# Avaliação das vantagens quanto à facilidade de manutenção e expansão de sistemas legados sujeitos à engenharia reversa e segmentação

Maria Istela Cagnin[1]
Rosângela A. D. Penteado

Universidade Federal de São Carlos
Departamento de Computação
C.P. 676 - 13565-905 - São Carlos (SP)
e-mail: istela@icmc.sc.usp.br
rosangel@dc.ufscar.br

## Resumo

*Dois processos de reengenharia com mudança de paradigma de desenvolvimento do procedural para o orientado a objetos são apresentados. A documentação orientada a objetos utilizada foi recuperada em um processo de engenharia reversa com apoio da abordagem Fusion/RE. Segmentação é um dos processos de reengenharia no qual a linguagem de programação do sistema legado é preservada com a adição de características orientadas a objeto. No outro processo, a linguagem de programação é alterada de C para Java e a forma de armazenamento, de arquivos-texto para banco de dados relacional, Sysbase, além da utilização de padrões de projeto. Nesse segundo processo de reengenharia foram experimentadas três variantes do padrão de projeto Persistece Layer para realizar a persistência dos dados usando o banco de dados relacional. O sistema legado utilizado é o ambiente StatSim, para edição e simulação de statecharts, desenvolvido em linguagem C e XView, com aproximadamente 30K LOC. Esse ambiente foi submetido, totalmente, ao primeiro processo de reengenharia, segmentação, e parcialmente ao segundo. Métricas para avaliação do esforço com a realização desses processos são analisadas.*

**Palavras-chave:** Sistema Legado, Engenharia Reversa, Reengenharia, Padrões de Projeto, Orientação a Objetos.

## *Abstract*

*Two processes of reengineering with change of the procedural development paradigm to object-oriented are shown. The object-oriented documentation utilized was obtained by reverse engineering with the Fusion/RE approach. Segmentation is one reengineering process in which the programming language is preserved and object-oriented characteristics are aditioned. In the other process, the legacy system programming language is changed from C to Java and the data storing from text files to Sybase database, besides utilizing design patterns. In this second reengineering process, three variants of a design pattern for providing data persistency are used. The StatSim environment, for editing and simulation statecharts, has been taken as the legacy system. It has been originally development in C and XView with about 30K LOC. It has been submitted in its full extent to the first reengineering project and partially in the second. Metrics for the evaluation of the effort spent with these processes are analyzed.*

**Keywords:** Legacy System, Reverse Engineering, Reengineering, Design Patterns, Object-Oriented.

## 1. Introdução

Das fases do ciclo de vida do software a manutenção é a mais temida, pois nela é necessário entender e alterar o código fonte de sistemas, na maioria das vezes desenvolvido por outras pessoas, com diferentes estilos de programação. A substituição desses sistemas é

[1] Financiado pela FAPESP. Proc. 97/12208-0

inviável tanto financeira quanto operacionalmente, uma vez que os usuários estão acostumados a operá-los e uma possível substituição pode gerar transtornos, além de um certo tempo de treinamento e adaptação ao novo sistema.

Pensando em amenizar esses problemas, a técnica de engenharia reversa foi desenvolvida para recuperar documentação atualizada do sistema a partir do código fonte. Essa documentação pode melhorar a manutenibilidade e apoiar a realização de processos de reengenharia, que visam alterar parcialmente ou não a funcionalidade original do sistema, o paradigma de desenvolvimento e a linguagem de programação.

Entre as diversas abordagens de engenharia reversa orientada a objetos existentes, Fusion/RE [13] gera documentação de análise orientada a objetos a partir de sistemas desenvolvidos originalmente com orientação a procedimentos.

Segmentação é um processo de reengenharia que preserva, do sistema legado, a linguagem procedural, adicionando características orientadas a objetos, quando possíveis. A documentação obtida com a engenharia reversa é utilizada para auxiliar o engenheiro de software a reagrupar as linhas de código para adição dessas características.

Neste trabalho dois processos de reengenharia são conduzidos: um de segmentação e o outro com a mudança da linguagem de programação (de C para Java) e da forma de armazenamento (de arquivos-texto para banco de dados relacional Sybase [16]). Para isso, o sistema exemplo utilizado é o ambiente StatSim, para edição e simulação de statecharts [11], originalmente implementado em C e com armazenamento de dados em arquivos-texto. A documentação produzida quando esse sistema foi submetido ao processo de engenharia reversa com Fusion/RE [13,14] é utilizada nos dois processos de reengenharia.

