso de métodos tradicionais estruturados para o uso de metodologias e processos orientados a objetos, como UP e Catalysis. Este artigo está estruturado como se segue. Na seção 2 investiga-se aspectos essenciais de cada uma das metodologias. Na seção 3 é apresentado o conjunto de diretrizes para auxiliar engenheiros de requisitos na transição do uso de metolodogias. Na seção 4, apresentamos as dificuldades encontradas no uso das diretrizes propostas e na seção 5 as considerações finais, bem como trabalhos futuros.

2. Usando Metodologias de Desenvolvimento de Sistemas

Independente da metodologia escolhida para a construção de um produto de software, a mesma deve possuir uma semântica bem-definida, de modo que, qualquer outro desenvolvedor possa interpretar e ser capaz de entender os propósitos do sistema sem ambigüidades.
A metodologia deve também propiciar e nortear o desenvolvimento de forma que o produto final de software possua características fundamentais relacionadas a qualidade do produto em si. Algumas destas características e sub-características de software são definidas em normas e modelos, como por exemplo na norma ISO/IEC9126[9].
Ao definirmos a metodologia a ser utilizada devemos ter em mente que a abordagem de cada uma difere em relação a conceitos e processos a serem utilizados, sem contar que a experiência do analista é fator primordial para o sucesso de qualquer análise.
Muitas suposições são feitas quando tratamos a respeito de metodologias e seus benefícios e vantagens em relação a outras já existentes. O que se observa na prática, é uma constante evolução decorrente de experiências pessoais de um grupo de pessoas. Essa evolução tem levado a criação de abordagens de desenvolvimento de software mais sistemáticas, bem definidas e preocupadas em integrar efetivamente os usuários no processo de desenvolvimento de software, e de forma mais critica, no processo de engenharia de requisitos. Busca-se também desenvolver softwares mais coesos, de fácil gerenciamento, passíveis de certificação e com um custo menor. Recentemente, processos tais como o Processo Unificado e Catalysis têm sido propostos para desenvolver softwares com estas características.
Neste contexto, se observarmos que sistemas legados precisam evoluir e em muitos casos essa evolução implica em mudança de paradigma. A seguir fazemos um breve relato das principais metodologias de desenvolvimento de sistemas existentes. Em seguida, propomos um conjunto de diretrizes que visam facilitar o trabalho de engenheiros de requisitos no processo de evolução de sistemas legados, quando há a necessidade de mudança de abordagens de desenvolvimento.

2.1 Análise Estruturada de Sistemas

Em meados dos anos 70 o problema da especificação dos requisitos verdadeiros começou a atrair séria atenção. Em 1975, o primeiro trabalho foi publicado por Douglas T. Ross e Kennety E. Schoman Junior, Técnicas de Projeto de Análise Estruturada ou SADT. Este trabalho foi um grande passo à frente na tecnologia de definição de requisitos, por ter sido a primeira abordagem a propor um conjunto prático de ferramentas de modelagem gráfica que superaram as deficiências evidentes da especificação narrativa [7].
O maior problema com o SADT é que Ross e Choman não definiram claramente o processo de desenvolvimento de uma especificação de requisitos. Em 1977, Gane e Sarson [5], melhoraram o SADT oferecendo uma estratégia rudimentar para construir um modelo de requisitos de um sistema baseado em dados. Na verdade, sua proposta era a utilização adicional das ferramentas de modelagem da análise estruturada: utilização específica do diagrama do fluxo de dados em combinação com o dicionário de dados e a descrição de processos e procedimentos.
Estas técnicas, e mais especificamente a Análise Estruturada, tiveram como objetivo principal auxiliar o analista a determinar se requisitos verdadeiros foram omitidos ou

se houve introdução de preferências ou influências tecnológicas. Porém, seus autores não foram felizes em fornecer informações suficientes e orientações conceituais para que os analistas pudessem realizar esta distinção.
Na verdade, este ponto crucial e de extrema importância foi fator importante para a maioria dos analistas da época, pois apenas a experiência ditava como proceder.
Assim, todo o processo de desenvolvimento de sistemas por esta técnica era baseado em:

- Definir os fluxos de dados recebidos pelo sistema – entradas;
- Definir os fluxos de dados fornecidos pelo sistema – saídas; e
- Definir os dados que devem ser armazenados e os processos que devem ser executados para transformar os fluxos de entrada nos fluxos de saída

2.2 Análise Essencial de Sistemas

A Análise Essencial ou Análise Estruturada Moderna tem por finalidade fornecer uma declaração dos requisitos verdadeiros do sistema [10]. Este requisito verdadeiro, também conhecido como requerimento essencial ou lógico é uma característica que o sistema deve ter, qualquer que seja a tecnologia utilizada para implementá-lo. Atualmente, estes requisitos receberam a denominação de Requisitos Funcionais.
A Análise Essencial de Sistemas tentou resolver o problema principal que existe na maior parte dos projetos de software, a má formulação destes requisitos, bem como sua distinção entre a visão lógica e física.

Com isso, todo o processo de desenvolvimento de sistemas constituia-se dos seguintes aspectos:

- Definir requisitos essenciais ou lógicos;
- Identificar os eventos – estímulos, aos quais o sistema deve responder para atingir ao seu propósito;
- Identificar os dados que devem ser armazenados e a atividade que deve ser executada para que o sistema responda completamente ao evento.

2.3 Processo Unificado

O Processo Unificado – UP , em comparação com outros processos conhecidos, é bastante completo em sua definição [2].

O UP encaixa-se na definição geral de processo: um conjunto de atividades executadas para transformar um conjunto de requisitos do cliente em um sistema de software. Porém, é uma estrutura genérica de processo que pode ser customizado adicionando-se ou removendo-se atividades com base nas necessidades específicas e nos recursos disponíveis para um projeto orientado a objetos [2].

Seus princípios chave são: *dirigido a casos de uso* (ações executadas por um ou mais atores), *centrado na arquitetura (*considerada o alicerce fundamental sobre o qual o UP se erguerá) e *iterativo e incremental* (uma iteração é um miniprojeto que resulta em uma versão do sistema liberada interna ou externamente).

O UP subdivide o desenvolvimento de software em quatro grandes **fases** denominadas de **Concepção, Elaboração, Construção e Transição**. Estas fases também são

marcos de referência (“milestones”) no desenvolvimento, sendo que cada fase possui alguns objetivos e conteúdo associados[2].
No processo iterativo e incremental, os *stakeholders* (o conjunto de sistemas, grupos ou indivíduos que são afetados pelo sistema de software) acompanham e participam de cada iteração no desenvolvimento de software. Esta integração permite que mudanças ou adaptações necessárias ao sistema sejam mais facilmente acomodadas e que problemas mais críticos no desenvolvimento possam ser descobertos em iterações e fases iniciais do desenvolvimento. Outro aspecto positivo da abordagem iterativa é que a mesma é **orientada a riscos**, sendo que em cada iteração os riscos de desenvolvimento são avaliados e decisões podem ser tomadas, permitindo um controle gradual dos riscos durante a evolução do sistema. Consideramos também que o processo iterativo e incremental lida com uma realidade freqüentemente ignorada, na qual usuários não conseguem definir todos os requisitos de um sistema completamente no início do processo. Iterações sucessivas permitem uma evolução gradual.

2.4 Processo Catalysis

A Análise e Projeto orientado a objetos utilizam os mesmos conceitos básicos para descrever o domínio do usuário e o software. No Catalysis, estes conceitos básicos são objetos, representando um *cluster* (menor unidade de informação) de informação e funcionalidade, e uma ação, representando um evento, tarefa, trabalho, mensagem, mudança de estado, interação, ou atividade. No Catalysis, a ação é colocada no mesmo nível de objeto, porque um bom projeto requer cuidado sobre as ações ocorridas e o que elas executam [8].
Diferente de alguns métodos de orientação a objeto, o Catalysis não se inicia sempre determinando responsabilidades de ações para objetos específicos. Primeiramente determina-se o que acontece e, em seguida, determina-se qual objeto é responsável pela ação e qual é responsável pela sua inicialização. Finalmente determina-se como será efetuada a ação.
O Catalysis utiliza o *Refinamento,* como um meio de mostrar objetos e ações em diferentes escalas. Alguns atores podem ser peças de um software, como um sistema de escalonamento, por exemplo. O Catalysis utiliza o conceito de programação orientada a objeto, na qual unidades podem ser encapsuladas organizadamente em objetos ou classes. Cada objeto possui uma série de tarefas que executam dados que são utilizados para detalhar estas tarefas em grandes escalas. O Catalysis considera os casos essenciais do projeto como os mesmos em todas as escalas, mostrando variações no nível de detalhamento.

