UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

JORGE VALDIR ALCHIERI

APLICAÇÃO DO PRINCÍPIO DA SEPARAÇÃO DE INTERESSES NO DESENVOLVIMENTO DE PROCEDIMENTOS ARMAZENADOS EM BANCO DE DADOS ORACLE

TRABALHO DE CONCLUSÃO DE CURSO

MEDIANEIRA
2015

JORGE VALDIR ALCHIERI

APLICAÇÃO DO PRINCÍPIO DA SEPARAÇÃO DE INTERESSES NO DESENVOLVIMENTO DE PROCEDIMENTOS ARMAZENADOS EM BANCO DE DADOS ORACLE

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas - TADS - da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Dr. Claudio Leones Bazzi

MEDIANEIRA

2015

RESUMO

Este trabalho tem como objetivo evidenciar através da aplicação de métricas de código fonte e do princípio da separação de interesses a eficiência na utilização dos procedimentos armazenados em banco de dados Oracle. Para isso será desenvolvido um estudo teórico de todos os itens necessários para que se possa obter um resultado de eficiência na utilização dos procedimentos. Posteriormente serão apresentados alguns procedimentos fazendo uso da separação de interesses e aplicadas as métricas sobre os mesmos para no final obter o resultado da eficiência no uso da separação de interesses em procedimentos armazenados em banco de dados Oracle.
Palavras chave: Acoplamento, coesão, métricas de código-fonte.

ABSTRACT

This work aims to show through the application source code metrics and the principle of separation of concerns the effcient use of stored procedures in Oracle database. For it will develop a theoretical study of all the necessary items so that we can get a result of efficient use of procedures. After that will be presented some procedures making use of separation of concerns and applied metrics on them to get the result at the end of the efficient use of separation of concerns in stored procedures in Oracle database.

Keywords: Coupling, cohesion, source code metrics.

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

ACC	Afferent Connections per Class
ANPM	Average Number of Parameters per Method
COF	Couplin Factor
NOA	Number Of Attributes
PL/SQL	Procedural Language/Structured Query Language
SC	Structural Complexity
SQL	Structured Query Language

SUMÁRIO

1 INTRODUÇÃO

Com os avanços dos softwares, é preciso lidar cada vez mais com a necessidade de uma melhor utilização dos princípios de engenharia de software para uma maior compreensão destes sistemas. Com o passar do tempo, estas aplicações vão crescendo, tornando-se robustas e cada vez com maior valor para as empresas.

Para que estas aplicações continuem crescendo e agregando sempre valor para a empresa, é imprescindível que ela adote princípios da engenharia de software, para que consiga alcançar os seus objetivos. E neste contexto o princípio da separação de interesses tem o objetivo de decompor a aplicação de software em partes que se sobrepõem em termos de funcionalidade o mínimo possível. A preocupação em resolver um determinado problema é considerada como sinônimo de resolver uma característica ou comportamento da aplicação (PRESSMAN, 2011).

Estas aplicações são integradas com outras aplicações da empresa, de terceiros ou até realizado a sua manutenção corretiva, para que se consiga realizar estas ações, sem haver qualquer problema, é necessário manter a aplicabilidade da separação de interesses, pois ela nos traz os seguintes benefícios: manutenibilidade, flexibilidade, reutilização e confiabilidade. Além destes benefícios, é possível seguir as métricas de baixo acoplamento e alta coesão no ambiente de desenvolvimento de software (SOMMERVILLE, 2007).

Neste trabalho apresenta-se a aplicação do princípio de separação de interesses em procedimentos armazenados de banco de dados Oracle em códigos PL/SQL, objetivando explicar as melhores práticas de desenvolvimento destes códigos, evidenciando os problemas enfrentados quando não aplicada à separação de interesses no ambiente de desenvolvimento.

1.1 OBJETIVO GERAL

Identificar a eficiência no uso da separação de interesses em procedimentos armazenados em banco de dados Oracle através da aplicação de métricas de código fonte.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos são:

- Desenvolver um referencial teórico sobre o tema em estudo;
- Aplicar métricas de código-fonte para identificar as melhorias e a qualidade no desenvolvimento de procedimento armazenados utilizando a separação de interesses, sendo estas:
    - Total Number of Modules or Classes – Número total de módulos ou classes;
    - NOA (Number of Attributes – Numero de atributos);
    - ANPM (Average Number of Parameters per Method – Média do número de parâmetros por método);
    - ACC (Afferent Connections per Class – Conexões aferentes de uma classe);
    - COF (Coupling Factor – Fator de acoplamento);
    - SC (Structural Complexity – Complexidade estrutural);
- As métricas são objetivas, todas serão calculadas a partir de uma análise estática do código-fonte. Seus resultados serão mapeados em intervalos para serem interpretados quando analisados;

1.3 JUSTIFICATIVA

A separação de interesses está no centro da engenharia de software, como um princípio fundamental do projeto e da implementação de software, que tenta sanar um problema complexo por meio da subdivisão do mesmo em trechos a serem resolvidos e/ou otimizados independentemente, onde cada fragmento deve tratar de um interesse único facilitando a sua manutenção e utilização. Esta definição é tratada por Dijkstra (1976) como “Separação de Concerns”, por Pressman (2011) como “Separação de Interesses” e por Sommerville (2007) como “Separação de Assuntos”. Utiliza-se nessa pesquisa a terminologia adotada por Pressman (2011).

Um procedimento armazenado é derivado a partir de um bloco com estrutura em PL/SQL pré-compilado que fica armazenado como um objeto do esquema no banco de dados. Segundo GUPTA (2012) os procedimentos implementam as lógicas de negócio dos aplicativos. Por esta razão, os procedimentos são muitas vezes referidos como Gerentes de Negócios em PL/SQL, que não só mantem o repositório da lógica de negócios, mas também demonstram a escalabilidade da solução em uma forma modular de programação. Para DATE (1985) a perda de desempenho associada ao processamento no nível de registro em um sistema cliente/servidor pode ser em parte compensada pela criação de um procedimento armazenado conveniente para realizar esse processamento diretamente no banco de dados.

Segundo ANTONIO (2014) ao utilizar a separação de interesses no desenvolvimento de projetos orientado a objeto, tornam os mesmos mais flexíveis possibilitando o reaproveitamento e reutilização de códigos em novas gerações ou versões do sistema. Porém no desenvolvimento de procedimentos armazenados em banco de dados a capacidade dos sistemas se tornarem mais flexíveis está diretamente ligada ao banco de dados, afirma FOWLER (2003). Torna-se imprescindível o estudo da aplicação da separação de interesses, visando melhorar a capacidade de reutilização e manutenção de procedimentos armazenados em banco de dados Oracle.

1.4 METODOLOGIA

Para realização do presente trabalho, primeiramente será realizada uma pesquisa bibliográfica, a qual “abrange toda a bibliografia já tomada pública em relação ao tema em estudo” (MARCONI & LAKATOS, 2007, p71).

Em um segundo momento será utilizada uma observação sistemática, quanto a aplicação da separação de interesses em procedimentos armazenados em banco de dados Oracle. Para realizar uma observação sistemática, as condições devem ser controladas, para responder a propósitos preestabelecidos. Todavia, as normas não devem ser padronizadas nem rígidas demais, deve ser planejada com cuidado e sistematizada. Na observação sistemática, o observador sabe o que procura e o que precisa de importância em determinada situação; deve ser objetivo, reconhecer possíveis erros e eliminar sua influência sobre o que vê ou recolhe (MARCONI & LAKATOS, 2007).

2 PRINCÍPIO DA SEPARAÇÃO DE INTERESSES

O princípio da separação de interesses surgiu por meio do desafio de estabelecer limites nas aplicações, sendo que estas delimitações, físicas ou lógicas, determinaram a motivação deste princípio. Estes limites incluem as definições do comportamento da aplicação (BACCARO, 2014).

Dijkstra (1974) menciona que, "a separação de interesses, mesmo que não seja perfeitamente possível, é ainda a única técnica disponível para ordenação eficaz dos pensamentos".

O princípio da separação de interesses defende que, para resolver a complexidade, deve-se solucionar um interesse (ou *concern*) importante por vez (DIJKSTRA, 1976). Na engenharia de software, esse princípio está relacionado à modularização e à decomposição de sistemas (PARNAS, 1972). Os sistemas de software complexos devem ser decompostos em unidades modulares menores e claramente separadas, cada uma lidando com um único interesse (PARNAS, 1972). Se bem realizada, a separação de interesses pode proporcionar diversos benefícios cruciais (DIJKSTRA, 1976).

Os interesses podem ser universais, dependentes do domínio e dependentes da aplicação. Os interesses universais estão inseridos em boa parte dos sistemas de software, independente do domínio e da funcionalidade da aplicação. Alguns aspectos típicos onde utiliza-se os interesses universais são em tratamento de erro, distribuição, persistência, auditoria etc. Os interesses dependentes do domínio estão conexos a um domínio de aplicação explicito (CZARNECKI & EISENECKER, 2000).

> "A Separação por interesses é um conceito de projeto que sugere que qualquer problema complexo pode ser retratado mais facilmente se for subdividido em trechos a serem resolvidos e ou otimizados independentemente. Interesse se manifesta como uma característica ou comportamento especificados como parte do modelo de requisitos do software. Por meio da separação por interesses em blocos menores e, portanto, mais administráveis, um problema toma menos tempo para ser resolvido. " (Pressman, 2007).