No processo de segmentação todo o código legado foi submetido à reengenharia. Assim, tem-se o sistema implementado em linguagem C com características de orientação a objetos e a mesma funcionalidade que o sistema legado.

No outro processo, a reengenharia foi realizada somente na parte do código legado correspondente à edição de estados e transições default e 1 para 1. Como existem diferenças entre os paradigmas relacional e orientado a objetos, o padrão *Persistence Layer* [18], que facilita o mapeamento de objetos para banco de dados relacional, foi também utilizado nesse processo. Essa forma de reengenharia foi realizada visando à avaliação do processo de segmentação.

As versões do sistema obtidas antes e após esses processos serviram de base para a realização de experimentos a fim de avaliar a melhoria da manutenibilidade e legibilidade em sistemas legados submetidos a esses processos de reengenharia. Essa avaliação foi realizada por meio de métricas aplicáveis ao código fonte tanto em linguagem procedural como em orientada a objeto.

Esse conjunto de informações constitui um elenco de opções disponíveis aos usuários que possuem sistemas desenvolvidos com orientação a procedimentos e que estão preocupados em fazer sua reengenharia para orientação a objetos.

Este trabalho está organizado da seguinte forma: na Seção 2 é apresentado o processo de segmentação e na Seção 3, o de reengenharia com a mudança da linguagem de implementação e da forma de armazenamento dos dados. Na Seção 4, a avaliação dos processos de reengenharia é realizada por meio de métricas específicas e de um estudo piloto, na Seção 5 considerações sobre a manutenibilidade do ambiente StatSim são apresentadas e, finalmente, na Seção 6, resultados e conclusões são discutidos.

## 2. O Primeiro Processo de Reengenharia

A Figura 1 exibe os passos desse primeiro processo de reengenharia, chamado de segmentação. Embora tenham sido propostos com base na linguagem de programação C,

podem ser adaptados para outras linguagens procedurais. A realização dos passos foi apoiada pelo comando utilitário *grep*, por recursos do editor de texto, por ferramentas disponíveis no sistema operacional Unix, e pela documentação produzida na engenharia reversa [13,14].

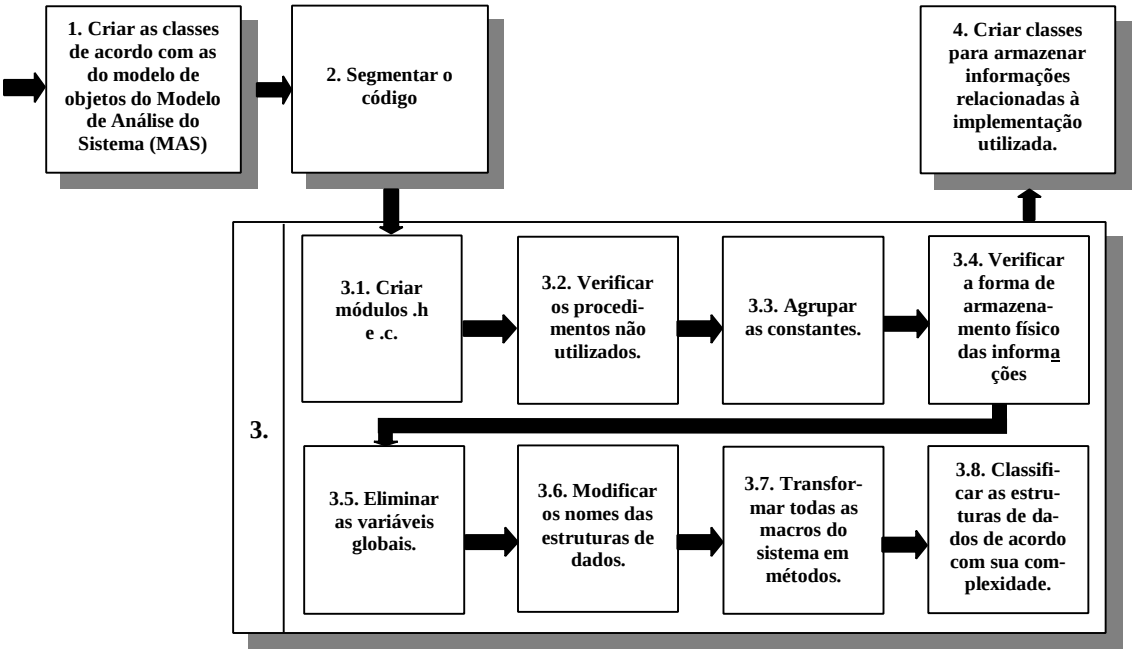



**Figura 1 – Passos para realizar a segmentação**

As informações a seguir complementam aos passos de mesmo número apresentados na Figura 1.

