CARLOS EDUARDO LINO

REESTRUTURAÇÃO DE SOFTWARE COM ADOÇÃO DE PADRÕES DE PROJETO PARA A MELHORIA DA MANUTENIBILIDADE

LAVRAS - MG
2011

CARLOS EDUARDO LINO

REESTRUTURAÇÃO DE SOFTWARE COM ADOÇÃO DE PADRÕES DE PROJETO PARA A MELHORIA DA MANUTENIBILIDADE

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Sistemas de Informação para obtenção do título de Bacharel em Sistemas de Informação.

Orientador:

Dr. Antônio Maria Pereira de Resende

LAVRAS - MG
2011

CARLOS EDUARDO LINO

REESTRUTURAÇÃO DE SOFTWARE COM ADOÇÃO DE PADRÕES DE PROJETO PARA A MELHORIA DA MANUTENIBILIDADE

Monografia de graduação apresentada ao Departamento de Ciência da Computação da Universidade Federal de Lavras como parte das exigências do curso de Sistemas de Informação para obtenção do título de Bacharel em Sistemas de Informação.

APROVADA em 23 de Novembro de 2011.

Dr. André Luiz Zambalde UFLA

p/ Ana Paula P. Melchiori

Dr. Raphael Winckler de Bettio UFLA

Dr. Antônio Maria Pereira de Resende - UFLA

(Orientador)

LAVRAS - MG
2011

Aos meus pais, José Carlos e Maria de Fátima, a meus irmãos, Kleber e Thiago, e a minha namorada Ana Carolina.

DEDICO

AGRADECIMENTOS

Agradeço a meus pais, José Carlos e Maria de Fátima, e minha família pelo apoio e incentivo. Agradeço a meus amigos e colegas de curso pelos momentos que passamos, pelas conversas e discussões que muito me ajudaram. Agradeço minha namorada Ana Carolina por me dar apoio e estar do meu lado. Agradeço ao meu orientador Antônio Maria por me apoiar e me aconselhar durante a realização deste trabalho.

RESUMO

A manutenibilidade de software é um requisito desejado durante o processo de desenvolvimento. Muito tem se falado sobre a adoção de padrões de projeto para melhorar a qualidade de sistemas orientados a objetos. Neste trabalho, foi definida a necessidade de estudos que comprovem a eficácia da adoção de padrões de projeto na melhoria da manutenibilidade, por meio da análise de medidas obtidas pela aplicação de métricas de software.

Realizou-se um estudo de caso com um sistema comercial em Java, o qual foi reestruturado por meio da aplicação de padrões de projeto visando melhorar a qualidade do código. Essa reestruturação contou com a adoção dos padrões de projeto DAO, MVC, Singleton e Factory.

Após a reestruturação, métricas de manutenibilidade foram aplicadas nas duas versões do sistema, legado e reestruturado, visando analisar e comparar as características relacionadas à manutenibilidade. Após coletar as medidas, perceberam-se melhorias no software reestruturado, obtendo-se menor complexidade, menor tamanho, melhor nível de abstração, dentre outros. Obtiveram-se também dados teóricos que indicam que a reestruturação proporcionou ao software melhor manutenibilidade, sendo necessário menor esforço para realizar a manutenção obtendo-se menor propagação dos impactos das modificações.

Pretendeu-se evidenciar, a partir de um estudo de caso, que padrões de projeto podem melhorar a manutenibilidade de software, e os dados encontrados indicam que no caso estudado favoreceram para a melhoria da manutenibilidade.

Palavras-chave**: manutenibilidade, padrões de projeto, métricas**.

ABSTRACT

The maintainability of software is a desired requirement during the development process. Much has been said about the adoption of design standards to improve the quality of object-oriented systems. In this study, we defined the need for studies to prove efficacy of the adoption of design standards to improve the maintainability, through the analysis of measurements obtained by the application of software metrics.

Was conducted a case study with a trading system in Java, which was restructured by applying design patterns to improve the quality of the code. This restructuring included the adoption of the DAO, MVC, Singleton and Factory design patterns.

After the refactoring, maintainability metrics were applied to the two versions of the system, legacy and restructured in order to analyze and compare the features related to maintainability. After collecting the measures, improvements were realized in software restructured, resulting in reduced complexity, smaller size, higher level of abstraction, among others.

We obtained theoretical data also indicate that the restructuring provided the best software maintainability, requiring less effort to perform maintenance resulting in lower spread of the impacts of changes.

It was intended to evince from a case study that design patterns can improve the maintainability of software, and our data indicate that in the case study contributed to improved maintainability.

Keywords**: maintainability, design patterns, metrics**.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

ABS	*Abstraction* (Abstração)
AJAX	*Asynchronous JavaScript and XML*
avMLOC	Média de linhas de código por método
avPAR	Média de número de parâmetros
CBO	*Coupling Between Object Classes* (Acoplamento entre objetos)
CIP	Clientes do Instituto de Pesquisa
CYCLO	*Cyclomatic Complexity* (Complexidade ciclomática)
DAO	*Data Access Object* (Objeto de acesso a dados)
DIH	*Depth of the Inheritance Hierarchy* (Profundidade da árvore de herança)
HTM	Hipertext Markup Language
IP	Instituto de Pesquisa
JDBC	Java DataBase Connectivity
JPA	Java Persistence API
LCOM	Lack of Cohesion in Methods (Falta de coesão em métodos)
LOC	Line of Code (Linhas de Código)
MACSOO	Modelo de Avaliação de Conectividade em Sistemas Orientados por Objetos
MI	Maintainability Index (Índice de Manutenibilidade)
MVC	Model-View-Controller
NOC	Number of Children (Número de Filhos)
OO	Orientação a Objetos
ORM	Object-Relational Mapping (Mapeamento Objeto-Relacional)
RFC	Response For a Class (Resposta de uma classe)
RM	Requisição de Modificação
SGBD	Sistema Gerenciador de Banco de Dados
SQA	Software Quality Assurance (Garantia de Qualidade de Software)
URI	Uniform Resource Identifier