3. O Processo de Transição em Sistemas Legados

Muitas são as contribuições relacionadas a atualização de sistemas legados em relação a uma nova metodologia [12,13,14,15,16]. Fontanette et al [13], relata em seu artigo que os sistemas legados possuem lógica de programação, decisões de projeto, requisitos de usuário e regras de negócio, que podem ser recuperados, facilitando sua reconstrução, sem perda da semântica. Assim, abandoná-los e começar o projeto do zero seria uma perda considerável.

Nas próximas seções descreve-se as diretrizes que apontamos como decisivas para realizar a transição entre metodologias, visando melhorar/atualizar os sistemas legados.

3.1 Treinamento de Pessoal

Este treinamento requer que a equipe que irá realizar esta difícil tarefa, tenha algumas qualidades que julgamos serem essenciais:

a) Conhecimento da nova metodologia(Catalysis e RUP, na maioria dos casos);
b) Conhecimento de ferramentas CASE que facilitem o trabalho de migração;
c) Excelente ambiente de trabalho, não só em relação ao grau de relacionamento do grupo como também em relação a um local apropriado para o desenvolvimento do trabalho, sem precisar dividir o ambiente de trabalho com várias pessoas de setores diferentes, por exemplo.

Estas qualidades são fundamentais para que o processo de migração seja mais efetivo e produtivo.

3.2 Organização da Base de Informações

A organização da base de informações é decisiva para facilitar a engenharia reversa [17] de um sistema legado utilizando outra metodologia. As seguintes situações podem ser observadas:

a) Um sistema sem documentação – neste caso, estão disponíveis apenas as informações do código fonte do sistema;
b) Um sistema com documentação atualizada – Neste caso, se temos uma documentação em ordem, o processo é facilitado, uma vez que podemos verificar a análise efetuada anteriormente e as atualizações que se tornaram necessárias com o passar dos anos. Temos um histórico completo do sistema, simplificando a organização.
c) Um sistema com uma documentação não atualizada – Nesta situação bastante comum, as equipes mal tiveram tempo de realizar as atualizações nos códigos fontes do sistema. Normalmente, as equipes que trabalham no sistema não foram as mesmas que o desenvolveram.

Em qualquer das situações acima, devemos ter em mente que a equipe que irá realizar a tarefa de evolução de sistemas legados utilizando uma nova metodologia, não será a mesma que o desenvolveu. Assim, normalmente precisamos nos basear em alguns pontos cruciais para o desenvolvimento do projeto, conforme segue:

- Deve-se integrar efetivamente, o usuário/cliente no processo de desenvolvimento. A integração e interação constante de usuário/cliente durante todo o processo de transição do uso de metodologias modificando e reconstruindo sistemas legados é ponto decisivo para o sucesso do processo.
- Deve-se considerar efetivamente os requisitos que o sistema deve atender (ou requisitos verdadeiros). Os requisitos do sistema são o que o sistema produz ou emite em forma de relatórios e consultas visuais em tela para os respectivos usuários. O que o sistema produz justifica a sua existência e a sua essência.

Preocupar-se com a satisfação dos requisitos de clientes e usuários é uma questão chave no processo de transição adotado.

Para compreendermos melhor este processo, tomemos como base os três tipos diferentes de situações apresentadas em relação a sistemas legados (baseados na documentação existente) e suas respectivas ações a serem desenvolvidas, de acordo com a Tabela 2.

Neste exemplo, supomos a transição da metodologia da Análise essencial ou estruturada para uma metodologia de Orientada a Objetos, como RUP ou Catalysis. Cabe ressaltar que ambas as metodologias orientadas a objetos consideram os casos de uso como elementos chaves e essenciais no processo de desenvolvimento.