Desde os primeiros sistemas de software que foram desenvolvidos, foi percebido que era importante para eles serem modulares. Sendo indispensável seguir uma metodologia, quanto a decomposição de um sistema em módulos e isso comumente é feito por enfocando as métricas de qualidade de software de acoplamento e coesão (MAKABEE, 2012).

Ao aplicar a separação de interesses pode obter como resultado alguns benefícios como:

- A carência de duplicação e singularidade do efeito dos artefatos individuais tornam o sistema global mais fácil de manter;
- O sistema como um todo, torna-se mais imutável, como um subproduto do aumento da capacidade de manutenção;
- As táticas necessárias para garantir que cada componente só se preocupe com um único conjunto de responsabilidades coesas muitas vezes derivam em pontos de extensibilidade naturais;
- A dissociação que deriva da necessidade de componentes para se concentrar em um único proposito leva a componentes que são mais prontamente reutilizados em outros sistemas ou contextos diferentes dentro do próprio sistema;
- O aumento da manutenção e extensão pode ter um amplo impacto sobre a taxa do sistema de comercialização e aprovação; e
- O princípio da separação de interesses também é benéfico quando aplicada dentro das empresas, garantido que os grupos e sub organizações são cominados a um conjunto único de responsabilidades de coesão. Isso promove os objetivos gerais de negócios, minimizando coordenação entre as equipes e maximizando o potencial de cada equipe ao se concentrar em sua responsabilidade coletiva, núcleo de competência e aperfeiçoando a resolução de problemas em amplos sistemas. Quando as responsabilidades ficam devidamente delineadas, a identificação do problema se torna mais simples e a resolução se desanda mais rápida (BACCARO, 2013).

Para dois problemas, p^1 e p^2, se a complexidade de p^1 for superior que a entendida para p^2, segue que o empenho necessário para resolver p^1 é maior do que o esforço necessário para solucionar p^2. Como caso comum, esse resultado é intuitivamente óbvio. Leva mais tempo para resolver um enigma complexo (PRESSMAN, 2011).

Segue também que a complexidade compreendida de dois problemas, quando estes são compatíveis, normalmente é maior do que soma da complexidade atingida quando cada um deles é adotado individualmente. Isso leva a uma tática dividir-para-conquistar – é mais simples resolver um problema complexo quando ele é subdividido em partes gerenciais. Isso tem implicações importantes em relação à modularidade do software (PRESSMAN, 2011).

Pressman (2011) diz que “a separação por interesses se manifesta em outros conceitos relacionados ao projeto: modularidade, aspectos, independência funcional e refinamento”.

Um sistema flexível faz com que seja simples realizar algumas alterações. Pois é possível acomodar às mudanças na solução de forma que não exista receios por parte da empresa ou das pessoas (BACCARO, 2014).

O processo de realização da separação de interesses envolve uma divisão do conjunto de funções, cujo objetivo não é reduzir um sistema em partes indivisíveis, mas sim a organização do sistema em partes desacopladas e de responsabilidades coesas. Este princípio tem sua essência na ordem, cujo objetivo geral é estabelecer que um sistema fosse bem organizado, onde cada parte cumpre o papel designado a intuitivo, maximizando sua capacidade de adaptarem-se as mudanças (BACCARO, 2014; SOMMERVILLE, 2007).

A separação de interesses pode ser aplicada em vários segmentos na construção de uma solução tecnológica, desde a camada arquitetural à camada de apresentação, separando em módulos com suas respectivas responsabilidades pode se conseguir diversos benefícios atrelados a ela, facilitando os objetivos gerais de negócio da solução. Quando as responsabilidades são devidamente delineadas, a identificação do problema ou a sua resolução acabam se tornando mais fáceis e eficientes (BACCARO, 2014).

2.1 BAIXO ACOPLAMENTO

O acoplamento é um conceito bem amplo e abrange todos os aspectos do desenvolvimento de software, na análise até a programação. As conexões físicas entre elementos do projeto orientado a objeto representam o acoplamento em um sistema (PRESSMAN, 2011; TROSIN, 2014).

Deve-se buscar reduzir o acoplamento entre os artefatos do sistema, pelo motivo bem conhecido, de facilitar a evolução independente de cada um destes artefatos (TROSIN, 2014). É imprescindível que os artefatos do projeto contribuam uns com os outros. No entanto, a cooperação deverá ser mantida em um nível mínimo aceitável. Se um modelo de projeto é altamente acoplado, o sistema é complicado de implementar, testar e manter com o transcorrer do tempo (PRESSMAN, 2011).

2.2 ALTA COESÃO

Um componente deverá ser projetado de modo que tenha todas as operações funcionando em conjunto para alcançar um objetivo único e bem definido (PRESSMAN, 2011). Quando um componente abrange muitas responsabilidades, e muitos motivos para mudar, então ele não é muito coeso. Baixa coesão é percebível quando as responsabilidades comuns estão espalhadas por todos os componentes (TROSIN, 2014). A coerência de um componente é apurada examinando-se o grau segundo o qual "o conjunto de propriedades que ele possui faz parte do domínio do problema ou projeto" (PRESSMAN, 2011).

Uma classe de projeto coesa é limitada. Ela tem um conjugado de responsabilidades pequeno e concentrado, e aplica de maneira simples atributos e métodos para implementar aquelas responsabilidades (PRESSMAN, 2011). É fácil de ser realizada a manutenção de componentes altamente coesivos. Se toda a lógica de lidar com o componente está definida de uma forma que exista apenas uma responsabilidade para seu funcionamento, não se faz necessário procurar por todo o sistema onde estão suas chamadas (TROSIN, 2014).

2.3 MANUTENIBILIDADE

Manutenção de software é definida como um processo de modificação de um determinado produto ou componente, de forma a corrigi-lo, melhorá-lo ou adaptá-lo para uma mudança de ambiente operacional. A manutenibilidade é o atributo que caracteriza a facilidade destes processos (MARTINS, 2014).

Manutenibilidade é uma medida de quão fácil é manter o sistema. Como consequência do baixo acoplamento, há uma probabilidade reduzida de que uma mudança em um módulo será propagada para outros módulos. Como consequência da alta coesão, há um aumento da probabilidade de que uma mudança nos requisitos do sistema afetará apenas um pequeno número de módulos (MAKABEE, 2012).

A manutenção de uma aplicação de software sempre foi e sempre será de maior importância para as empresas. Produzindo deste modo um produto que garanta seus serviços de uma melhor forma e atenda às necessidades do ambiente onde este está inserido (MARTINS, 2014).

2.4 FLEXIBILIDADE

A flexibilidade de um componente está relacionada há como ele é capaz de se adequar as mudanças solicitadas por um software. Mudanças acontecem a todo o momento durante o processo de desenvolvimento e após a sua entrega final. O produto deve estar preparado para isso. A ausência deste princípio gera, naturalmente, um receio por mudanças, é devido à ausência deste princípio que as empresas enfrentam problemas com relação às mudanças (MARTINS, 2014).

A extensibilidade mede quão facilmente o sistema pode ser expandido com novas funcionalidades. Como consequência do baixo acoplamento, deve ser mais fácil introduzir novos módulos. Como consequência da alta coesão, deve ser mais fácil de implementar novos módulos sem se preocupar com os aspectos que não estão diretamente relacionados com a sua funcionalidade (MAKABEE, 2012).

2.5 REUSABILIDADE

O conceito de reusabilidade refere-se à facilidade ou a potencialidade ao fazer uso de componentes existentes a fim de reutilizar o que já é funcional ou o que já foi desenvolvido (PICOLO, 2014).

Este princípio está relacionado diretamente à alta coesão e baixo acoplamento, pois desacoplado os componentes, e definindo responsabilidades únicas para eles, pode-se reutiliza-los em outro componente sem ter a necessidade de mudanças no que já foi implementado (PICOLO, 2014).

Com a reusabilidade pode-se reduzir custos, tempo de desenvolvimento e de manutenções para o produto. É uma forma de melhorar a qualidade geral do processo de criação e manutenção de software. Garantindo a confiança de componentes e a organização dos mesmos (PICOLO, 2014; SOMMERVILLE, 2007).

Reutilização é uma medida de como é fácil reutilizar um módulo em um sistema diferente. Como consequência do baixo acoplamento, deve ser mais simples de reutilizar um módulo que foi desenvolvido no passado para um sistema anterior, porque esse módulo deve ser menos dependente do restante do sistema. Como decorrência da alta coesão, a

funcionalidade fornecida por um módulo deve ser bem definido e completo, tornando-o mais útil como um componente reutilizável (MAKABEE, 2012).

2.6 CONFIABILIDADE

Confiabilidade diz respeito à capacidade de um componente de software funcionar como o esperado, mantendo seu funcionamento de forma rotineira sem ocorrência de falhas. Este conceito é um dos principais atributos da qualidade de software, pois é necessário que este software seja testado para que possa garantir o funcionamento esperado quando estiver operando (MARTINS, 2014).

3 PROCEDIMENTOS ARMAZENADOS DE BANCO DE DADOS ORACLE

Quando usado procedimentos armazenados ou módulos armazenados persistentes, como é o termo conhecido pelo padrão SQL, na construção de software, faz-se utilização de uma linguagem de programação de terceira geração para isto (ELMASRI, 2011).