WMC Weighted Methods per Class (Métodos ponderados por classe)

SUMÁRIO

1. INTRODUÇÃO

1.1 Contextualização e Motivação

Em um ambiente instável, a manutenção do software é feita frequentemente, o que pode gerar custos muito altos, tanto pelo custo direto da manutenção quanto pelo custo da indisponibilidade do sistema, (XIONG; XIE; NG, 2011). Devido à necessidade contínua de adaptação de sistemas computacionais, a indústria de software busca ferramentas, métodos e processos que auxiliem na redução de custos durante a manutenção de software. Durante o desenvolvimento de um software, deve ser levado em consideração o contexto em que o sistema será utilizado, e quais são as mudanças que podem ocorrer no ambiente ou na tecnologia que poderão implicar em uma manutenção no sistema, para que ele continue funcionando de forma eficiente.

A adoção de padrões de projetos de software busca melhorar a legibilidade do software para facilitar o entendimento pelos mantenedores do sistema, sendo considerado como uma parte da documentação do projeto, (GAMMA *et al.*,1995). Espera-se da adoção de padrões de projeto, dentre outras características, melhor reusabilidade, extensibilidade e manutenibilidade, pois fornecem soluções mais completas e melhor estruturadas do que uma simples solução para resolver um problema imediato. Entretanto, pode-se introduzir uma complexidade desnecessária em um código, usando uma solução complexa para um problema simples (VOKÁČ *et al*.,2004)

Segundo BENNETT e RAJLICH (2000), a maioria das pesquisas em Ciência da Computação são voltadas para o desenvolvimento inicial do software. Devido a falta de um consenso sobre os benefícios da utilização de padrões de projeto ma qualidade do código, conforme observado por HERNANDEZ; KUBO; WASHIZAKI (2010), estudos devem ser realizados pela comunidade e pelo mercado com o propósito de analisar, por meio de

aplicação de métricas, a viabilidade da adoção de padrões de projeto para a melhoria da manutenibilidade de software.

1.2 Problema de Pesquisa

Diante da necessidade em melhorar a capacidade de manutenção dos sistemas, define-se o problema de pesquisa a ser tratado neste trabalho:

"*a comunidade de engenharia de software, científica, tecnológica e do mercado, necessitam de mais estudos demonstrando quantitativamente, por meio das métricas de software, os ganhos que a adoção de padrões de projetos, utilização de framework e o uso eficiente da orientação a objetos podem promover na melhoria da manutenibilidade de software em sistemas orientados a objetos*[1]".

Para buscar informações que possam contribuir com o problema identificado, a comunidade científica e tecnológica deve:

"*realizar estudos para quantificar o impacto da utilização de padrões de projetos na manutenibilidade de software, através de modelos para avaliação de manutenibilidade e realizando a analise de um caso real*".

1.3 Objetivos do Trabalho

1.3.1 Objetivo Geral

Neste trabalho de tencionou-se contribuir com a comunidade científica e tecnológica na quantificação do impacto de padrões de projeto na manutenibilidade do sistema. Para isto, definiu-se como objetivo geral:

"*Analisar o impacto da aplicação de padrões de projetos na manutenibilidade de um software comercial em Java*".

---

[1] A orientação a objetos é um paradigma de programação que busca abordar a resolução de um problema através da análise das entidades e seus relacionamentos

1.3.2 Objetivos Específicos

Os seguintes objetivos específicos foram definidos, a fim de alcançar o objetivo geral proposto:

1. Pesquisar artigos e livros, atualizando o conhecimento referente às últimas pesquisas na área deste trabalho de pesquisa;
2. Pesquisar ferramentas, modelos e métricas de software para avaliação da manutenibilidade em sistemas desenvolvidos utilizando a Orientação a Objetos (OO);
3. Analisar um sistema comercial em Java sem a utilização de padrões de projeto;
4. Avaliar que melhorias podem ser aplicadas no sistema;
5. Gerar um sistema reestruturado, adotando padrões de projetos e framework;
6. Avaliar quantitativamente a melhoria da manutenibilidade por meio de métricas de software.

1.4 Estrutura do Trabalho

Este trabalho está estruturado da seguinte maneira:

- Capítulo 2 apresenta o Referencia Teórico, mostrando os principais conceitos utilizados no trabalho como manutenção de software, métricas de software e padrões de projeto. É apresentado também modelos para avaliação de manutenibilidade de software.
- Capítulo 3 apresenta a Metodologia do trabalho, detalhando a sequência de atividades, a maneira de condução do trabalho e o modo como os conceitos foram aplicados para alcançar o resultado esperado.

- Capítulo 4 apresenta os resultados deste trabalho de conclusão de curso, mostrando quais foram os resultados finais.
- Capítulo 5 apresenta a conclusão do trabalho realizado e as considerações finais, bem como trabalhos futuros.