<table>
<tr><th colspan="3">Sistemas Legados</th></tr>
<tr><th colspan="3">Situação da Documentação</th></tr>
<tr><th rowspan="2">Inexistente</th><th>Precária</th><th>Atualizada</th></tr>
<tr><th>Análise Estruturada/Análise Essencial</th><th>Implementado na Análise Estruturada</th></tr>
<tr><td rowspan="4">Fases:
1- Analisar código fonte do Sistema;
2- Analisar E/S do Sistema
3- Validar com os Stakeholders as E/S apontadas na Fase 2.</td><td rowspan="4">Fases:
1 – Atualizar DFDs - Analisar DFDs existentes e compará-los com a análise efetuada no código fonte;
2 – Realizar a Conversão de DFDs em Casos de Uso;
3 – Analisar E/S do Sistema
4 –Validar com os Stakeholders as E/S apontadas na Fase 3.
5 – Organizar Casos de Uso</td><td>Fases:
1 – Realizar a Conversão de DFDs em Casos de Uso;
2 – Analisar E/S do Sistema
3 –Validar com os Stakeholders as E/S apontadas na Fase 2.
4 – Organizar Casos de Uso.</td></tr>
<tr><td>Implementados na Análise Essencial</td></tr>
<tr><td>Fases:
1 – Analisar Diagrama de Contexto;
2 – Realizar a Conversão de DFDs em Casos de Uso;
3 – Analisar E/S do Sistema
4 – Validar Stakeholders apontados na Fase 1 e as E/S apontadas na Fase 3.
5 – Organizar Casos de Uso.</td></tr>
</table>

Tabela 2. Passos a serem seguidos na realização de engenharia reversa em sistemas legados.

Observando a Tabela 2, podemos visualizar as análises e comparações a serem efetuadas em sistemas legados considerando a documentação existente.

É importante observar que relatos na literatura apontam para o consenso de que a análise do código fonte e sua organização, auxilia o engenheiro de software a

remodelar o sistema sob o enfoque orientado a objetos [13,14,15,16]. Contudo, pouca atenção se dá as outras possibilidades de uso de informações, como por exemplo, o mapeamento de modelos e diagramas que representam, tipicamente, os aspectos essenciais de um sistema. Desta forma, pode-se enriquecer a coleta de informações necessária a atualização/reconstrução de sistemas legados. Modelos tais como DFDs podem efetivamente ser mapeados para casos de uso em UML, facilitando o trabalho de transição entre metodologias.
A seguir apresentamos um conjunto de diretrizes que permitem mapear Diagramas de Fluxos de Dados para casos de uso em UML. As diretrizes a serem propostas podem ser aplicadas para mapear elementos de metodologias tradicionais (Estruturada ou Essencial), para o Processo Unificado ou para o Catalysis, uma vez que nosso principal enfoque será obter os casos de uso referentes ao sistema, os quais são elementos chaves nos dois processos orientados a objetos.