Com a linguagem PL/SQL pode-se construir procedimentos armazenados e integrar com o SQL, tornando possível a construção de aplicações complexas e robustas. O PL/SQL é executado no banco de dados, podendo incluir instruções SQL em seu código sem ter que estabelecer uma conexão separa para isto (ELMASRI, 2011).

Segundo FOWLER (2003), existe uma dependência direta entre um sistema de banco de dados e a capacidade do sistema se tornar mais flexível. O código totalmente implementado no banco de dados, caracteriza por outro lado um alto desempenho se comparado com as chamadas, preparadas em Java.

Os principais tipos de módulos armazenados persistentes que podemos criar no banco de dados Oracle são: procedure, função, pacote e trigger. Uma vez que armazenados no banco de dados, esses módulos escritos em linguagem PL/SQL podem ser usados como blocos de construção para diversas aplicações diferentes (WATSON, 2010).

Um procedimento armazenado é proveniente a partir de um bloco com estrutura em PL/SQL pré-compilado que permanece armazenado como um objeto do esquema no banco de dados. Segundo GUPTA (2012) os procedimentos implementam as lógicas de negócio dos aplicativos. Por esta razão, os procedimentos são muitas vezes referidos como Gerentes de Negócios em PL/SQL, que não só mantem o repositório da lógica de negócios, mas também evidenciam a escalabilidade da solução em uma forma modular de programação. A perda de desempenho associada ao processamento no nível de registro em um sistema cliente/servidor pode ser em parte compensada pela criação de um procedimento armazenado conveniente para realizar esse processamento inteiramente no banco de dados (DATE, 1985).

Os procedimentos armazenados podem ser compilados e executados com diferentes parâmetros e resultados, e eles podem ter qualquer combinação de parâmetros de entrada, saída e de entrada/saída. A maioria dos sistemas de gerenciamento de banco de dados suportam os procedimentos armazenados, mas existe uma boa quantidade de variações na sua sintaxe e capacidade para desempenhar determinadas operações (ORACLE, 2014).

Os procedimentos armazenados são uteis nas seguintes circunstâncias:

- Se um programa de banco de dados é necessário por várias aplicações, ele pode ser armazenado no servidor e invocado por qualquer um dos programas de aplicações. Isso reduz a duplicidade de esforço e melhora a modularidade do software (WATSON, 2010).
- A execução de um programa no servidor pode reduzir a transferência de dados e o custo de comunicação entre o cliente e o servidor em certas situações (WATSON, 2010).
- Esses procedimentos podem melhorar o poder de modelagem fornecido pelas visões ao permitir que tipos mais complexos de dados derivados estejam disponíveis aos usuários de banco de dados. Além disso, eles podem ser usados para verificar restrições complexas que estão além do poder de especificação de asserções e triggers (WATSON, 2010).
- Tais procedimentos podem ser usados para ocultar do usuário uma variedade de detalhes específicos do SGDB e/ou do banco de dados, fornecendo assim um grau maior de independência de dados do que se teria sem eles (DATE, 1985).
- Um procedimento armazenado pode ser compartilhado por vários clientes (DATE, 1985).
- A otimização pode ser feita no momento em que o procedimento armazenado é compilado e não em tempo de execução. (Naturalmente, essa vantagem se aplica apenas a sistemas que normalmente efetuam a otimização em tempo de execução.) (DATE, 1985).
- Procedimentos armazenados podem oferecer mais segurança. Por exemplo, um usuário qualquer poderia ser autorizado a chamar determinado procedimento, mas não a operar diretamente sobre os dados acessados por esse procedimento (DATE, 1985).

Uma desvantagem é que fornecedores diferentes oferecem recursos muito diferentes nessa área (DATE, 1985).

O formato de declarações da criação de uma procedure no Oracle é o seguinte (Quadro 1):

```
CREATE OR REPLACE PROCEDURE <nome da procedure>(<parâmetros>) AS
   <declarações de variáveis locais>
BEGIN
   <corpo da procedure>;
```

```
END <nome da procedure>;
```

Quadro 1 – Estrutura e declarações para criação de um procedimento

Fonte: ORACLE, 2014.

Os parâmetros e declarações locais são opcionais e especificados apenas se necessário. Cada procedimento armazenado, independente do seu tipo, deve possuir um nome, ou seja, uma identificação no esquema de objetos de banco de dados (ORACLE, 2014).

Para declarar uma função, um tipo de retorno é necessário, segue o formato de declaração de uma função (Quadro 2):

```
CREATE OR REPLACE FUNCTION <nome da função>(<parâmetros>)
RETURN <tipo de retorno> AS
   <declaração de variáveis locais>
BEGIN
   <corpo da função>;
   RETURN <variável ou valor de retorno>
END <nome da função>;
```

Quadro 2 – Estrutura e declaração para criação de uma função

Fonte: ORACLE, 2014.

Em geral, cada parâmetro deve ter um tipo, o qual é um dos tipos de dados da SQL. Cada parâmetro também deve ter um modo, que é um dentre IN, OUT, ou INOUT. Esses correspondem a parâmetros cujos valores são apenas de entrada, apenas de saída (retornados) ou de entrada e saída, respectivamente (ORACLE, 2014).

Aplicações de nível empresarial tem uma complexidade muito maior, algumas das interfaces e tipos estão diretamente disponíveis para o usuário, enquanto outros são usados somente por funções e procedimentos e nunca são chamados pelo usuário. A linguagem PL/SQL no Oracle permite declarar formalmente a relação entre esses subprogramas, colocando-os em um pacote, que é um objeto do esquema do banco de dados que agrupa vários elementos, tais como: variáveis e procedimentos armazenados (ELMASRI, 2011).

Outra razão para que os procedimentos armazenados sejam definidos em uma especificação de pacote, é pela limitação do desenvolvimento em grande escala, os procedimentos armazenados suportam apenas parâmetros escalares (NUMBER, VARCHAR2, e DATE). Não pode ser utilizado uma estrutura composta (RECORD) quando os procedimentos armazenados são criados de forma autônomas, ou seja, não são definidos em uma especificação de pacote (ORACLE, 2014).

Os pacotes possuem duas partes de definição, uma delas é a especificação, e a outra é o corpo. O pacote é definido pela especificação, que declara os tipos, variáveis, constantes, exceções, cursores e procedimentos armazenados que podem ser referenciados de fora do pacote. A especificação é a interface para o pacote, é nela que se coloca quais procedimentos armazenados poderão ser referenciados. Uma aplicação que necessita dos procedimentos armazenados dentro do pacote, só precisa saber os nomes e os parâmetros da especificação do pacote (ORACLE, 2014). A declaração da especificação segue o seguinte padrão (Quadro 3):

```
CREATE OR REPLACE PACKAGE <nome do pacote> AS
   <tipo de definição para estruturas compostas>;
   <constantes>
   <exceções>
   <variáveis globais>
   PROCEDURE <nome da procedure>(<parâmetros>);
   FUNCTION <nome da função>(<parâmetros>) RETURN <tipo de
retorno>;
END <nome do pacote>;
```

Quadro 3 – Estrutura e declaração da especificação de criação de um pacote

Fonte: ORACLE, 2014.

O corpo do pacote contém o código que implementa os procedimentos armazenados. Os procedimentos armazenados que não precisam ser vistos de fora do pacote, não deverão ser assinados na especificação do pacote, estes poderão ser chamados apenas de dentro do corpo do pacote. Pode-se alterar a implementação dos procedimentos armazenados sem ter que invalidar chamadas destes que estão sendo referenciados nas aplicações de fora do pacote (ORACLE, 2014). O corpo do pacote segue a seguinte estrutura de declarações (Quadro 4):

```
CREATE OR REPLACE PACKAGE BODY <nome do pacote> AS
   PROCEDURE <nome da procedure>(<parâmetros>) AS
      ...
   BEGIN
      ...
   END <nome da procedure>;
   FUNCTION <nome da função>(<parâmetros>) RETURN <tipo do
retorno> AS
      ...
   BEGIN
      ...
      RETURN <variável ou valor de retorno>
   END <nome da função>;
END <nome do pacote>;
```

Quadro 4 – Estrutura e declaração de criação do corpo de um pacote

Fonte: ORACLE, 2014.

Deve-se projetar e definir a especificação do pacote antes de escrever a implementação no corpo do pacote. Na especificação, inclui-se apenas as partes que devem ser publicamente visíveis para a chamada de fora do pacote, e esconde-se as declarações privadas dentro do corpo do pacote (ORACLE, 2014). A forma de chamar um procedimento armazenado de dentro de um pacote é pelo seguinte código (Quadro 5):

```
<nome do pacote>.<nome do procedimento>(<parâmetros>);
```

Quadro 5 – Estrutura do código de chamada de um procedimento de dentro de um pacote

Fonte: ORACLE, 2014.

4 MÉTRICAS DE CÓDIGO-FONTE

Especificamente no desenvolvimento de software, qualidade é determinada como um conjunto de características atendidas, em um determinado nível, para que o produto de software contemple as necessidades explicitas e implícitas de seus usuários (ROCHA, 2001). O padrão Internacional Software Engineering – Product quality, ISO/IEC 9126 (ISO/IEC25010:2011, 2011), prevê um conjunto de parâmetros para a avaliação da qualidade de software, com o objetivo de diminuir a subjetividade do conceito de qualidade.