**1.** Uma coluna, com o nome do arquivo em que o procedimento se encontra no sistema legado, é adicionada ao quadro de correspondência entre o MAS e o MASA [13], elaborado quando da realização da engenharia reversa com Fusion/RE. Os procedimentos com e sem anomalias são migrados, como métodos, para as classes a que pertencem, de acordo com o modelo de objetos do MAS e o quadro mencionado anteriormente. Referências a esses procedimentos devem ser colocadas nas classes que os utilizam.

**2.** A segmentação do código inicia-se com a divisão de um procedimento em vários métodos. A partir da classificação dos procedimentos em métodos, na fase de engenharia reversa, nesse passo determina-se qual dos métodos é o principal, isto é, aquele que inicia a operação. O código legado é reagrupado de acordo com as estruturas de dados que o procedimento modifica ou consulta. Assim, se o procedimento modifica duas estruturas de dados (classes), a priori, é dividido em dois métodos. Cada método então irá manipular somente uma classe, aquela a que está associado. Muitas vezes essa correspondência não é tão direta, pode ser necessária a criação de métodos intermediários, como o de leitura e/ou o de atribuição de dados, para que o encapsulamento dos dados não seja violado.

**3.4.** Verificar como as informações estão armazenadas fisicamente, se em arquivos-texto, se em banco de dados relacional ou orientado a objetos, etc.. Em seguida, se necessário, realizar as modificações quanto à forma de armazenamento.

**3.8.** Uma estrutura é simples se possui elementos de dados com tipos primitivos e/ou com outras estruturas simples; as demais estruturas são consideradas complexas. A partir dessa classificação, as estruturas de dados simples são migradas para uma única classe enquanto que cada estrutura complexa é migrada para uma nova classe do sistema.

**4.** São criadas classes para manter atributos e/ou métodos relacionados com o tipo de implementação utilizado, por exemplo: componentes de interface, árvores, listas, pilhas, filas, etc.

Nesse processo de reengenharia o código fonte é alterado gradativamente, quando são adicionadas características de orientação a objetos. Conceitos como herança e polimorfismo não podem ser implementados em linguagens procedurais. Maiores detalhes e exemplificação sobre a aplicação completa desses passos ao ambiente StatSim podem ser encontrados em [3,5,9].

## 3. O Segundo Processo de Reengenharia

Este processo de reengenharia altera a linguagem de programação de C para Java e a forma de armazenamento de arquivos-texto para banco de dados relacional Sybase. Surge problema quando se deseja utilizar uma linguagem orientada a objetos e um banco de dados relacional, pois existem algumas incompatibilidades entre esses dois paradigmas. Uma solução para esse caso é a utilização do padrão *Persistence Layer* (PL) [18], que procura isolar a camada da aplicação da camada de persistência e utiliza uma classe abstrata, chamada `PersistentObject`, em sua implementação.

Os três modos para implementar o padrão PL, propostos em [18], são:

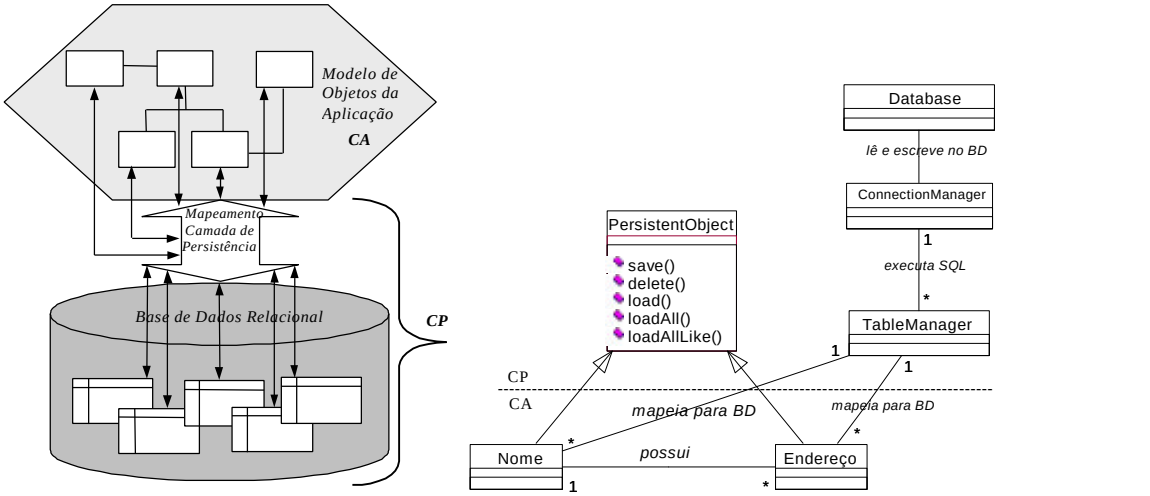
1. implementar as operações do padrão CRUD nas classes da aplicação que possuem tabelas correspondentes no banco de dados. Essas classes são herdeiras da classe abstrata `PersistentObject`;
2. criar classes específicas para cada classe da aplicação, que possui tabela correspondente no banco de dados, para implementar as operações do padrão CRUD. Essas classes específicas são herdeiras da classe abstrata `PersistentObject`;
3. criar uma classe *Broker* [2] para implementar métodos que realizem o mapeamento de qualquer tipo de objeto para o banco de dados. Essa classe deve criar automaticamente comandos SQL's para cada tipo de objeto que se deseja recuperar ou atualizar no banco de dados. As classes da aplicação, que possuem tabelas correspondentes no banco de dados, são herdeiras da classe abstrata `PersistentObject` e invocam os métodos da classe *Broker*.