2. REFERENCIAL TEÓRICO

A indústria de software vem buscando métodos e processos que proporcionem ganhos na qualidade do processo de desenvolvimento de software. Os processos de desenvolvimento descrevem as etapas para o desenvolvimento de software, que vão desde a especificação dos requisitos, desenvolvimento, até entrega do produto final. O desenvolvimento do software é a etapa em que se analisam os requisitos especificados para que o trabalho possa ser dividido entre os programadores, cuja função é transformar o design, anteriormente concebido, em código (ALMEIDA; DE MIRANDA, 2010).

Mas o trabalho dos desenvolvedores não se encerra após a produção correta do código. É preciso considerar a necessidade de que à medida que surjam novos requisitos, alterações ou descoberta de falhas, a equipe aprimore o código.

2.1 Manutenção de Software

Segundo as normas IEEE Std 610.12 (1990) e IEEE Std 1219-1998 (1998), a manutenção de software é definida como o processo de modificação em um produto de software após a entrega para correção de falhas, para melhorar o desempenho ou outros atributos de qualidade, ou para adaptar o produto a modificações do ambiente. SOMMERVILLE (2007) considera que a manutenção de software é um processo de modificação de um sistema após ser entregue. As mudanças podem ser desde a correção de um trecho de código, adição de um novo componente até a reestruturação do software.

A norma IEEE Std 14764-2006 (2006) categoriza a manutenção do software em quatro grupos:

- Manutenção adaptativa: é a modificação em um produto de software após a entrega para manter o produto utilizável após mudanças no ambiente;
- Manutenção corretiva: é a modificação em um produto de software para corrigir falhas descobertas após a entrega;
- Manutenção preventiva: é a modificação de um produto para corrigir defeitos no software, detectados após a entrega, antes que causem falhas operacionais;
- Manutenção de aperfeiçoamento: é a modificação de um produto de software após a entrega para aperfeiçoar o software quando os requisitos do software modificam em resposta as mudanças da organização ou as regras de negócio. Proporciona também melhorias para o usuário, melhoria da documentação, desempenho ou segurança ou outros atributos de software.

O processo de manutenção deve ser realizado visando garantir a qualidade do software existente.

As atividades de manutenção de software consomem consideráveis recursos para implementar mudanças em sistemas existentes.

Dados mostram que reduzir os custos da manutenção de software é indispensável, e que tal redução pode ser obtida principalmente quando o software possui características de qualidade que tornem possível realizar alterações no software de forma mais fácil, (FERREIRA; BIGONHA; BIGONHA, 2008).

2.1.1 Qualidade de Software

Segundo a norma IEEE Std 610.12 (1990), a qualidade é "*o grau em que um sistema, componente ou processo satisfaz os requisitos especificados*". Uma segunda definição da mesma norma é "*o grau em que um sistema, componente ou processo atende as necessidades ou expectativas*

dos cliente ou usuário". PRESSMAN (2006) define a qualidade de software como a conformidade do software com os requisitos e as normas explicitamente declaradas, que são esperadas em todo software desenvolvido profissionalmente. Pode-se considerar que um software de qualidade é o software correto para o usuário, contém as características desejadas para o seu propósito, e é feito de forma correta, seguindo requisitos e normas de qualidade visando, além da eficácia, melhor eficiência do produto final.

A garantia da qualidade de software, do inglês *Software Quality Assurance* (SQA), é um processo tratado por diversos autores, como PRESSMAN (2006) e SOMMERVILLE (2007), que definem como a característica qualidade será tratada, e quais atividades serão exercidas para que a qualidade do software seja obtida.

A norma ISO/IEC 9126-1:2001 (2001) estabelece um modelo de qualidade de software relacionado ao produto de software, ao processo de desenvolvimento e a qualidade em uso.

- Processo: relacionado à qualidade do processo de desenvolvimento do software, e contribui para melhorar a qualidade do produto;
- Produto: refere-se aos atributos de qualidade do produto, que podem ser externos ou internos, e contribui para melhorar a qualidade em uso;
- Qualidade em uso: refere-se à qualidade em uso percebida pelo usuário.

A avaliação da qualidade do produto é um processo do ciclo de vida do software que procura satisfazer as necessidades de qualidade de software por meio da mensuração de atributos internos (como medidas estáticas de produtos intermediárias), de atributos externos (como medição do comportamento da execução do código) além dos atributos da qualidade em uso (ISO/IEC 9126-1:2001, 2001).

Os atributos de qualidade externos e internos devem ser abordados durante o processo de desenvolvimento de software, pois permitem a entrega de um software melhor para o cliente.

A qualidade interna está relacionada com a qualidade do software do ponto de vista interno, como modelo estático e dinâmico, outros documentos e código-fonte, e podem ser utilizados como metas para validação durante a fase de desenvolvimento, por meio de métricas internas.

A qualidade externa está relacionada à qualidade do produto final do software, quando ele é executado, e pode ser medida e avaliada durante testes num ambiente controlado, com dados simulados e usando métricas externas.

A norma ISO/IEC 9126-1:2001 (2001) distribui os atributos de qualidade em seis categorias principais, cada uma com subcategorias, conforme a **Erro! Fonte de referência não encontrada.**. Uma das ategorias que está diretamente relacionada com a manutenção do software é a manutenibilidade, que aborda características desejáveis em um software para facilitar o processo de manutenção.