3.3 Mapeando elementos de Diagramas de Fluxos de Dados para Casos de Uso em UML.

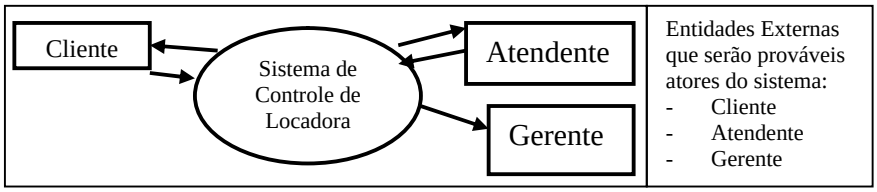
Um dos maiores problemas atuais no desenvolvimento de software está relacionado ao processo de transição entre metodologias. Atualmente, muitas organizações são obrigadas a mudar seus sistemas computacionais utilizando novas abordagens de desenvolvimento, as quais apresentam algumas vantagens tais como: maior flexibilidade, suporte automatizado para todo o processo de desenvolvimento, etapas e artefatos bem definidos, bem como possibilidade de uso efetivo do paradigma orientado a objetos. Contudo, na maioria das situações, existem informações importantes nos sistemas legados que devem ser mapeadas para novos artefatos gerados no uso de novas metodologias.
Neste contexto, verificamos que tipicamente Diagramas de Fluxos de Dados desenvolvidos utilizando metodologias tradicionais precisam ser mapeados de forma mais efetiva e sistemática para a técnica de casos de uso. É importante ressaltar que casos de uso fazem parte da UML e são considerados essenciais na maioria dos processos de desenvolvimento orientados a objetos atualmente utilizados. Não é correto assumir que apenas o código fonte em sistemas legados é suficiente para elicitar e documentar todos os requisitos essenciais de sistemas computacionais objetos de transição e evolução. É necessário atentar para toda documentação disponível que possa auxiliar na transição e evolução de sistemas de software.
Neste sentido, propomos a seguir, um conjunto de diretrizes que visam auxiliar desenvolvedores no processo de mapeamento entre os artefatos gerados nas metodologias tradicionais e artefatos gerados nas metodologias orientadas a objetos. O foco das diretrizes está no mapeamento de Diagramas de Fluxos de Dados (DFDs) para Casos de Uso em UML. Após a descrição das diretrizes, apresentamos uma tabela que mostra a aplicabilidade das diretrizes para cada situação, tomando como base a documentação disponível dos sistemas legados.
Estas diretrizes são dependentes exclusivamente da situação documental disponível, de acordo com a Tabela 2, podendo ser aplicadas, dependendo da situação da documentação do software legado, conforme descrito na Tabela 3.

Situação da Documentação Existente	Diretrizes a serem implementadas

	A	B	C	D	E
Inexistente				X	X
Precária	X		X	X	X
Atualizada – Estruturada			X	X	X
Atualizada – Essencial		X	X	X	X

Tabela 3. Uso das diretrizes propostas de acordo com a situação documental disponível.

a) Diretriz A – *Análise do Diagrama de Contexto* (DFD nível 0) - Esta diretriz tem por objetivo relacionar as entidades externas como prováveis atores do sistema legado. Esta informação deverá *ser* validada com a Diretriz E. Nessa diretriz todas as entidades externas serão relacionadas como possíveis atores do sistema.



Quadro 1. Diretriz A – análise do diagrama de contexto

b) Diretriz B – *Conversão do Diagrama de Fluxo de Dados para Diagrama de Casos de Uso*

b.1)Subdiretriz B1 - Para cada DFD de *ultimo nível* do sistema legado será elaborado um Caso de Uso, como exemplificado na Figura 1.

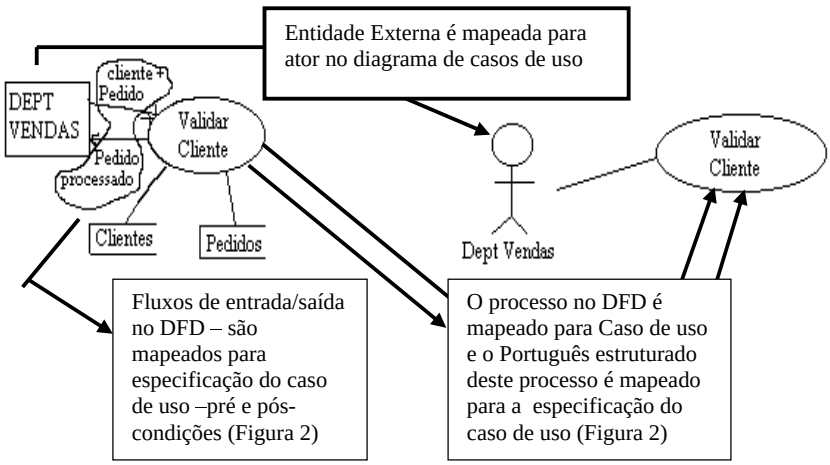


Figura 1. Conversão DFD para diagrama de caso de uso.

b.2) Subdiretriz B2 - Analisar a especificação do processo do DFD (Figura 3). Caso o DFD não possua Entidade Externa, ele é o resultado de um outro processo anterior que o originou. Neste caso, deve ser definido apenas como um

caso de uso do tipo <<include>> ou <<extend>> , o qual será chamado em outro caso de uso. Devemos ter o cuidado de validá-lo com o DFD que o chamou, e assim incluí-lo no processo de conversão (Figura 2 e Figura 3). Normalmente, a especificação do processo indica quais são os passos necessários para se atingir um determinado objetivo. Deste modo, pode-se a partir da especificação do processo analisado já incluir no caso de uso correspondente as colaborações e generalizações indicadas de acordo com a Figura 3.