As operações do padrão CRUD são implementadas por meio dos métodos públicos `save`, `delete`, `findall` e `findlike`. Esses métodos invocam métodos específicos (`insertDB`, `updateDB`, `deleteDB`, `findallDB`, `findlikeDB`) para manipular informações no banco de dados relacional. Os métodos específicos devem ser privados, ou seja, somente eles podem alterar informações no banco de dados e utilizam o padrão *SQL Code Description* para realizar a persistência dos objetos no banco de dados.

Além dos métodos que implementam as operações do padrão CRUD há também outros métodos que mapeiam um registro recuperado do banco de dados para um objeto, chamados `setDBtoObject` e `setObject`. Esses métodos possuem visibilidade privada e pública, respectivamente. Como a persistência é realizada em um banco de dados, há necessidade de fornecer um atributo, que é a chave primária, por meio dos métodos `getlast` e `getlastDB`.

A Figura 2 ilustra os recursos de implementação utilizados neste processo de reengenharia do ambiente Statsim conforme descritos e apresentados em [3]. As classes de aplicação foram mapeadas para tabelas no banco de dados relacional Sybase. Nesse mapeamento foram utilizados alguns dos padrões que compõem o padrão PL: CRUD (operações Create, Read, Update, Delete), *SQL Code Description* (descrições em comandos SQL) e *Connection Manager* (estabelece e encerra uma conexão com o banco de dados). O padrão *Table*

*Manager*, classe *Broker*, que realiza o mapeamento de qualquer objeto da aplicação para o banco de dados é utilizado somente no terceiro modo de implementação.