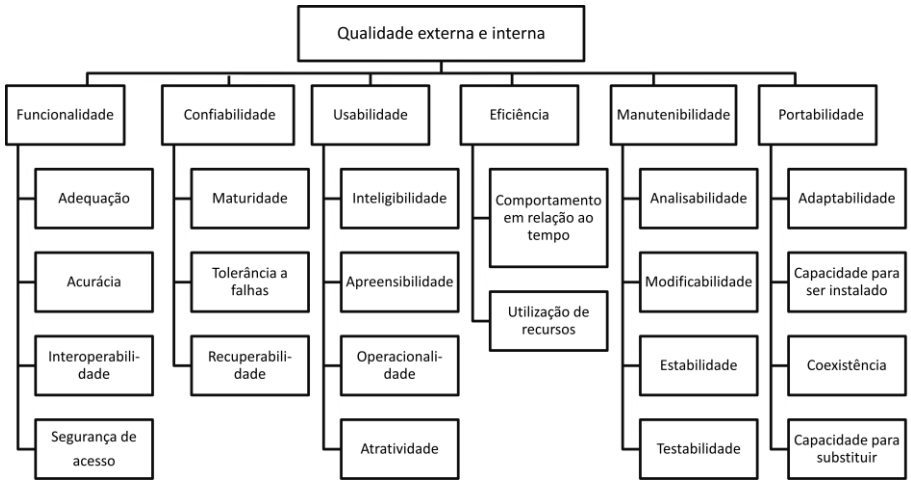

Figura Modelo de qualidade para a qualidade externa e interna.
Fonte: Adaptado de ISO/IEC 9126-1:2001 (2001).

2.1.2 Manutenibilidade de Software

A manutenibilidade é a facilidade com que um sistema ou componente de software pode ser modificado para corrigir falhas, melhorar o desempenho ou outros atributos, ou adaptar a uma modificação do ambiente (IEEE Std 610.12, 1990). A norma ISO/IEC 9126-1:2001 (2001) define a manutenibilidade de software como "*Capacidade do produto de software de ser modificado"*. As modificações podem incluir correções, melhorias ou adaptações do software devido a mudanças no ambiente e nos seus requisitos ou especificações funcionais. A manutenibilidade é dividida em cinco subcaracterísticas:

- Analisabilidade: Capacidade do produto de software de permitir o diagnóstico de deficiências ou causas de falhas no software, ou a identificação de partes a serem modificadas;
- Modificabilidade: Capacidade do produto de software de permitir que uma modificação especificada seja implementada;
- Estabilidade: Capacidade do produto de software de evitar efeitos inesperados decorrentes de modificações no software;
- Testabilidade: Capacidade do produto de software de permitir que o software, quando modificado, seja validado, e
- Conformidade: Capacidade do produto de software de estar de acordo com normas ou convenções relacionadas à manutenibilidade.

Segundo (PRESSMAN, 2006), McCall também trata sobre os fatores de qualidade, e os categoriza em dois grupos: os que podem ser medidos diretamente, e o que são medidos indiretamente. Além disso, McCall separa os fatores de qualidade de acordo com três aspectos do produto: características operacionais, capacidade de modificação e

capacidade de adaptação, Figura 1. A manutenibilidade é um fator relacionado à revisão do produto, que não pode ser medido diretamente.

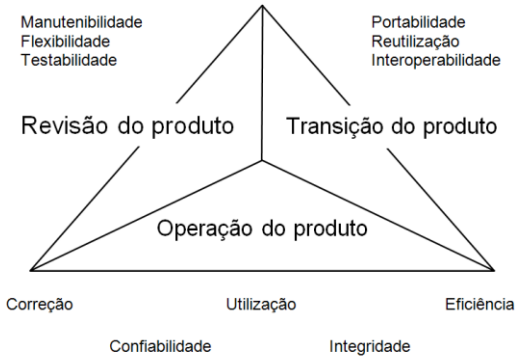

Figura 1 Fatores de qualidade de software de MCCall.
Fonte: Adaptado de PRESSMAN (2006).

A norma ISO/IEC 9126-1:2001 (2001) é um padrão que tem várias aplicações para auxiliar no gerenciamento da qualidade de software, pois, no caso da manutenibilidade por exemplo, "*considera a existência ou não de fatores considerados importantes para a manutenção*" (FERREIRA, 2011). Entretanto, a norma ISO/IEC 9126-1:2001 (2001) e os fatores de qualidade de McCall não consideram fatores estruturais de um software, como modularidade, coesão e acoplamento.

O Modelo FURPS desenvolvido pela Hewlett-Packard, ilustrado na Figura 2, considera os fatores de qualidade funcionalidade, utilização, confiabilidade, desempenho e suportabilidade, e relaciona cada um dos fatores quais métricas de software estão relacionadas (PRESSMAN, 2001). A modularidade, a consistência, a instrumentação, a autodocumentação e a simplicidade são fatores de qualidade analisados para medir a manutenibilidade.

Métrica de qualidade de software / Fator de qualidade	Correção	Confiança	Eficiência	Integridade	Manutenibilidade	Flexibilidade	Testabilidade	Portabilidade	Reusabilidade	Interoperabilidade	Usabilidade
Auditabilidade				x			x				
Precisão		x									
Comunicação comunalidade										x	
Plenitude	x										
Complexidade		x				x	x				
Concisão			x		x	x					
Consistência	x	x			x	x					
Comunalidade de dados										x	
Tolerância a erros		x									
Eficiência de execução			x								
Expansibilidade						x					
Generabilidade						x		x	x	x	
Independência de hardware								x	x		
Instrumentação				x	x		x				
Modularidade		x			x	x	x	x	x	x	
Operabilidade			x								x
Segurança				X							
Autodocumentação					x	x	x	x	x		
Simplicidade		x			x	x	x				
Independência de sistema								x	x		
Rastreabilidade	x										
Treinamento											x

Figura 2 Relação entre fatores dede qualidade e métricas de software.
Fonte: Adaptado de PRESSMAN (2001).