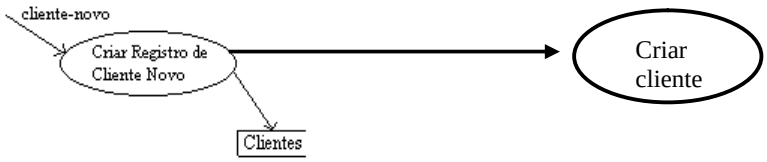


Figura 2. Conversão DFD para caso de uso – especial

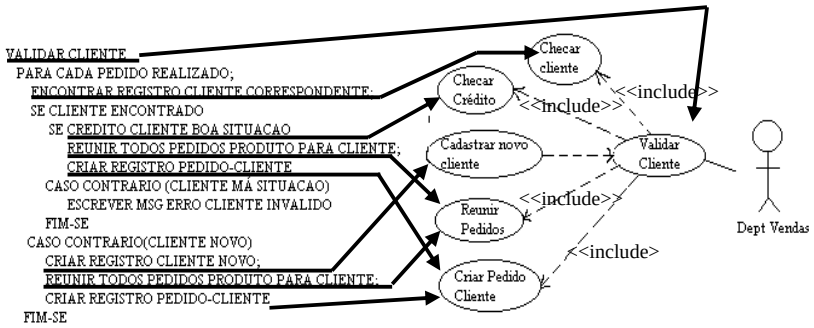


Figura 3. Análise e conversão da especificação do processo *Validar Cliente*.
Fonte: Código do Português Estruturado [6].

b.3) Subdiretriz B3 – Analisar os fluxos de entrada e saída do DFD, bem como a especificação do processo do DFD. Este passo é importante para validar os depósitos de dados identificados no DFD (Figura 1, 3 e Tabela 4).

Vejamos, por exemplo, a Tabela 4. Nesta tabela o caso de uso Validar Cliente é descrito com base nas informações contidas na especificação do processo do DFD (Figura 3). Definimos o objetivo do caso de uso analisado, sua prioridade, indicando se sua execução é essencial ao sistema, as pré-condições existentes para sua execução, as pós-condições após a sua execução e o cenário principal juntamente com os fluxos secundários, indicando alternativas de execução. Descreve-se também os caso de uso pai e subordinados, os quais são detectados no decorrer da conversão dos DFDs.

No passo 1 do cenário principal (*O caso de uso inicia Quando DeptVendas requer a validação do* cliente) devemos indicar o responsável pela execução do mesmo. Apesar da especificação do processo do DFD não indicar o ator, o DFD possui como Entidade Externa o DeptVendas, logo, ele é incluído como ator do Caso de Uso. Nos

outros passos é realizada uma referência a especificação do processo, como por exemplo, em "*Se cliente encontrado"* no DFD, que foi representado pelo passo 2 "*O sistema verifica a existência do cliente"*.
Os fluxos de exceção são tratados pelos casos de uso secundários (include ou extend), associados ao caso de uso principal – Validar Cliente.