| Legenda: CP: Camada de Persistência | CA: Camada de Aplicação |
|---|---|

**Figura 2- Recursos de implementação utilizados no segundo processo de reengenharia**

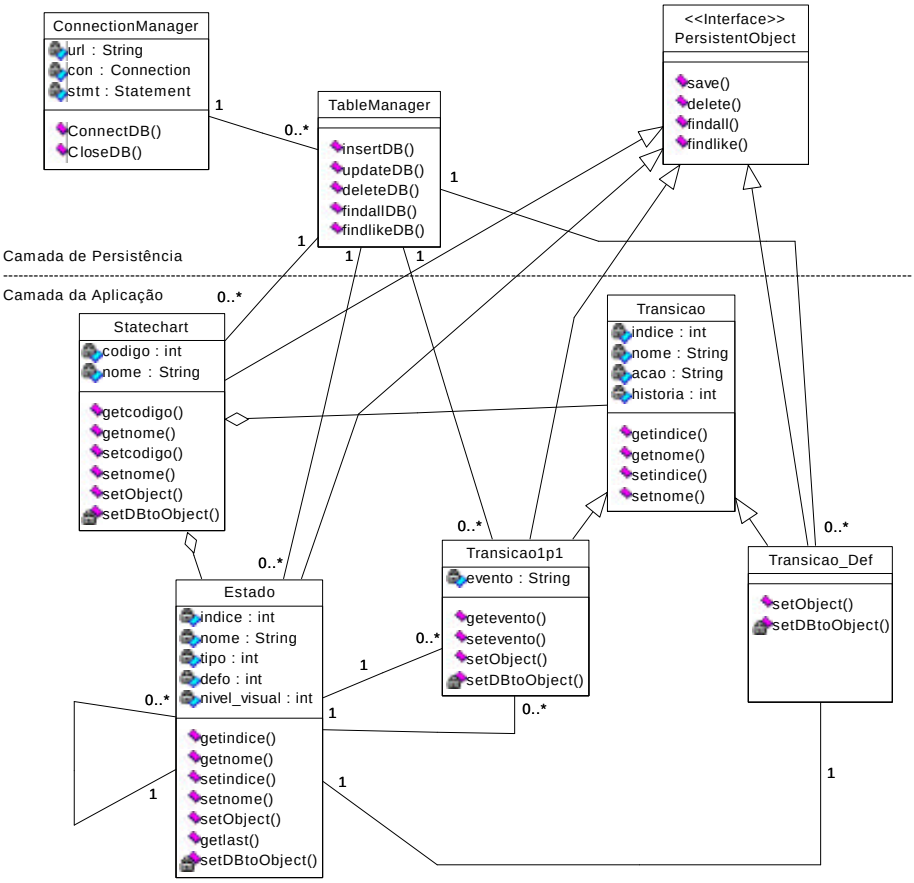



**Figura 3- Modelo de Classes do Terceiro Modo de Implementação do padrão PL**

Devido à restrição de espaço não são exibidos os Modelos de Classes e o código fonte correspondente a cada um dos modos de implementação do padrão PL. Esses podem ser obtidos em [3,4,6,7,8]. Para ilustrar o processo de reengenharia realizado, a Figura 3 apresenta o Modelo de Classes do terceiro modo de implementação do padrão PL.

Diferenças existentes entre os modos de implementação do PL: no primeiro modo os métodos específicos, aqueles que contêm comandos SQL que manipulam o banco de dados, são implementados nas classes persistentes da aplicação, aquelas que são mapeadas como tabelas no banco de dados; enquanto que no segundo modo eles são implementados nas classes da camada de persistência, criadas para cada classe persistente da aplicação; no terceiro modo, os métodos específicos são implementados na classe *Broker* (`TableManager`, existente somente nesse modo). Alguns atributos são criados, nas classes persistentes da aplicação, para possibilitar o mapeamento dos objetos para o banco de dados.

O Quadro 1 fornece alguns detalhes quanto à utilização do padrão PL, nos três modos sugeridos por Yoder [18].

**Quadro 1 - Considerações dos modos de implementar o padrão PL**

| | **Primeiro Modo** | **Segundo Modo** | **Terceiro Modo** |
|---|---|---|---|
| **utilização do padrão PL** | parcial, ou seja, padrões CRUD, *Connection Manager* e *SQL Code Description.* | parcial, ou seja, padrões CRUD, *Connection Manager* e *SQL Code Description.* | parcial, ou seja, padrões CRUD, *Connection Manager*, *SQL Code Description* e *Table Manager.* |
| **desenvolvimento** | menos trabalhoso que o Segundo e o Terceiro Modo. | mais trabalhoso que o Primeiro Modo e menos que o Terceiro Modo. | mais trabalhoso que o Primeiro e o Segundo Modo. |
| **reuso dos métodos específicos** | parcial, pois alterações são realizadas nos comandos SQL's. | parcial, pois alterações são realizadas nos comandos SQL's. | total, pois os métodos específicos, implementados na classe *Broker*, são construídos genericamente. |
| **modificação da forma de armazenamento** | mais trabalhosa que a do Segundo Modo e menos que a do Terceiro. | menos trabalhosa que a do Primeiro Modo e menos que a do Terceiro. | mais trabalhosa que a do Primeiro e do Segundo Modo. |
| **quantidade de classes da camada de persistência** | menor que a do Segundo e Terceiro do Modo. | maior que a do Primeiro e do Terceiro Modo. | menor que a do Segundo Modo e maior que do Primeiro. |
| **facilidade de uso** | mais difícil que o Terceiro Modo e igual ao Segundo Modo. | mais difícil que o Terceiro Modo e igual ao Primeiro Modo. | mais fácil que o Primeiro e o Segundo Modo. |

## 4. Avaliação dos Processos de Reengenharia

Propriedades do sistema legado e das versões obtidas após a reengenharia do ambiente StatSim são quantificadas nesta seção, com a utilização de métricas de software para código procedural e orientado a objetos. As facilidades e dificuldades de manutenção nos sistemas após essas reengenharias também são comentadas, com base em experimentos realizados por dois tipos de programadores: os que participaram desses processos de reengenharia e os que não participaram e, também, não conheciam o sistema.

Esta seção também apresenta os resultados de um estudo piloto realizado para que se possa fazer o planejamento de um experimento que dimensiona a melhoria da manutenibilidade com significância estatística.