A modularidade é um dos fatores de qualidade que podem ser avaliados em sistemas OO, pois o paradigma define que cada classe deva representar uma entidade, com atributos e funções bem definidas. Com isso, a modularidade é um reflexo da boa implementação de um sistema OO.

2.1.3 Modularidade

A modularidade do software pode proporcionar que alterações no software sejam realizadas mais facilmente e que não impactem em todo o sistema. Em sistemas estáveis pode-se alterar uma propriedade de um

módulo sem que os demais sejam influenciados, ou alterando o mínimo possível.

Segundo a norma IEEE Std 610.12 (1990), para um sistema composto por diversos componentes, a modularidade é determinada como sendo o grau que cada componente pode ser modificado causando o mínimo impacto nos demais. Pode-se conseguir modularidade reduzindo o acoplamento e aumentando a coesão de um módulo ou classe.

2.1.4 Coesão

Segundo FERREIRA (2011), a coesão do software é o grau de comunicação entre os elementos internos de um módulo. Podemos definir também como "*a forma e o grau em que o tarefas executadas por um único módulo de software são relacionados entre si*", (IEEE Std 610.12, 1990).

Em sistemas desenvolvidos utilizando o paradigma da OO, a coesão indica que um módulo, ou classe, depende menos dos outros componentes, e mais de sí mesmo. A modularidade é uma característica desejável na qualidade de software e favorece na melhoria da manutenibilidade, pois modificações em um módulo mais coeso impactam menos outras partes do software. Cada classe abstrai a sua implementação, uma alteração na lógica interna de um método não causa menor impacto nas classes dependentes. Além disso, um módulo com alta coesão é mais fácil de ser testado.

2.1.5 Acoplamento

O acoplamento é uma medida que relaciona o grau de dependência entre dois módulos. A norma IEEE Std 610.12 (1990) define acoplamento como sendo "*forma e o grau de interdependência entre os módulos de software*".

Em sistemas que são estruturados em módulos, deve-se criar interfaces bem definidas para a comunicação entre os diversos módulos de

um sistema, reduzindo assim o impacto que a modificação de um modulo possa causar no sistema. Em sistemas desenvolvidos utilizando o paradigma da OO, deve-se evitar que uma classe acesse diretamente os atributos de outra. Cada classe deve ser coesa, e disponibilizar métodos para manipular os atributos, encapsulando-os.

2.2 Métricas de Software

Para obter dados que possam mensurar características do software, a engenharia de software costuma utilizar métricas, tanto na fase de desenvolvimento quanto na fase de manutenção.

Tanto na fase de desenvolvimento quanto na fase de manutenção, a engenharia de software analisa características do software para avaliar sua qualidade. Para se mensurar quantitativamente os atributos de qualidade de um software são utilizadas métricas de software. Uma métrica é um método para determinar se um sistema, componente ou processo possui certo atributo (IEEE Std 610.12, 1990). Uma vez que o software é um produto intangível, as métricas de software proporcionam uma forma de mensurar algumas características de qualidade do software. Os resultados podem ser, por exemplo, usados para prever custos de manutenção ou melhorias na qualidade do código (MÄKELÄ; LEPPÄREN 2007).

Vários estudos científicos buscam avaliar a utilização de métricas durante o desenvolvimento do software como forma de estimar custos e prazos, além de auxiliando na melhoria da qualidade do produto final. Tais métricas e padrões de comparação devem ser definidos pela equipe de desenvolvimento e adaptada ao contexto e escopo do projeto.

2.2.1 Métricas de projeto de software

As métricas de projeto de software auxiliam durante o processo de desenvolvimento de um software fornecendo dados sobre o andamento do

projeto e sobre o produto em desenvolvimento. As métricas podem ser utilizadas como indicadores de qualidade do produto final.

Embora vários autores tenham buscado definir uma métrica capaz de avaliar por si só a qualidade de software, ainda não existe uma só métrica que possa quantificar essa característica. Para isso, temos um conjunto de métricas que podem ser aplicadas em diversos níveis e fases do projeto de software. PRESSMAN (2006) categorizou diversas métricas que estão relacionadas com o projeto do software. Dentre elas podem-se citar as métricas para modelo de projeto OO, métricas orientadas à classe, e métricas de projeto em nível de componente. Existem diversas métricas de software que podem ser utilizadas em situações específicas.

2.2.1.1 Métricas para modelo de projeto OO

PRESSMAN (2006) cita nove características relacionadas ao projeto orientado a objetos que podem ser mensuradas a partir de métricas. Tais informações podem ser benéficas para o desenvolvimento do software, pois proporcionam uma visão objetiva do projeto, fornecendo dados sólidos tanto ao projetista novato quanto ao experiente. As métricas são utilizadas para mensurar as seguintes características:

- Tamanho: está relacionada ao volume total de um software, podendo ser medido pelo número de linhas, número de classes, dentre outros fatores.
- Complexidade: está relacionada com a complexidade computacional, além da complexidade do relacionamento entre outras classes e da complexidade de analisar e testar uma classe.
- Acoplamento: está relacionada com a dependência entre elementos de um sistema por meio de conexões.