Na qualidade de software os predicados usabilidade e portabilidade podem ser considerados inteiramente não-funcionais, ou seja, não estão relacionados com as tarefas concretizadas pelo código em si. Já os atributos funcionalidade, confiança, eficiência e manutenibilidade, ficam parcialmente ou totalmente ligados com as características funcionais, em que, na maioria das vezes, os requisitos e os testes do software são baseados. Sendo assim, para requisitos funcionais, há métricas de software objetivas, que em certo grau, podem refletir a qualidade e, portanto, são empregadas também na avaliação da qualidade de métodos (BANSIYA e DAVIS, 2002) e de modelos (ZUSE, 1990) de software.

Uma maneira prática de se observar as características de um código-fonte é analisando os valores de suas métricas. Entretanto, ainda não está difundida uma análise sistemática da qualidade do código-fonte produzido.

Uma métrica, segundo definição da ISO/IEC25010:2011 (2011), “é a composição de procedimentos para a definição de escalas e métodos para medidas”. Então, métricas são utilizadas para estimar um cronograma e custos de desenvolvimento do software e medir a produtividade e qualidade do produto.

São consideradas boas métricas aquelas que promovem a medição dos parâmetros de qualidade definidos para um determinado software (MILLS, 1988). Na tentativa de medir tais parâmetros, há o risco de se perder na enorme quantidade de métricas propostas pela literatura (ZUSE, 1990). O tratamento de um amplo volume de dados pode ser humanamente impraticável. Sendo assim, é preciso definir e justificar os critérios adotados na escolha das métricas (MEIRELLES, 2013). É desejável que as métricas possuam as seguintes características:

- O que a métrica se propõe a medir deve ser claro;
- A métrica deve ser formalmente definida, seu valor deve estar atrelado ao objeto medido, independente de quem/qual o obtenha;

- Deve ser possível obter seu valor rapidamente e o baixo custo;
- A métrica deve medir efetivamente o proposto por ela;
- Pequenas mudanças no software, por exemplo, não podem causar grande mudanças no valor obtido;
- Deve haver formas de mapeamento das métricas para o entendimento do comportamento das entidades analisadas através da manipulação dos números obtidos;
- O resultado da métrica deve ser independente da linguagem de programação utilizada.

As métricas de software podem ser classificadas de diversas formas, sendo quanto ao âmbito da sua aplicação, quanto ao critério utilizado na sua definição e quanto ao método de obtenção da medida. A maneira mais ampla de classificá-las é quanto ao objeto da métrica.

As métricas podem ser divididas em primitivas e compostas quando classificadas pelo método de obtenção. As métricas primitivas são aquelas que podem ser diretamente observadas em uma única métrica. As métricas compostas são as combinações de duas ou mais métricas.

Outra maneira de classificar as métricas é quanto aos critérios utilizados para determina-las. Assim, as métricas são diferenciadas em métricas objetivas e métricas subjetivas. As métricas são obtidas através de regras bem definidas, sendo a melhor forma de possibilitar comparações posteriores consistentes. As subjetivas dependem do sujeito que está realizando a medida e, portanto, dificultam as comparações e a reprodutibilidade das medidas. Para possibilitar uma avaliação de software de forma sistemática, eficiente, eficaz e de baixo custo, os valores obtidos através das métricas deveriam ser sempre os mesmos, independentemente do momento, condições ou indivíduo que realiza as medidas (MEIRELLES, 2013).

Após a classificação as métricas devem ser associadas a uma escala de medição que proporcione significado ao valor obtido no seu cálculo. Elas precisam ser coletadas em um modelo de dados específico que pode envolver cálculos ou análises estatística. Para isso, é importante considerar o tipo de informação obtida. Assim, quatro tipos de dados de medidas foram reconhecidos por estatísticos para as métricas de software:

- Nominal: na interpretação dos valores de cada atributo, a ordem não possui significado.

- Ordinal: os valores possuem ordem, mas a distância entre eles não possui significado. Por exemplo, nível de experiência dos programadores.
- Intervalo: a ordem dos resultados é importante, assim como o tamanho dos intervalos que separam os pontos, mas as proporções não são necessariamente válidas. Por exemplo, um programa com "complexidade" de valor 6 é mais complexo em 4 unidades do que um programa com a complexidade de valor 2, mas isso não é muito significativo para dizer que o primeiro programa é 3 vezes mais complexo do que o segundo.
- Racional: semelhante à medida por intervalo, mas preservando as proporções.

As métricas classificadas como objetivas tratam de características do código-fonte.

4.1.1 MÉTRICA DE TAMANHO

Total Number of Modules or Classes – Número total de módulos ou classes é um indicador de tamanho, que é menos influenciado por linguagens de programação, nível dos desenvolvedores e estilos de codificação. Assim, essa métrica pode ser usada para comparar projetos escritos em diferentes linguagens de programação (TEMPERO, 2008).

4.1.2 MÉTRICAS ESTRUTURAIS

Ao considerar características como esforço de manutenibilidade, flexibilidade, compreensão e qualidade do código-fonte, em geral, deve-se levar em consideração não só as métricas de tamanho acima, descritas, mas também indicadores estruturais.

NOA (*Number of Attributes* – Número de atributos) calcula o número de atributos de uma classe. Seu valor mínimo é zero e não existe um limite máximo para o seu resultado. Uma classe com muitos atributos pode indicar que ela tem muitas responsabilidades e apresentar pouca coesão, i.e., deve estar tratando de vários assuntos diferentes.

ANPM (*Average Number of Parameters per Method* – Média do Número de Parâmetros por Método) calcula a média de parâmetros dos métodos da classe. Seu valor mínimo é zero e não existe um limite máximo para o seu resultado, mas um número alto de

parâmetros pode indicar que um método pode ter mais de uma responsabilidade, ou seja, mais de uma função (BANSIYA e DAVI, 1997).

4.1.3 MÉTRICAS DE ACOPLAMENTO

Acoplamento é uma medida de como uma classe está ligada a outras classes no software. Altos valores de acoplamento indicam uma maior dificuldade para alterar uma classe do sistema, pois uma mudança em uma classe pode ter um impacto em todas as outras classes que estão acopladas a ela. Em outras palavras, se o acoplamento é alto, o software tende a ser menos flexível, mais difícil de se adaptar e modificar e mais difícil de entender.

ACC (Afferent Connections per Class – Conexões aferentes de uma classe) mede a conectividade de uma classe. Se uma classe Cc acessa um método ou atributo da classe Cs, entende-se que Cc é cliente da classe fornecedora Cs e denota-se por Cc => Cs. Considera-se a seguinte função:

$$cliente(C_i, C_j) = \begin{cases} 1 & \text{sse } C_i \Rightarrow C_j \wedge C_i \neq C_j \\ 0 & \text{caso contrário} \end{cases}$$

Figura 1 – Função do ACC
Fonte: MEIRELLES, 2013.

Então:

$$ACC(C) = \sum_{i=1}^{n} cliente(C_i, C)$$

Figura 2 – Fórmula do ACC
Fonte: MEIRELLES, 2013.

Onde n é o número total de classes do sistema. Se o valor dessa métrica for grande, uma mudança na classe tem potencialmente mais efeitos colaterais, tornando mais difícil a manutenção. Os intervalos sugeridos são: até 2 (bom); entre 2 e 20 (regular); de 20 em diante (ruim).

COF (Coupling Factor – Fator de acoplamento) indica o quão interconectado é o software. Seu valor é dado por:

$$COF = \frac{\sum_{i=1}^{n} ACC(C_i)}{n^2 - n}$$

Figura 3 – Fórmula do COF
Fonte: MEIRELLES, 2013.

O numerador é o total de ligações entre as classes e o denominador é o total possível de ligações. Portanto, COF é a razão de ligações. Um software fortemente conectado apresenta maior COF, indicando um baixo grau de independência entre os módulos, alta complexidade e difíceis entendimento e manutenção. Os intervalos sugeridos são: até 0,02 (bom); entre 0,02 e 0,14 (regular); de 0,14 em diante (ruim).

4.1.4 MÉTRICA DE COESÃO

Coesão mede a diversidade de “assuntos” que uma classe implementa. Altos valores de coesão indicam se o “foco” de uma classe está em um único aspecto do sistema. Enquanto uma baixa coesão indica que a classe trata de diferentes aspectos. Assim, uma classe deve ser coesa.

SC (*Structural Complexity* – Complexidade Estrutural) quanto mais complexo for um software, mais difícil será alterá-lo e evoluí-lo. Acoplamento e coesão foram descritos e discutidos em outros trabalhos como indicadores essenciais de complexidade estrutural (Darcy et al., 2005). É aconselhável manter um baixo acoplamento e uma alta coesão das classes (Richter, 1999).

5 DESENVOLVIMENTO DOS PROCEDIMENTOS E APLICAÇÃO DAS MÉTRICAS

A solução idealizada refere-se a manter diferentes esquemas de banco de dados através de um esquema padrão que deverá ser mantido, atualizado e revisado. Para compor essa solução faz-se necessários diferentes procedimentos armazenados que serão representados por suas chamadas.