### 4.1 Avaliação da Segmentação

A atividade de medição é realizada quanto ao tamanho e complexidade do sistema. As métricas selecionadas para serem aplicadas ao código procedural quanto ao tamanho são: **LOC** (quantidade de linhas de código fonte), **LCom** (quantidade de linhas de comentários), **Com** (quantidade de comentários), **LBr** (quantidade de linhas em branco); e quanto à complexidade são: **FAN-IN** (quantidade de procedimentos que chamam um determinado procedimento), **FAN-OUT** (quantidade de procedimentos que são chamados por um determinado procedimento). Essas métricas foram aplicadas para cada procedimento do sistema legado e para cada método do sistema segmentado. A ferramenta computacional KDSI [15], foi utilizada para aplicação das métricas de tamanho**,** enquanto que as métricas de complexidade foram aplicadas manualmente.

O tempo para a realização da segmentação foi quantificado em relação a todos os fatores que afetaram esse processo, entre outros, mudança do nome de cada procedimento para método(s), transformação de um procedimento em método(s), etc.. O tempo total gasto para realizar a segmentação foi de 6 meses, ou seja, aproximadamente 100 dias ou 600 horas, sendo que o engenheiro de software, por ela responsável, não participou do desenvolvimento do sistema legado.

A partir da aplicação e análise das métricas de complexidade observou-se que após a segmentação houve uma diminuição do valor da métrica **FAN-IN** e um aumento da métrica **FAN-OUT**. Essa diminuição é devida à criação de métodos com funcionalidade única. O aumento ocorreu pois com a adição de características orientadas a objetos ao código fonte houve a necessidade de chamar vários métodos de diversas classes para realizar a mesma tarefa que no código legado era feita com a chamada de um único procedimento que manipulava diversas estruturas de dados.

Pôde-se observar que o número de métodos criados para o encapsulamento de variáveis estáticas é maior do que o criado para o encapsulamento de atributos. Isso se deve ao elevado número de variáveis globais que o programa possuía e que foram transformadas em variáveis estáticas.

### 4.2 Avaliação da Reengenharia que altera a linguagem e a forma de armazenamento dos dados

As métricas utilizadas para avaliar, manualmente, o código implementado em linguagem orientada a objetos, Java, são as propostas por Chidamber e Kemerer [10]:

i) **WMC**: quantidade de métodos de uma determinada classe, considerando-se que as complexidades dos métodos são iguais,

ii) **DIT**: profundidade da árvore de herança, isto é, quantas classes ancestrais podem afetar potencialmente uma determinada classe. Se o seu valor for igual a 1 afirma-se que os testes nas classes não são considerados complexos e o potencial de reuso dos métodos herdados é baixo;

iii) **NOC**: quantidade de subclasses imediatas subordinadas à classe na hierarquia de classes, isto é, quantas subclasses estão herdando os métodos da classe pai. Quanto menor o seu valor menor é o potencial de reuso das subclasses e,

iv) **CBO**: quantidade de outras classes a que uma determinada classe está acoplada.

O Quadro 2 mostra os valores obtidos após a aplicação das métricas para as versões do ambientes StatSim após os dois processos de reengenharia. Apenas as métricas **WMC** e **CBO** foram aplicadas ao código em Java e às partes correspondentes do sistema segmentado em C,

já que as demais métricas quantificam características de herança das classes que as linguagens procedurais não possuem.

**Quadro 2 – Valores das métricas aplicadas às versões do ambiente StatSim**

| Métricas | Segmentado | Java 1º Modo | Java 2º Modo | Java 3º Modo |
|---|---|---|---|---|
| **WMC** | 280 | 214 | 259 | 236 |
| **CBO** | 25 | 3 | 6 | 6 |
| **DIT** | ----- | 1 | 1 | 1 |
| **NOC** | ----- | 4 | 4 | 4 |

Observando-se o Quadro 2, nota-se que a versão em Java do primeiro modo de implementação do padrão *Persistence Layer* possui valor total da métrica **WMC** inferior aos valores das outras versões, pois nessa versão a edição do estado AND não foi implementada e algumas consistências não foram consideradas. A redução no valor dessa métrica do segundo para o terceiro modo deve-se à criação da classe *Broker*. O mais alto valor das métricas **WMC** e **CBO** para a versão segmentada, em relação às demais versões em Java, ocorre devido à implementação gráfica nela existente e não nas demais. O aumento do valor dessa métrica no segundo modo é causado pela utilização das classes da camada de persistência criadas para cada classe persistente da aplicação. Mais detalhes sobre a aplicação e análise das métricas em todas as versões do ambiente StatSim podem ser encontradas em [3]. O valor encontrado para a métrica **NOC**, indica que há baixo potencial de reuso dos métodos e isso é confirmado pelos valores obtidos para a métrica **DIT**.