- Suficiência: está relacionado com as características que uma classe deve possuir para ser útil, refletindo o domínio da aplicação.
- Completeza: tem implicação direta no grau de abstração de com componente de projeto.
- Coesão: a coesão de uma classe determina que todas as operações devam trabalhar juntas para atingir um propósito único bem definido.
- Primitividade: define que uma operação deve ser atômica, e uma classe deve encapsular operações primitivas.
- Similaridade: refere-se ao grau que duas classes são semelhantes, em relação à função ou estrutura.
- Volatilidade: característica relacionada à probabilidade de que o software precise ser modificado

2.2.2 Métricas orientadas à classe

As métricas orientadas à classe são aplicadas individualmente em classes, que representam uma entidade funcional do sistema. Um conjunto de métricas proposto por CHIDAMBER; KEMERER (1994), comumente chamado de *conjunto de métricas CK*, contem seis métricas relacionadas à classe, sendo elas:

- Métodos ponderados por classe (*weighted methods per class* - WMC): é calculada a partir da métrica de complexidade ciclomática de cada método, definida na seção 2.2.4.
- Profundidade da árvore de herança: (*depth of the inheritance hierarchy* - DIH): representa o comprimento máximo que uma árvore de herança possui.

- Número de filhos (*number of children* – NOC): representa a quantidade de classes filhas de primeiro grau uma classe possui.
- Acoplamento entre as classes (*coupling between object classes* - CBO): Está relacionado ao número de colaborações de uma classe, tanto aferente quanto eferente.
- Resposta de uma classe (*response for a class* - RFC): corresponde ao conjunto de métodos possíveis de serem executados em resposta a uma solicitação.
- Falta de coesão em métodos (*lack of cohesion in methods* - LCOM): está relacionado à coesão de uma classe. A métrica LCOM avalia a quantidade de métodos que acessam um mesmo atributo dentro de uma classe.

Além do conjunto de métricas CK também existem o conjunto de métricas MOOD que possui, dentre outras métricas, fator de herança de métodos (*method inhetance factor* - MIF) e fator de acoplamento (*coupling factor* - CF) (PRESSMAN, 2006).

2.2.3 Métricas de código fonte

Para projetos em nível de componente, o foco das métricas se dá a partir da análise de três características: coesão, acoplamento e complexidade do módulo.

HALSTEAD (1977) apud PRESSMAN (2006) propõe um conjunto de métricas calculadas a partir de métricas primitivas. HALSTEAD (1977) apud PRESSMAN (2006) desenvolveu expressões para calcular o tamanho do programa, o volume de um algoritmo, o volume real em bits, o nível do programa (medida de complexidade), o nível de linguagem, e outras características. Essas métricas são calculadas a partir de quatro métricas primitivas:

- n1 = número de operadores distintos;

- n2 = número de operandos distintos;
- N1 = número total de operadores;
- N2 = número total de operandos.

2.2.4 Métricas de manutenibilidade

A manutenibilidade é um fator de qualidade que não pode ser medido diretamente. Por isso pode-se utilizar várias métricas simples ou compostas para avaliar a manutenibilidade. A norma IEEE Std 1219-1998 (1998) descreve as métricas que podem ser utilizadas para avaliar os fatores de qualidade relacionando com o modelo de processo de manutenção de software. A Tabela 1 categoriza as métricas de software que relacionam com a manutenibilidade, durante todas as fases do processo de manutenção.

Tabela 1 Métrica para o processo de manutenção.
Fonte: Adaptado de IEEE Std 1219-1998 (1998).

Fase	Métricas relacionadas
Identificação do problema	- Número de omissões de requisição de modificação (RM) - Número de submissão de RM - Número de RMs duplicados - Tempo gasto para validação do problema
Análises	- Mudanças de requisitos - Taxa de erro de documentação - Esforço por área de função (SQA, SE, etc.) - Tempo decorrido (agenda) - Taxas de erro gerado por prioridade e tipo
Projeto	- Complexidade do software - Alterações de design - Esforço por área de função - Tempo decorrido - Planos de teste e procedimento de mudanças - Taxas de erro gerado por prioridade e tipo - Número de linhas de código adicionadas, excluídas, modificadas ou testadas. - Número de aplicações
Implementação	- Volume/funcionalidade (pontos de função ou linhas de código fonte) - Taxas de erro gerado por prioridade e tipo

SOMMERVILE (2007) também relaciona algumas métricas de software com o fator manutenibilidade, apresentado na Figura 3, e faz um mapeamento do fator manutenibilidade com as métricas: número de parâmetros por método, complexidade ciclomática, tamanho do programa em linhas de código e quantidade de comentários e manual do usuário.

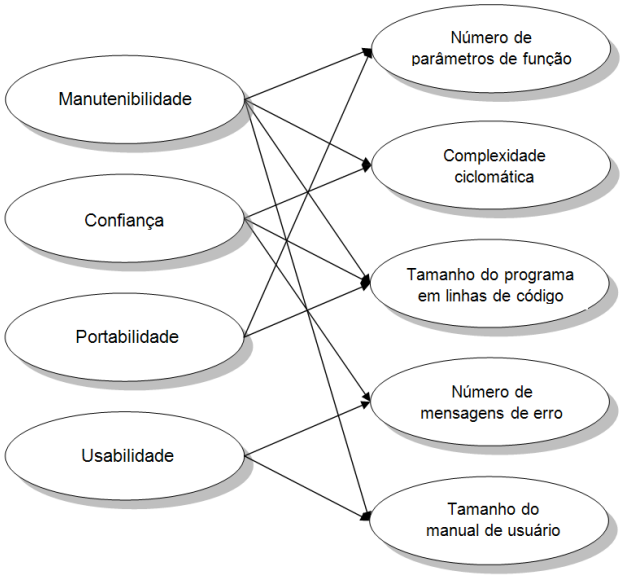

Figura 3 Relacionamento entre atributos internos e externos de software.
Fonte: Adaptado de SOMMERVILLE (2007).