A figura 4 apresenta a modelagem relacional do esquema padrão que é tomado como referência pelos demais esquemas, as tabelas PRODUTO, SUBSTANCIA, PRODUTO_SUBSTANCIAS, CONCENTRACAO, APRESENTACAO, FABRICANTE, PRODUTO_CODIGO_DE_BARRAS e CODIGO_DE_BARRAS são as tabelas que devem ser revisadas e ter seus dados atualizados nos demais esquemas, as tabelas terminadas em "_JOURNAL" contém informações referentes aos dados que devem ser atualizados nos outros esquemas, e suas informações são alimentadas através de triggers (gatilhos).

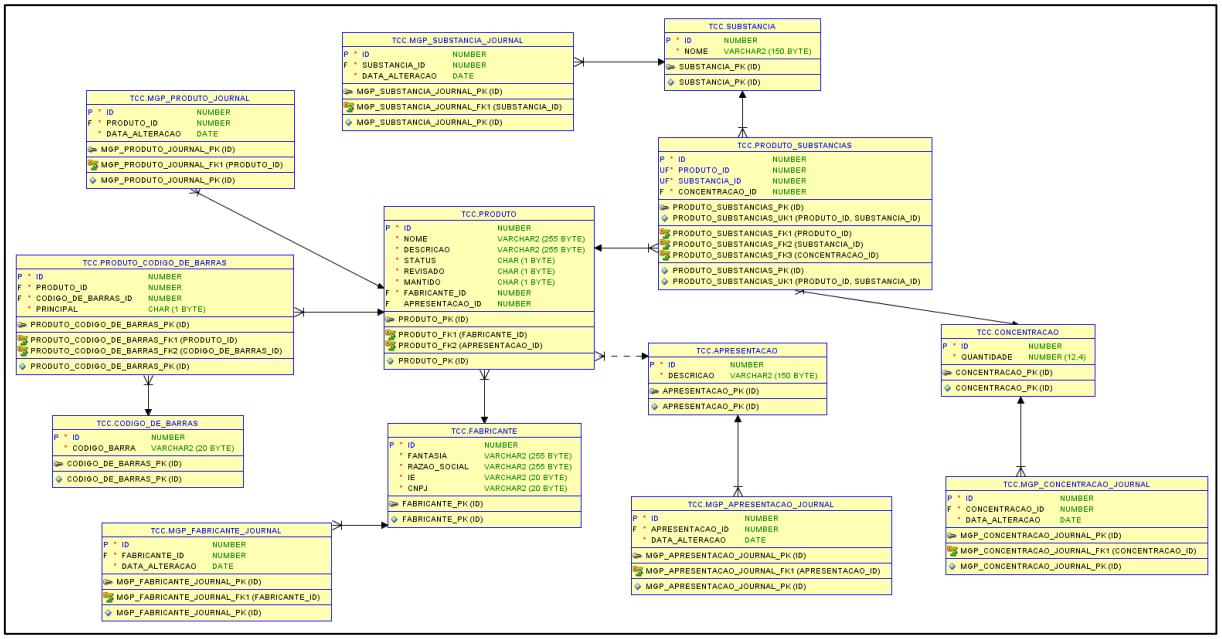

Figura 4 – Modelagem relacional esquema padrão

Fonte: Autoria própria.

A figura 5 apresenta a modelagem relacional dos esquemas mantidos, as tabelas PRODUTO, SUBSTANCIA, PRODUTO_SUBSTANCIAS, CONCENTRACAO, APRESENTACAO, FABRICANTE, PRODUTO_CODIGO_DE_BARRAS e CODIGO_DE_BARRAS são as tabelas que terão os dados atualizados a tabela

MGP_PRODUTO_JOURNAL contém informações referentes aos produtos novos cadastrados nos esquemas mantidos e é alimentada através de uma trigger.

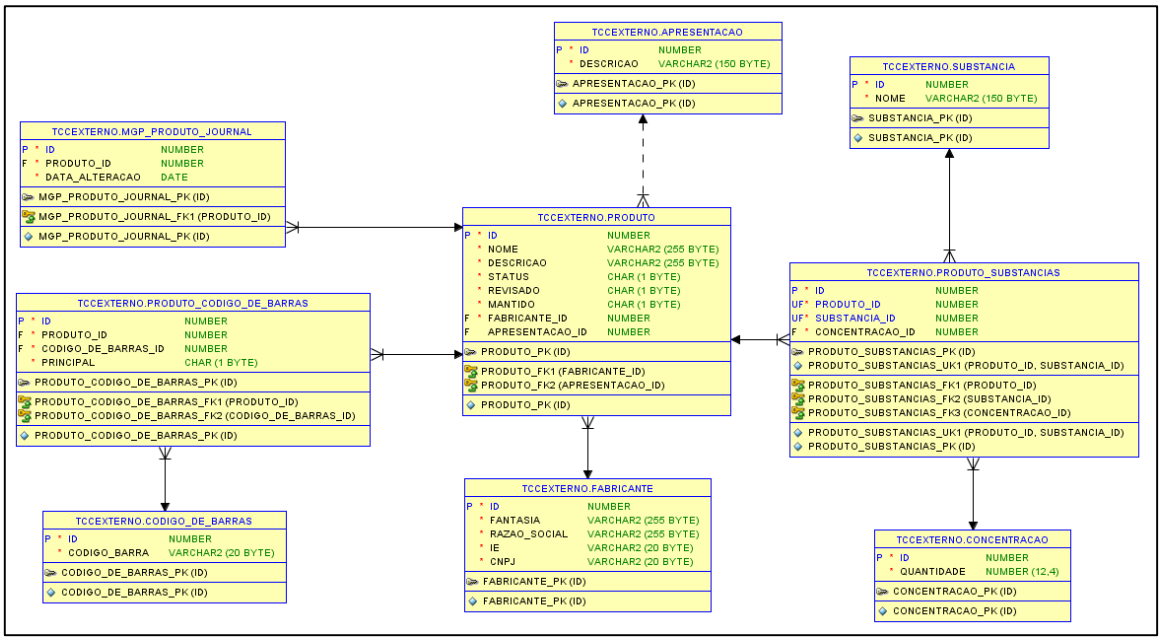

Figura 5 – Modelagem relacional esquemas mantidos

Fonte: Autoria própria.

Os dados existentes nos esquemas mantidos são valores gerados anteriormente ou até mesmo importados de outros bancos de dados e suas chaves devem ser preservadas. Portando para manter os dados atualizados com base no esquema padrão é necessário determinar alguns campos de referência para realizar o processo de manutenção dos esquemas mantidos.

O principal interesse da solução é manter diversos esquemas atualizados conforme esquema padrão. Esse interesse será divido em diversos interesses para reduzir as responsabilidades e melhor entendimento dos procedimentos que serão criados para solução.