## 4. Avaliação da Variação da Manutenibilidade do Ambiente StatSim

A variação da manutenibilidade do ambiente StatSim, antes e após os processos de reengenharia, foi avaliada por meio de experimentos de manutenção nas versões implementadas existentes, C, C segmentado e os três modos de implementação do padrão PL em Java. Ressalta-se que a atividade de manutenção realizada foi a mesma na parte existente em todas as versões e que esses experimentos são simplesmente demonstrativos dos que podem ser conduzidos para avaliar a manutenibilidade de um sistema. Essa atividade constou da eliminação de nomes repetidos para os estados que compõem os statecharts.

O Quadro 3 mostra a participação de cada programador no desenvolvimento das versões, e a versão pela qual iniciou a manutenção. Esses experimentos foram realizados por programadores que conheciam C. Somente o Prog5 conhecia bem Java, sendo que os Prog1 e 4 não conheciam, e os Prog2 e 3 conheciam um pouco. O Quadro 4 exibe os resultados obtidos com a realização de uma manutenção em todas as versões.

**Quadro 3 - Características dos programadores que realizaram a manutenção**

| **Programador** | **Versão em que iniciou a manutenção** | **Participação no desenvolvimento** |
|---|---|---|
| **Prog. 1** | Legado | não |
| **Prog. 2** | Java 2º Modo | não |
| **Prog. 3** | Java 1º Modo | não |
| **Prog. 4** | Segmentado | não |
| **Prog. 5** | Java 3º Modo | Segmentado; Java 1º Modo, 2º Modo e 3º Modo |

**Quadro 4 – Tempo médio gasto na realização da manutenção**

| **Versões** | **Tempo Médio** | **Documentação Existente** |
|---|---|---|
| Legado –C | 11:44 | Não |
| Segmentado –C | 6:12 | Sim |
| Java – 1º Modo | 3:40 | Sim |
| Java – 2º Modo | 2:54 | Sim |
| Java – 3º Modo | 2:32 | Sim |

Com as informações existentes no Quadro 4 pode-se observar que:

1) a versão do sistema legado foi a que consumiu maior tempo de manutenção em relação às outras, devido à falta de documentação. O tempo de manutenção do sistema segmentado foi praticamente a metade em relação ao do legado;
2) a versão, em Java, do primeiro modo, consumiu menor tempo de manutenção que a do sistema segmentado; pois o padrão PL foi utilizado, fornecendo considerável nível de reuso;
3) a versão, em Java, do segundo modo, consumiu menos 1/3 do tempo de manutenção que o do primeiro modo, devido à semelhança de implementação;
4) a versão, em Java, do terceiro modo, consumiu menor tempo de manutenção que os dos dois modos anteriores, devido à utilização da classe *Broker* para implementar os métodos.

## 6. Resultados e Conclusões

Este trabalho realizou diversos experimentos de reengenharia em um sistema real a fim de estabelecer diretrizes a serem seguidas pelos interessados em substituir ou "atualizar" os sistemas existentes.

No processo de reengenharia, chamado de segmentação, há aumento do número de linhas de código fonte em relação ao do sistema legado, mas os métodos são menores e mais fáceis de serem entendidos, pois estão associados a uma única classe e possuem nomes mais significativos. As linhas de código fonte, utilizadas para construir a interface do sistema, antes espalhadas, foram agrupadas em métodos. Nesse processo notou-se a degradação do desempenho do sistema devido ao encapsulamento de dados realizado, porém ela não foi considerada devido à melhoria do entendimento e da manutenibilidade do código fonte do sistema.

O processo de reengenharia realizado, que muda a linguagem de programação e a forma de armazenamento, utilizou os três modos de implementar o padrão PL, que facilita o mapeamento de objetos para banco de dados relacional.

Com a implementação do padrão *Persistence Layer* foi possível avaliar e validar a segmentação realizada. Apenas o primeiro e o segundo modo de implementação do padrão PL também podem ser implementados com características orientadas a objetos em uma linguagem de programação procedural. A herança da classe abstrata `PersistentObject` é dispensável considerando que os métodos que implementam a persistência podem ser migrados para classes específicas, criadas para implementar a camada de persistência.