Métricas de complexidade são comumente utilizadas para prever a confiabilidade e a manutenibilidade de um software (PRESSMAN, 2006). Essas métricas podem auxiliar a identificar possíveis pontos críticos no software, com alta instabilidade.

Pode-se destacar as métricas de complexidade de McCabe:

- CYCLO (*Cyclomatic Complexity*): Complexidade ciclomática é uma métrica que calcula a complexidade de um método. Seu valor mínimo é 1 (um) e não existe limite máximo. Quanto menor o valor, menos complexo é o método.

- Métodos ponderados por classe (*weighted methods per class* - WMC): é calculada a partir da média ponderada da complexidade ciclomática de cada método, definindo a complexidade da classe

As métricas de acoplamento e coesão também são utilizadas para avaliar a manutenibilidade. Pode-se observar na sessão 2.2.2 que as Métricas de CK avaliam a modularidade, calculando nível de herança, falta de coesão, número de acoplamento dentre outros fatores.

2.3 Avaliação da manutenibilidade

Existem diversas métricas e modelos que buscam avaliar o esforço necessário para a manutenção do software. Algumas dessas estudadas serão descritas a seguir.

2.3.1 Índice de Manutenibilidade

O Índice de Manutenibilidade (MI) é um índice calculado a partir de quatro métricas, e indica o nível de manutenibilidade de um software. Esse índice foi desenvolvido por OMAN; HAGEIMEISTER (1992), onde explica como os elementos do MI foram calibrados e validados pela Hewlett-Packard (HP).

O polinômio utilizado para calcular o MI é o seguinte:

*MI = 171 - 5.2 * In(aveV) - 0.23 * aveV(g') - 16.2 *In (aveLOC)*

Dado:

aveV = média do Volume (V) de Halstead por módulo;

aveV(g') = média estendida da complexidade ciclomática por módulo;

aveLOC = média do numero de linhas (LOC) por módulo;

O índice de manutenibilidade é utilizado pela indústria de software para avaliar a manutenibilidade dos sistemas, e foi adotado pelo SEI (*Software Engineering Institute*). A manutenibilidade está diretamente relacionada ao índice, quanto maior o valor, melhor é a manutenibilidade do software. Porém, o MI foi desenvolvido para analisar a manutenibilidade de software procedural, e não leva em consideração aspectos da orientação a objetos.

2.3.2 Modelo de avaliação de Conectividade

O Modelo de Avaliação de Conectividade em Sistemas Orientados por Objetos (MACSOO), proposto por FERREIRA; BIGONHA; BIGONHA (2008) é "*um método que usa métricas para avaliar os principais fatores de impacto na conectividade em sistemas OO*", indicando quais as atividades necessárias para a redução da conectividade. O modelo constitui-se de três etapas:

1. Indicação dos fatores mais importantes de impacto na conectividade;
2. Indicação de um conjunto de métricas necessárias para a avaliação dos fatores envolvidos;
3. Passos e estágios para se obter um software com menor grau de conectividade.

Estes tópicos são detalhados em FERREIRA; BIGONHA; BIGONHA (2008), que detalha também as métricas utilizadas para avaliar a conectividade, analisando os fatores: ocultação de informação, acoplamento entre classes, coesão interna de classes e profundidade da árvore de herança.

Os estágios do processo de redução da conectividade podem ser resumidos em: Avaliação da estabilidade; Avaliação da conectividade do sistema; Identificação das classes mais conectadas; Avaliação do acoplamento; Avaliação da coesão; Avaliação de ocultação de informação e Avaliação da profundidade da árvore de herança. O processo de redução da

conectividade pode ser cíclico, pois a cada ciclo é realizada uma avaliação da conectividade, avalia-se quais classes devem ser modificadas, realiza-se as modificações necessárias, e avalia-se novamente a conectividade. Este ciclo é realizado até que a avaliação da conectividade forneça uma conectividade baixa.

Para realizar a modificação das classes, é necessário realizar a reestruturação da classe visando melhorar o fator que a métrica indica contribuir para a conectividade alta.

2.3.3 Modelo de avaliação de propagação de modificações

Visando prever o impacto de modificações em sistemas orientados a objetos, FERREIRA (2011 p.135) propôs *"um modelo de predição de amplitude de propagação de modificações contratuais em software orientado por objetos, denominado K3B"*. O modelo é dado por uma expressão que envolve cinco parâmetros: quantidade de módulos, número de módulos modificados inicialmente, taxa de contaminação das modificações, taxa de melhora de modificações e grau de conectividade do sistema.

Segundo FERREIRA (2011), foi desenvolvida uma ferramenta chamada *Connecta* que possui este modelo implementado e é capaz de calcular as métricas em sistemas desenvolvidos com a linguagem Java. Os resultados de testes com a ferramenta para a validação do modelo mostram que o K3B apresenta boa eficácia ao prever o impacto de modificações em software. A métrica pode ser utilizada em substituição à simulação, por ser um processo demorado e caro.

2.4 Reestruturação de Software

A reestruturação pode ser caracterizada como reestruturação de código, ou reestruturação de dados. A reestruturação de código não modifica a arquitetura do sistema, já a reestruturação de dados é uma atividade de

reengenharia em escala plena, onde na maioria dos casos começa com a engenharia reversa e as modificações resultarão tanto em modificações arquiteturais quanto em modificações de código (PRESSMAN, 2006).