Para dar início a solução é precisa criar triggers no esquema padrão, elas serão para inserções e alterações nas tabelas que serão mantidas e também de deleção nas tabelas de junção de produtos e códigos de barras e produtos e substancias. No quadro 6 é apresentada a criação desses gatilhos.

```
CREATE OR REPLACE TRIGGER TRG_INS_UPD_PRODUTO AFTER INSERT OR UPDATE ON
PRODUTO
FOR EACH ROW
      BEGIN
        IF INSERTING THEN
          INSERT INTO MGP_PRODUTO_JOURNAL(ID, PRODUTO_ID, DATA_ALTERACAO)
VALUES (MGP_PRODUTO_JOURNAL_SEQ.NEXTVAL, :NEW.ID, DATA_ALTERACAO());
        ELSIF UPDATING THEN
          INSERT INTO MGP_PRODUTO_JOURNAL(ID, PRODUTO_ID, DATA_ALTERACAO)
VALUES (MGP_PRODUTO_JOURNAL_SEQ.NEXTVAL, :OLD.ID, DATA_ALTERACAO());
```

```
        END IF;
      END TRG_INS_UPD_PRODUTO;

CREATE OR REPLACE TRIGGER TRG_INS_UPD_PRODUTO_CBARRAS AFTER INSERT OR
UPDATE ON PRODUTO_CODIGO_DE_BARRAS
FOR EACH ROW
      BEGIN
        IF INSERTING THEN
          INSERT INTO MGP_PRODUTO_JOURNAL(ID, PRODUTO_ID, DATA_ALTERACAO)
VALUES (MGP_PRODUTO_JOURNAL_SEQ.NEXTVAL, :NEW.PRODUTO_ID,
DATA_ALTERACAO());
        ELSIF UPDATING THEN
          INSERT INTO MGP_PRODUTO_JOURNAL(ID, PRODUTO_ID, DATA_ALTERACAO)
VALUES (MGP_PRODUTO_JOURNAL_SEQ.NEXTVAL, :NEW.PRODUTO_ID,
DATA_ALTERACAO());
          INSERT INTO MGP_PRODUTO_JOURNAL(ID, PRODUTO_ID, DATA_ALTERACAO)
VALUES (MGP_PRODUTO_JOURNAL_SEQ.NEXTVAL, :OLD.PRODUTO_ID,
DATA_ALTERACAO());
        END IF;
END TRG_INS_UPD_PRODUTO_CBARRAS;

CREATE OR REPLACE TRIGGER TRG_INS_UPD_PRODUTO_SUBS AFTER INSERT OR UPDATE
ON PRODUTO_SUBSTANCIAS
FOR EACH ROW
      BEGIN
        IF INSERTING THEN
          INSERT INTO MGP_PRODUTO_JOURNAL(ID, PRODUTO_ID, DATA_ALTERACAO)
VALUES (MGP_PRODUTO_JOURNAL_SEQ.NEXTVAL, :NEW.PRODUTO_ID,
DATA_ALTERACAO());
        ELSIF UPDATING THEN
          INSERT INTO MGP_PRODUTO_JOURNAL(ID, PRODUTO_ID, DATA_ALTERACAO)
VALUES (MGP_PRODUTO_JOURNAL_SEQ.NEXTVAL, :NEW.PRODUTO_ID,
DATA_ALTERACAO());
          INSERT INTO MGP_PRODUTO_JOURNAL(ID, PRODUTO_ID, DATA_ALTERACAO)
VALUES (MGP PRODUTO JOURNAL SEQ.NEXTVAL, :OLD.PRODUTO ID,
DATA_ALTERACAO());
        END IF;
END TRG_INS_UPD_PRODUTO_SUBS;

CREATE OR REPLACE TRIGGER TRG_DEL_PRODUTO_CODIGO_BARRAS BEFORE DELETE ON
PRODUTO_CODIGO_DE_BARRAS
FOR EACH ROW
      BEGIN
       INSERT INTO MGP_PRODUTO_JOURNAL(ID, PRODUTO_ID, DATA_ALTERACAO)
VALUES (MGP_PRODUTO_JOURNAL_SEQ.NEXTVAL, :OLD.ID, DATA_ALTERACAO());
END TRG_DEL_PRODUTO_CODIGO_BARRAS;

CREATE OR REPLACE TRIGGER TRG_DEL_PRODUTO_SUBSTANCIA BEFORE DELETE ON
PRODUTO_SUBSTANCIAS
FOR EACH ROW
      BEGIN
        INSERT INTO MGP_PRODUTO_JOURNAL(ID, PRODUTO_ID, DATA_ALTERACAO)
VALUES (MGP_PRODUTO_JOURNAL_SEQ.NEXTVAL, :OLD.ID, DATA_ALTERACAO());
END TRG_DEL_PRODUTO_SUBSTANCIA;

CREATE OR REPLACE TRIGGER TRG_INS_UPD_SUBSTANCIA AFTER INSERT OR UPDATE
ON SUBSTANCIA
FOR EACH ROW
      BEGIN
        IF INSERTING THEN
          INSERT INTO MGP_SUBSTANCIA_JOURNAL(ID, SUBSTANCIA_ID,
```

```
DATA_ALTERACAO) VALUES (MGP_SUBSTANCIA_JOURNAL_SEQ.NEXTVAL, :NEW.ID,
DATA_ALTERACAO());
        ELSIF UPDATING THEN
          INSERT INTO MGP_SUBSTANCIA_JOURNAL(ID, SUBSTANCIA_ID,
DATA_ALTERACAO) VALUES (MGP_SUBSTANCIA_JOURNAL_SEQ.NEXTVAL, :OLD.ID,
DATA_ALTERACAO());
        END IF;
END TRG_INS_UPD_SUBSTANCIA;

CREATE OR REPLACE TRIGGER TRG_INS_UPD_APRESENTACAO AFTER INSERT OR UPDATE
ON APRESENTACAO
FOR EACH ROW
      BEGIN
        IF INSERTING THEN
          INSERT INTO MGP_APRESENTACAO_JOURNAL(ID, APRESENTACAO_ID,
DATA_ALTERACAO) VALUES (MGP_APRESENTACAO_JOURNAL_SEQ.NEXTVAL, :NEW.ID,
DATA_ALTERACAO());
        ELSIF UPDATING THEN
          INSERT INTO MGP_APRESENTACAO_JOURNAL(ID, APRESENTACAO_ID,
DATA_ALTERACAO) VALUES (MGP_APRESENTACAO_JOURNAL_SEQ.NEXTVAL, :OLD.ID,
DATA_ALTERACAO());
        END IF;
END TRG_INS_UPD_APRESENTACAO;

CREATE OR REPLACE TRIGGER TRG_INS_UPD_CONCENTRACAO AFTER INSERT OR UPDATE
ON CONCENTRACAO
FOR EACH ROW
      BEGIN
        IF INSERTING THEN
          INSERT INTO MGP_CONCENTRACAO_JOURNAL(ID, CONCENTRACAO_ID,
DATA_ALTERACAO) VALUES (MGP_CONCENTRACAO_JOURNAL_SEQ.NEXTVAL, :NEW.ID,
DATA_ALTERACAO());
        ELSIF UPDATING THEN
          INSERT INTO MGP_CONCENTRACAO_JOURNAL(ID, CONCENTRACAO_ID,
DATA ALTERACAO) VALUES (MGP CONCENTRACAO JOURNAL SEQ.NEXTVAL, :OLD.ID,
DATA_ALTERACAO());
        END IF;
END TRG_INS_UPD_CONCENTRACAO;

CREATE OR REPLACE TRIGGER TRG_INS_UPD_FABRICANTE AFTER INSERT OR UPDATE
ON FABRICANTE
FOR EACH ROW
      BEGIN
        IF INSERTING THEN
          INSERT INTO MGP_FABRICANTE_JOURNAL(ID, FABRICANTE_ID,
DATA_ALTERACAO) VALUES (MGP_FABRICANTE_JOURNAL_SEQ.NEXTVAL, :NEW.ID,
DATA_ALTERACAO());
        ELSIF UPDATING THEN
          INSERT INTO MGP_FABRICANTE_JOURNAL(ID, FABRICANTE_ID,
DATA_ALTERACAO) VALUES (MGP_FABRICANTE_JOURNAL_SEQ.NEXTVAL, :OLD.ID,
DATA_ALTERACAO());
        END IF;
END TRG_INS_UPD_FABRICANTE;
```

Quadro 6 – Triggers do esquema padrão

Fonte: Autoria própria.

Nos esquemas mantidos é necessário criar uma trigger para inserção de novos produtos, para serem inseridos no esquema padrão e verificados para depois serem enviados para os demais esquemas. O quadro 7 apresenta a trigger de inserção de produtos em esquemas mantidos.

```
CREATE OR REPLACE TRIGGER MGP_PRODUTO_JOURNAL_TRG BEFORE INSERT ON
MGP_PRODUTO_JOURNAL
      FOR EACH ROW
      BEGIN
        IF INSERTING AND :NEW.ID IS NULL THEN
          SELECT MGP_PRODUTO_JOURNAL_SEQ.NEXTVAL INTO :NEW.ID FROM
SYS.DUAL;
        END IF;
END MGP_PRODUTO_JOURNAL_TRG;
```

Quadro 7 – Trigger de inserção de produtos nos esquemas mantidos.

Fonte: Autoria própria.

Os dados que devem ser mantidos atualizados nos esquemas, norteiam informações dos produtos, sendo assim, quando há alteração e/ou inserção de um produto no esquema padrão as tabelas ligadas a este produto devem ser verificadas também. Não somente no momento em que há alteração e/ou inserção nas tabelas especificas.

No quadro 8 é apresentada a estrutura da chamada do procedimento ATUALIZA_APRESENTACAO que é responsável por pegar a lista de apresentações que tenham que ser atualizadas com base nos ids mantidos na tabela MGP_APRESENTACAO_JOURNAL, para após identificadas as apresentações verificar no esquema mantido através da abreviação se é necessário inseri-las ou atualiza-las.

```
CREATE OR REPLACE PROCEDURE
ATUALIZA_APRESENTACAO(CURSOR_APRESENTACOES_JOURNAL IN SYS_REFCURSOR) IS
BEGIN
  ...
END ATUALIZA_APRESENTACAO;
```

Quadro 8 – Procedimento ATUALIZA_APRESENTACAO.

Fonte: Autoria própria.

No quadro 9 é apresentada a estrutura da chamada do procedimento ATUALIZA_CONCENTRACAO que é responsável por pegar a lista de concentrações que tenham que ser atualizadas com base nos ids mantidos na tabela MGP_CONCENTRACAO_JOURNAL, para após identificadas as concentrações verificar no esquema mantido através da quantidade se é necessário inseri-las ou atualiza-las.

```
CREATE OR REPLACE PROCEDURE
ATUALIZA_CONCENTRACAO(CURSOR_CONCENTRACOES_JOURNAL IN SYS_REFCURSOR) IS
```

```
BEGIN
  ...
END ATUALIZA_CONCENTRACAO;
```

Quadro 9 – Procedimento ATUALIZA_CONCENTRACAO.

Fonte: Autoria própria.

No quadro 10 é apresentada a estrutura da chamada do procedimento ATUALIZA_SUBSTANCIA que é responsável por pegar a lista de substancias que tenham que ser atualizadas com base nos ids mantidos na tabela MGP_SUBSTANCIA_JOURNAL, para após identificadas as substancias verificar no esquema mantido através do código dcb se é necessário inseri-las ou atualiza-las.

```
CREATE OR REPLACE PROCEDURE
ATUALIZA_SUBSTANCIA(CURSOR_SUBSTANCIAS_JOURNAL IN SYS_REFCURSOR) IS
BEGIN
  ...
END ATUALIZA_SUBSTANCIA;
```

Quadro 10 – Procedimento ATUALIZA_SUBSTANCIA.

Fonte: Autoria própria.