A melhoria da manutenibilidade do ambiente StatSim foi avaliada por meio de experimentos empíricos. A partir desses experimentos observou-se que há fortes indícios de que em sistemas segmentados a manutenibilidade é superior à de sistemas legados e inferior à de sistemas desenvolvidos em linguagem orientada a objetos e que utilizam padrões de projeto em seu desenvolvimento. Mas pode-se afirmar que os experimentos realizados forneceram apenas uma indicação da melhoria da manutenibilidade dos sistemas antes e após os processos de reengenharia, pois foram apenas uma demonstração dos experimentos que podem ser conduzidos para realizar a avaliação efetiva da manutenibilidade, uma vez que, para se obter resultados completos, seguros e concretos é necessário aplicar experimentos planejados com técnicas apropriadas.

**Referências Bibliográficas**


**[1]** **Java Technology.** URL: http://www.java.sun.com

**[2]** **Buschmann, Frank; et al. -** *Pattern-Oriented Software Architeture.* In: European Conference on Object-Oriented Programming, 11, Finland. Proceedings. 1997.

**[3]** **Cagnin, M. I.** – *Avaliação das vantagens quanto à facilidade de manutenção e expansão de sistemas legados sujeitos a engenharia reversa e segmentação*. Dissertação de Mestrado – Departamento de Computação, UFSCar. São Carlos, 101 p. 1999.

**[4]** **Cagnin, M. I.; Penteado, R. D.; Masiero, P. C. –** *Reengenharia com o uso de Padrões de Projeto*. In: Simpósio Brasileiro de Engenharia de Software, SBES'99, 13, Florianópolis-Santa Catarina. Anais, p. 273-288, Outubro 1999.

**[5]** **Cagnin, M.I; Penteado, R. –** *Alternativas de Reengenharia para um Sistema Implementado em C*. In: Conferência Latino-Americana de Informática, CLEI'99, 25, Assunção-Paraguai. Anais, p. 577-588, Setembro 1999.

**[6]** **Cagnin, M. I.; Penteado, R. A. D.** – *Um Primeiro Modo de Implementação do padrão Persistence Layer por meio de um processo de reengenharia*. Documento de Trabalho – Departamento de Computação, UFSCar. São Carlos, 63 p. 1999.

**[7]** **Cagnin, M. I.; Penteado, R. A. D.** - *Um Segundo Modo de Implementação do padrão Persistence Layer por meio de um processo de reengenharia*. Documento de Trabalho - Departamento de Computação, UFSCar. São Carlos, 86 p. 1999.

**[8]** **Cagnin, M. I.; Penteado, R. A. D.** - *Um Terceiro Modo de Implementação do padrão Persistence Layer por meio de um processo de reengenharia*. Documento de Trabalho - Departamento de Computação, UFSCar. São Carlos, 86 p. 1999.

**[9]** **Cagnin, M. I.; Penteado, R. D. –** *Passos para condução de reengenharia de sistemas procedurais para sistemas orientados a objetos, preservando a linguagem de implementação original*. Documento de Trabalho – Departamento de Computação, UFSCar. São Carlos, 18 p. 1999.

**[10]** **Chidamber, S. R.; Kemerer, C. F. -** *A Metrics Suite for Object-Oriented Design*. IEEE Transactions on Software Engineering, v. 20, n. 6, p. 476-493. 1994.

**[11]** **Harel, D. –** *STATECHARTS: A Visual Formalism to Complex Systems.* Science of Computer Programming, v. 8, p. 231-274. 1987.

**[12]** **Henry, S.; Kafura, D. –** *Software Structure Metrics Based on Information Flow.* IEEE Transactions on Software Engineering, v. SE-7, n. 5, p. 510-518. 1981.

**[13]** **Penteado, R. A. D. -** *Um Método para Engenharia Reversa Orientada a Objetos*. Tese de Doutorado – Instituto de Física de São Carlos, USP. São Carlos, 237 p. 1996.

**[14]** **Penteado, R.; Germano, F.; Masiero, P. C.** – Na Overall Process Based on Fusion to Reverse Engineering Legacy Code. In: Working Conference Reverse Engineering, 3, Monterey-California. Proceedings. IEEE, p. 179-188. 1996**.**

**[15]** **Renaud, B.** URL: http://www.cs.umd.edu/users/cml/cmetrics.

**[16]** **SyBase Corporate.** URL: http://www.sybase.com.

**[17]** **Rational Corporation.** *Unified Modeling Language.*
URL: http://www.rational.com/uml/references.

**[18]** **Yoder, J.W.; Johnson, R.E.; Wilson, Q.D. –** *Connecting Business Objects to Relational Databases.* In: Conference on the Pattern Languages of Programs, 5, Monticello-IL, EUA. Proceedings. 1998.