Caso de Uso	Validar Cliente
Objetivo	Este caso de uso descreve o procedimento para que um cliente seja validado de acordo com sua situação decorrente do crédito
Prioridade	(x)Essencial ( ) Secundário
Pré-Condições	Dados do cliente e Pedido do cliente incluído .
Pós-Condições	Pedido do Cliente Processado.
Cenário Principal	1. O caso de uso inicia Quando DeptVendas requer a validação do cliente 2. O sistema verifica existência do cliente (caso de uso checar cliente é incluído) 3. O sistema verifica o crédito do cliente (caso de uso checar crédito é incluído) 4. O sistema reune todos os pedidos do cliente (caso de uso reunir pedidos é incluído) 5. O sistema cadastra todos os pedidos do cliente (caso de uso criar pedido cliente é incluído)
Fluxos Secundários	
Caso de Uso Pai	
Caso de Uso Subordinado	Checar cliente Checar crédito Reunir pedidos Criar pedido

Tabela 4. Especificação do Caso de Uso.

c) Diretriz C – *Organização das Entradas e Saídas do Sistema* – esta diretriz tem por objetivo relacionar e validar as entradas e saídas do sistema. Este passo requer o auxílio da equipe ou responsável **pela** manutenção do sistema, que indicará para cada entrada (em tela) e saída (relatório existente) o usuário responsável pela informação. Deste modo, o engenheiro de software terá uma relação de todos os atores (*stakeholders*) que são afetados.
d) Diretriz D – *Validação dos Stakeholders* – esta diretriz tem por objetivo validar os stakeholders apontados pela diretriz D. Deste modo, o engenheiro de software deve comparar com os atores já identificados pela diretriz B e realizar as alterações necessárias nos casos de uso identificados.
e) Diretriz E – *Organizar Casos de Uso* – esta diretriz tem por objetivo organizar os diagramas de caso de uso de acordo com as notações propostas pela UML. Casos de uso devem ser **agrupados** em pacotes ou pela especificação de relacionamentos de generalização, inclusão e extensão, existentes entre os mesmos [4].

4. Principais Dificuldades Presentes no Processo de Transição entre Metodologias

A experiência já ditava regras desde a Análise Estruturada, uma vez que apenas os analistas experientes tinham conhecimento para discernir o lógico do físico, pois os poucos que conseguiam aprender a fazer esta distinção, não tinham sucesso em passar esta aptidão aos colegas [7].
Na prática, o que verificamos é que, com pouquíssimas exceções, as pessoas não conseguem aprender uma nova metodologia de desenvolvimento com tanta facilidade. Para muitos analistas, uso do UP ou Catalysis envolve diversas dificuldades técnicas e de origem pessoal.
A realidade de nossas empresas, tanto privadas, quanto estatais ou do governo, como é o caso do Exército Brasileiro, está começando a mudar. A experiência de um dos autores do presente artigo no Exército Brasileiro mostra que grupos de desenvolvimento de sistemas de Brasília, Capital Federal, se encontram num dilema entre a transição de metodologia, pois ainda utilizam a Análise Estruturada. A transição tem se mostrado lenta e gradual buscando um só objetivo: o desenvolvimento e migração de sistemas existentes e a desenvolver. As diretrizes propostas neste artigo são fruto da experiência de desenvolvimento dos autores, bem como das dificuldades encontradas no âmbito da análise de sistemas do Exército Brasileiro.

5. Considerações Finais e Trabalhos Futuros

Apresentamos, inicialmente, neste artigo, um relato das principais metodologias de desenvolvimento de sistemas existentes, buscando facilitar o estudo dos fatores envolvidos no processo de migração de projetos elaborados através da Análise Estruturada ou a Análise Essencial para projetos elaborados utilizando processos e metodologias Orientadas a Objeto.
Com base neste estudo inicial, propomos um conjunto de diretrizes para auxiliar engenheiros de software/requisitos no processo de transição entre metodologias tradicionais e orientadas a objetos. Basicamente, propõe-se de forma sistemática, mapear as informações existentes na documentação do processo tradicional para o processo de desenvolvimento orientado a objetos. Apoiamo-nos no fato de que as informações presentes em DFDs podem auxiliar o processo de construção e descrição de casos de uso em UML. Defende-se a idéia de que as diretrizes propostas devem ser aplicadas de acordo com a documentação disponível em relação aos sistemas legados sendo tratados.
Verificamos neste trabalho que o processo de transição de abordagens de desenvolvimento tradicionais para as utilizadas na orientação a objetos implica, inicialmente, em uma mudança de cultura bastante complexa dentro da empresa. Consideramos também que outros aspectos importantes tais como rastreamento de requisitos, modelagem organizacional e tratamento de requisitos não-funcionais, não considerados na análise estruturada e essencial de sistemas, devem ser inseridos gradualmente na equipe de desenvolvimento aumentando progressivamente o grau de maturidade dessas organizações.