No quadro 11 é apresentada a estrutura da chamada do procedimento ATUALIZA_FABRICANTE que é responsável por pegar a lista de fabricantes que tenham que ser atualizadas com base nos ids mantidos na tabela MGP_FABRICANTE_JOURNAL, para após identificados os fabricantes verificar no esquema mantido através do CNPJ se é necessário inseri-las ou atualiza-las.

```
CREATE OR REPLACE PROCEDURE
ATUALIZA_FABRICANTE(CURSOR_FABRICANTES_JOURNAL IN SYS_REFCURSOR) IS
BEGIN
  ...
END ATUALIZA_FABRICANTE;
```

Quadro 11 – Procedimento ATUALIZA_FABRICANTE.

Fonte: Autoria própria.

Após a execução dos procedimentos ATUALIZA_APRESENTACAO, ATUALIZA_CONCENTRACAO, ATUALIZA_SUBSTANCIA e ATUALIZA_FABRICANTE para realizar as atualizações necessárias conforme esquema padrão. É necessário verificar se para os produtos que devem ser atualizados essas informações de apresentação, concentração, substancia e fabricantes precisam ser inseridas ou atualizadas, para realizar essa verificação no quadro 12 é apresentada a estrutura da chamada do procedimento AJUSTES_DE_PRODUTOS que é responsável por pegar a lista de produtos

que devem ser atualizadas com base nos ids mantidos na tabela MGP_PRODUTO_JOURNAL, para após identificados os produtos verificar no esquema mantido se as informações de apresentação, concentração, substancia e fabricante precisam ser inseridas ou atualizadas.

```
CREATE OR REPLACE PROCEDURE AJUSTES_DE_PRODUTOS(CURSOR_PRODUTOS_JOURNAL
IN SYS_REFCURSOR) IS
CURSOR_APRESENTACAO SYS_REFCURSOR;
CURSOR_CONCENTRACAO SYS_REFCURSOR;
CURSOR_SUBSTANCIA SYS_REFCURSOR;
CURSOR_FABRICANTE SYS_REFCURSOR;
BEGIN
  ...
  ATUALIZA_APRESENTACAO(CURSOR_APRESENTACAO);
  ATUALIZA_CONCENTRACAO(CURSOR_CONCENTRACAO);
  ATUALIZA_SUBSTANCIA(CURSOR_SUBSTANCIA);
  ATUALIZA_FABRICANTE(CURSOR_FABRICANTE);
  ...
END AJUSTES_DE_PRODUTOS;
```

Quadro 12 – Procedimento AJUSTES_DE_PRODUTOS.

Fonte: Autoria própria.

No quadro 13 é apresentada a estrutura da chamada do procedimento PRODUTO_INSERT_MANTIDO que é responsável por pegar a lista de produtos e seus códigos de barras que devam ser inseridos com base nos ids mantidos na tabela MGP_PRODUTO_JOURNAL do esquema mantido, para após identificados os produtos e seus códigos de barras verificar no esquema interno se os códigos de barras existem, caso existam inserir o id do produto do esquema padrão na tabela MGP_PRODUTO_JOURNAL para atualizar, caso contrário inserir o produto e seu código de barras no esquema padrão para serem revisados e posteriormente atualizados nos esquemas mantidos.

```
CREATE OR REPLACE PROCEDURE
PRODUTO_INSERT_MANTIDO(CURSOR_PRODUTOS_JOURNAL IN SYS_REFCURSOR) IS
CURSOR_CODIGOS_DE_BARRAS SYS_REFCURSOR;
BEGIN
  ...
END ATUALIZA_PRODUTO;
```

Quadro 13 – Procedimento PRODUTO_INSERT_MANTIDO.

Fonte: Autoria própria.

No quadro 14 é apresentada a estrutura da chamada do procedimento ATUALIZA_PRODUTO que é responsável por pegar a lista de produtos e seus códigos de barras que tenham que ser atualizados com base nos ids mantidos na tabela MGP_PRODUTO_JOURNAL, para após identificados os produtos verificar nos esquemas

mantidos através dos códigos de barras do produto se é necessário inseri-los ou atualiza-los. Nesse procedimento também é preciso identificar as apresentações, concentrações, substancias e fabricantes dos produtos para que ao realizar a atualização do produto sejam vinculadas as chaves corretas nos esquemas mantidos.

```
CREATE OR REPLACE PROCEDURE ATUALIZA_PRODUTO (CURSOR_PRODUTOS_JOURNAL IN
SYS_REFCURSOR) IS
CURSOR_CODIGOS_DE_BARRAS SYS_REFCURSOR;
CURSOR_PRODUTO_SUBSTANCIAS SYS_REFCURSOR;
APRESENTACAO_ID NUMBER;
CONCENTRACAO_ID NUMBER;
SUBSTANCIA_ID NUMBER;
FABRICANTE ID NUMBER;
BEGIN
  ...
END ATUALIZA_PRODUTO;
```

Quadro 14 – Procedimento ATUALIZA_PRODUTO.

Fonte: Autoria própria.

Os procedimentos AJUSTES_DE_PRODUTOS, PRODUTO_INSERT_MANTIDO e ATUALIZA_PRODUTO por se tratarem apenas de produtos pode-se criar um pacote para realizar um acoplamento dos procedimentos referentes aos produtos. No quadro 15 é apresentada estrutura da especificação do pacote e também o corpo do pacote com a estrutura da chamada dos procedimentos.

```
CREATE OR REPLACE PACKAGE PCT_PRODUTO AS
  PROCEDURE AJUSTES_DE_PRODUTOS;
  PROCEDURE PRODUTO_INSERT_MANTIDO;
  PROCEDURE ATUALIZA_PRODUTO;
END PCT_PRODUTO;

CREATE OR REPLACE PACKAGE BODY PCT_PRODUTO AS
  PROCEDURE AJUSTES_DE_PRODUTOS(CURSOR_PRODUTOS_JOURNAL IN
SYS_REFCURSOR) IS
    CURSOR_APRESENTACAO SYS_REFCURSOR;
    CURSOR_CONCENTRACAO SYS_REFCURSOR;
    CURSOR_SUBSTANCIA SYS_REFCURSOR;
    CURSOR_FABRICANTE SYS_REFCURSOR;
  BEGIN
    ...
    ATUALIZA_APRESENTACAO(CURSOR_APRESENTACAO);
    ATUALIZA_CONCENTRACAO(CURSOR_CONCENTRACAO);
    ATUALIZA_SUBSTANCIA(CURSOR_SUBSTANCIA);
    ATUALIZA_FABRICANTE(CURSOR_FABRICANTE);
    ...
  END AJUSTES_DE_PRODUTOS;

  PROCEDURE PRODUTO_INSERT_MANTIDO(CURSOR_PRODUTOS_JOURNAL IN
SYS_REFCURSOR) IS
    CURSOR_CODIGOS_DE_BARRAS SYS_REFCURSOR;
  BEGIN
    ...
  END ATUALIZA_PRODUTO;
```

```
  PROCEDURE ATUALIZA_PRODUTO (CURSOR_PRODUTOS_JOURNAL IN SYS_REFCURSOR)
IS
    CURSOR_CODIGOS_DE_BARRAS SYS_REFCURSOR;
    CURSOR_PRODUTO_SUBSTANCIAS SYS_REFCURSOR;
    APRESENTACAO_ID NUMBER;
    CONCENTRACAO_ID NUMBER;
    SUBSTANCIA_ID NUMBER;
    FABRICANTE_ID NUMBER;
  BEGIN
    ...
  END ATUALIZA_PRODUTO;
END PCT_PRODUTO;
```

Quadro 15 – Procedimento PCT_PRODUTO.

Fonte: Autoria própria.

Com os procedimentos da solução criados faz-se necessário a mensuração das métricas junto aos procedimentos. As definições apresentadas nas literaturas são com base em linguagens de programação orientadas a objetos, para aplica-las nos procedimentos armazenados precisa-se realizar uma leitura diferente das métricas.

As classes em linguagens de programação orientada a objetos têm a possibilidade de dar entrada e saída de parâmetros (objetos). Os procedimentos em PL/SQL também têm a possibilidade de dar entrada e saída de parâmetros, sendo assim, para a aplicação da métrica *Total Number of Modules or Classes*, identificou-se que os procedimentos têm o mesmo peso que uma classe.

Os métodos de linguagens de programação orientada a objetos têm a possibilidade de dar entrada e saída de parâmetros do mesmo modo acontece em *procedures* PL/SQL, sendo assim, para a aplicação da métrica ANPM, identificou-se que os parâmetros das *procedures* são equivalentes aos parâmetros de métodos.

Os resultados obtidos ao aplicar as métricas nos procedimentos apresentados podem ser observados na tabela 1.

Métrica	**Valor**	**Observação**
Total Number of Modules or Classes	17	Sendo 10 triggers e 7 procedures três dessas em um pacote.
NOA (Number of Attributes)	11	O procedimento AJUSTES_DE_PRODUTOS tem 4, PRODUTO_INSERT_MANTIDO tem 1 e ATUALIZA_PRODUTO contém 6 atributos.
ANPM (Average Number of Parameters per Method)	1	Cada procedimento tem apenas um parâmetro de entrada.

ACC (Afferent Connections per Class)	4	Regular.
COF (Coupling Factor)	0,0147	Bom.
SC (Structural Complexity)	Satisfatório	O resultado dessa métrica se dá pela observação da estrutura do código desenvolvido.

Tabela 1 – Resultados das métricas.

Fonte: Autoria própria.

O resultado obtido para a métrica Total Number of Modules or Classes deu-se por meio da soma de todos os procedimentos criados para a solução, sendo 10 triggers e 7 procedures três dessas em uma package.

Na aplicação da métrica NOA o procedimento AJUSTES_DE_PRODUTOS conta com 4 atributos pois os mesmos são utilizados nos procedimentos invocados a partir deste, tornando-os necessários para a realização dos ajustes das informações ligadas aos produtos. O procedimento PRODUTO_INSERT_MANTIDO conta com 1 atributo, uma vez que é necessário realizar a verificação dos códigos de barras do produto que se deseja atualizar, pois os códigos de barras de um produto podem estar ligados em diferentes produtos em outros esquemas. E por fim, o procedimento ATUALIZA_PRODUTO contém 6 atributos para realizar as ligações entre as chaves das informações para cada produto que será atualizado ou inserido.

Devido a cada procedimento ter seu interesse reduzido a apenas solucionar um pequeno problema, os mesmos precisaram apenas de um parâmetro de entrada para realizar a solução que se propunha, ficando assim o resultado 1 para média de parâmetros por procedimento (ANPM).

O resultado obtido para a métrica ACC deu-se pelo somatório de ligações entre os procedimentos, sendo assim, temos 4 procedimentos que fazem uma ligação cada, sendo eles ATUALIZA_APRESENTACAO, ATUALIZA_CONCENTRACAO, ATUALIZA_SUBSTANCIA e ATUALIZA_FABRICANTE todos realizando ligação com o procedimento AJUSTES_DE_PRODUTOS. Esse resultado é considerado regular segundo intervalos sugeridos pelas bibliográficas que são, até 2 (bom), entre 2 e 20 (regular) e acima de 20 (ruim).

O resultado obtido para a métrica COF deu-se pelo somatório de ligações entre os procedimentos que é 4, dividido pelo número possível de ligações (272), sendo assim, temos

0,0147. Esse resultado é considerado bom segundo intervalos sugeridos pelas bibliográficas que são, até 0,02 (bom), entre 0,02 e 0,14 (regular) e acima de 0,14 (ruim).

Ao observar as estruturas criadas e os resultados obtidos através das métricas é possível observar que existe alto grau de coesão nos procedimentos criados para a solução, sendo assim, o resultado para a métrica SC é satisfatório pois a solução atende o seu proposito e principalmente acolhe o princípio da separação de interesses e aprova-se através dos resultados obtidos pelas métricas aplicadas.

6 CONSIDERAÇÕES FINAIS

Com a aplicação da separação de interesses, cada problema pode ser resolvido de forma separada, sem interferir no funcionamento correto de outra parte da aplicação. Pois cada parte desta separação quando criada, tem como objeto resolver em partes únicas em determinado problema.

Fazendo uso do princípio da separação de interesses para a construção de procedimentos armazenados em banco de dados Oracle, fica evidenciado que a aplicação do princípio é eficiente, sendo estes procedimentos claros e de fácil manutenção.

Quando aplicadas a métrica de tamanho e as métricas estruturais em estudo, fica evidenciado que a utilização do princípio da separação de interesses em procedimento armazenados deixa o código com um tamanho e com uma estrutura organizada possibilitando assim uma facilidade de manutenção, flexibilidade, reusabilidade e confiabilidade.

Ao aplicar as métricas de acoplamento fica evidenciado que a utilização dos procedimentos armazenados é eficaz, pois cada procedimento tem seu objetivo muito bem definido e limitado, deixando assim um nível de acoplamento relativamente bom.

E quando aplicada a métrica de coesão fica ainda mais evidenciado que quando os interesses são bem definidos o acoplamento é mínimo e a coesão é grande, deixando assim o código com uma baixa complexidade estrutural, onde as manutenções e/ou melhorias não serão onerosas.

Os resultados obtidos através das métricas aplicadas nos procedimentos armazenados, onde houve o uso do princípio da separação de interesse, é possível concluir que a utilização dos procedimentos é eficaz, pois o nível de complexidade do código é baixo, é altamente coeso e com baixo acoplamento.

REFERÊNCIAS BIBLIOGRÁFICAS

ANTONIO, E. A., Disponível em: <http://www.devmedia.com.br/codigo-no-banco-vs-codigo-na-aplicacao-dez-variaveis-que-influenciam-na-tomada-de-decisao/17419>. Acesso em: 05 jan. 2015.

BACCARO, M., Separation of Concerns. Disponível em: <http://marcobaccaro.wordpress.com/2013/06/19/separation-of-concerns/>. Acesso em: 01 set. 2014.

BANSIYA, J.; DAVI, C., Using qmood++ for object-oriented metrics. Dr. Dobb's Journal, Dezembro 1997.

BANSIYA, J.; DAVIS, C. G., A Hierarchical Model for Object-Oriented Design Quality Assessment, IEEE Transactions on Software Engineering, Vol. 28, No. 1, Janeiro 2002. P 4-17.

CZARNECKI, K.; EISENECKER, U. Generative Programming: Methods, Tools, and Applications. New York: Addison-Wesley Professional, 2000.

DATE, C. J., Introdução a sistemas de banco de dados. Elsevier Brasil, 1985.

DIJKSTRA, E. W., Disponível em: <http://www.cs.utexas.edu/users/EWD/transcriptions/EWD04xx/EWD447.html>. Acesso em: 20 ago. 2014.

ELMASRI, R.; NAVATHE, S., Sistemas de banco de dados. 6ª ed. São Paulo: Pearson Addison Wesley, 2011.

FOWLER, M., Patterns of Enterprise Application Architecture. Addison-Wesley Professional, 2003.

GUPTA, A., Java EE 6 – Pocket Guide. 1ª ed. USA: O'Reilly Media, Inc., 2012.

ISO/IEC 25010:2011. Disponível em: < https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en>. Acesso em: 05 jan. 2014.

JONES, T. C., Applied Software Measurement: Assuring Productivity and Quality. New York: McGraw-Hill, 1991.

MAKABEE, H., Separation of Concerns. Disponível em: <http://effectivesoftwaredesign.com/2012/02/05/separation-of-concerns/>. Acesso em: 28 ago. 2014.

MARCONI, M. A.; LAKATOS, E. M., Metodologia do trabalho científico. 7ª ed. São Paulo: Atlas, 2007.

MARTINS, C., Princípios de projeto. Disponível em: <http://celsoavmartins.blogspot.com.br/2009/05/principios-de-projeto-parte-iv.html>. Acesso em: 27 ago. 2014.

McCABE, T. J., A complexity measure. IEEE Transactions Software Engineering, v. 2, n. 4, p. 308-320, dezembro 1976.

MEIRELLES, P. R. M., Monitoramento de métricas de código-fonte em projetos de software livre. Disponível em: <http://test.stoa.usp.br/articles/0030/6046/tesePauloMeirelles.pdf>. Acesso em: 05 jan. 2015.

MILLS, E. E., Software Metrics. SEI – Carnegie Mellon University, 1988.

ORACLE. Developing and Using Stored Procedure. Disponível em: <http://docs.oracle.com/cd/B28359_01/appdev.111/b28843/tdddg_procedures.html>. Acesso em: 01 set. 2014.

ORACLE. Introduction to Oracle Supplied PL/SQL Packages & Types. Disponível em: <http://docs.oracle.com/cd/ B28359_01/appdev.111/b28419/intro.htm#BABBEAGJ>. Acesso em: 31 ago. 2014.

PARNAS, D. L., On the criteria to be used in decomposing systems into modules. Communication of the ACM. Vol 15 No. 12. New York: ACM, 1972. P 1053 – 1058. Disponível em: < https://www.cs.umd.edu/class/spring2003/cmsc838p/Design/criteria.pdf >. Acesso em: 05 fev. 2015.

PICOLO, L., Engenharia de software Reusabilidade. Disponível em: <http://pt.slideshare.net/luizpicolo/engenharia-de-softwares-reusabilidade>. Acesso em: 01 set. 2014.

PRESSMAN, R. S., Engenharia de Software uma abordagem profissional. 7ª Ed. Porto Alegre: Bookman, 2011.

RICHTER, Charles. Designing Flexible Object-Oriented Systems with UML. New Riders Publishing, Thousand Oaks, CA, USA. ISBN 1578700981.

ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C., Qualidade de software: teoria e prática. São Paulo: Prentice Hall, 2001.

SOMMERVILLE, I., Engenharia de Software, 8ª Ed. São Paulo: Addison-Wesley Brasil, 2007.

TEMPERO, Ewan. On measuring java software. ACSC2008. Wollongong, Australia. Conferences in Research and Practice in Information Technology (CRPIT), v. 74, 2008.

TROSIN, A., Separation of Concern vs Single responsability. Disponível em: http://weblogs.asp.net/arturtrosin/archive/2009/01/26/separation-of-concern-vc-single-responsability-principle-soc-vs-srp.aspx. Acesso em: 29 set. 2014.

WATSON, J., OCA Oracle Database 11g: administração I: guia do exame IZO-052. Porto Alegre: Bookman, 2010.

ZUSE, H., Software Complexity: measures and methods. Walter de Gruyter, 1990.