Ressaltamos que as diretrizes propostas foram elaboradas com base em experiências, principalmente, no processo de análise de sistemas da Marinha do Brasil e do Exército Brasileiro, do qual um dos autores participa há mais de 12 anos.
Nossos estudos iniciais apresentados neste artigo servem de base para a elaboração de uma proposta mais ampla na qual pretendemos:

a) Associar a modelagem organizacional com a modelagem funcional [18] da empresa com base na documentação existente dos sistemas legados;
b) Desenvolver uma ferramenta que suporte a conversão do DFDs para Casos de Uso usando as diretrizes propostas neste artigo, pois como já mencionamos, a técnica de casos de uso é chave na UML, bem como nos principais processos de desenvolvimento Orientado a Objetos.

6. Referências Bibliográficas

[1] Filho, T. L. "*Metodologia de Desenvolvimento de Sistemas"* , Axcel Books, Rio de Janeiro,2003.
[2] Scott, Kendall. *"O Processo Unificado Explicado*". Bookman,, Porto Alegre,2003
[3] Chung, L.; *Nixon*, B.; Yu,Eric;Mylopoulos,John. "*Non-Functional requirements in Software engineering"*, USA: Kluwer Academic Publishers, 2000,439p.
[4] Booch, G.; Rumbaugh, J., Jacobson, I. *"The Unified Modeling Language"* , Addison-Wesley,1999.
[5] Shlaer,S.; Mellor,S. "*Análise de Sistemas Orientada para Objetos",* Makron Books, São Paulo,1990.
[6] Gane, C.; Sarson,Trish. "Structured Systems Analysis*"* , New York, Improved System Technologies, 1977.
[7] Jacobson, I.; Booch, G.; Rumbaugh, J. *"The Unified Software Development Process"* , Addison-Wesley (1999).
[8] D'Souza, D. and A.Wills. "*Objects, Components and Frameworks with UML: The Catalysis Approach",* USA: Addison-Wesley, (1995).
[9] ISO International Standard Organization. ISSO 9000 – Quality Management system. 1.ed. Geneva: ISSO 2000.
[10 McMenamin, S.; Palmer,J. F. "*Análise Essencial de Sistemas"* , Makron Books, São Paulo,1984.
[11] Toranzo, Marco A., "*Uma Proposta para Melhorar o Rastreamento de Requisitos de Software*", Centro de Informática, Universidade Federal de Pernambuco, Tese de Doutorado, Dezembro, (2002).
[12] Fukuda, A. P. "*Refinamento Automático de sistemas Orientados a Objetos Distruidos*" São Carlos/SP, 2000. Dissertação de Mestrado. Universidade Federal de São Carlos
[13] Fontanete,V. et al. "Reengenharia de Sistemas Legados Baseada em Componentes usando Transformações". São Carlos/SP,2003. Artigo. Universidade Federal de São Carlos.
[14] Jesus,E.S. *"Engenharia Reversa de Sistemas Legados Usando Transformações*". São Carlos/SP, 2002, Dissertação de Mestrado, Universidade Federal de São Carlos.
[15] Werner, C.M.L.; Braga, R. M.M. "*Desenvolvimento Baseado em Componentes"***.** XIV Simpósio Brasileiro de Engenharia de Software SBES2000 Minicursos e Tutoriais – pg. 297-329 – 2-6 de Outubro, 2000.
[16] Jesus, E. S. "*Engenharia Reversa de Sistemas Legados Usando Transformações"*, 2000 - Dissertação de Mestrado, Ciências da Computação, UFSCAR.
[17] Chikofsky, E. "*Reverse Engineering and Design Recovery – A Taxonomy*". IEEE Software
[18] Santander, Victor F.A. *"Integração de Modelagem Organizacional com Modelagem Funcional na Engenharia de Requisitos"*. Centro de Informática, Universidade Federal de Pernambuco, Tese de Doutorado, Dezembro, (2003).