A reestruturação do software pode ser realizada a partir de um processo de refatoração (do inglês *refactoring*), que é a modificação interna de um código preservando a sua funcionalidade (FOWLER *et al*., 1999). Para HIGO; KUSUMOTO; INOUE (2008 p.436), *"a refatoração é um conjunto de ações destinadas a melhorar a manutenção compreensibilidade ou outros atributos de um sistema de software sem alterar seu comportamento externo"*.

Portanto, entende-se neste trabalho a refatoração como sendo uma reestruturação de código realizada de forma pontual, e a reestruturação como sendo o resultado do processo de refatoração e de modificações na arquitetura.

A refatoração é muito utilizada para diminuir a complexidade de um método, para melhorar a analisabilidade, aumentar o desempenho, dentre outras características. *"As métricas tem sido utilizadas a cerca de três décadas na identificação de necessidades de reestruturação"* (FERREIRA; BIGONHA; BIGONHA, 2008).

As métricas de software têm uma estreita relação com a refatoração, pois as métricas indicam onde o código deve ser melhorado, e a refatoração melhora o valor das métricas. Durante o processo de desenvolvimento, as métricas podem ser coletadas por diversas ferramentas. Essa capacidade de coletar métricas de qualidade durante o desenvolvimento auxilia o desenvolvedor a escrever um código melhor.

As métricas auxiliam encontrar as situações em que se pode aplicar uma reestruturação, adotando ou não algum padrão de projeto, visando melhoria na qualidade do código, e consequentemente da sua manutenibilidade.

2.5 Padrões de Projeto

Padrões de projeto são arquiteturas testadas para a construção de softwares orientados a objetos. Diversos problemas encontrados pelos desenvolvedores são comuns, e se repetem em diversos casos. O padrão de projeto "*fornece um conjunto de interações específicas que podem ser aplicadas a objetos genéricos para resolver um problema conhecido"*, (CRAWFORD E KAPLAN, 2003 c. 1.1). Como cada desenvolvedor pensa de uma forma diferente, dificulta o trabalho em equipe, e consequentemente a manutenção do software. As vantagens de sua adoção é a legibilidade do código, uma vez que é mais fácil analisar um algoritmo que utiliza um padrão do que uma solução criada pelo desenvolvedor. GAMMA *et al.* (1995) definiram 23 padrões de projeto e dividiram em três categorias:

- Padrões criacionais: Examinam questões relacionadas à criação dos objetos, criando um padrão de implementação para cada política necessária, por exemplo, se é preciso que somente uma instância seja criada, ou se é necessário criar tipos de objetos diferentes sem que se saiba qual o tipo que será requerido;
- Padrões estruturais: Descrevem maneiras comuns de organizar classes e objetos em um sistema;
- Padrões comportamentais: Fornecem estratégias testadas para definir o comportamento de objetos, definindo como eles irão colaborar entre si, oferecendo comportamentos especiais para uma variedade de situações.

Na Figura 4 podemos ver resumidamente os 23 padrões de projetos definidos por GAMMA *et al.* (1995).

		Finalidade		
		De criação	**Estrutural**	**Comportamental**
Escopo	**Classe**	*Factory Method*	*Adapter*	*Interpreter* *Template Method*
	Objeto	*Abstract Factory* *Builder* *Prototype* *Singleton*	*Adapter* *Bridge* *Composite* *Decorator* *Façade* *Flyweight* *Proxy*	*Chain of Responsibility* *Command* *Iterator* *Mediator* *Memento* *Observer* *State* *Strategy* *Visitor*

Figura 4 Padrões de projeto agrupados pelo escopo e pela sua finalidade.
Fonte:Adaptado de RAPELI (2006).

Para se aplicar um padrão de projeto a um software existente, é necessário identificar situações que justifiquem a sua adoção. Cada padrão de projeto especifica, dentre outras características, nome, classificação, intenção e objetivo e diagrama de classe. A seguir podem ser verificados mais detalhes dos padrões de projeto mais utilizados e que se pretende utilizar no trabalho.

2.5.1 MVC

Classificação: O padrão Model-View-Controller (MVC) é um padrão de arquitetura que visa separar as camadas de Modelo, Visão e Controle de uma aplicação (SOMEMRVILLE, 2006).

Intenção: As vantagens da utilização do MVC são: separação de elementos de visualização da regra de negócio, maior facilidade em manutenção, portabilidade para vários tipos de interface, dentre outros.

Objetivos: O padrão MVC geralmente é implementado utilizando um framework. Os frameworks MVC utilizam os padrões *Observer*, *Composite* e *Strategy*. A utilização de frameworks reduz o número de linhas do código, facilita a manutenção, e por se tratar de um componente, proporciona o reuso de código.

Diagrama:

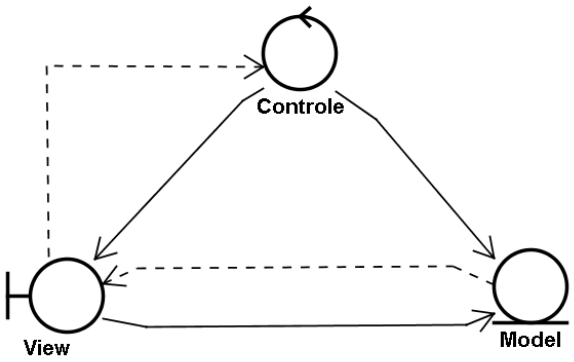

Figura 5 Diagrama de classe do padrão MVC

2.5.2 DAO

Classificação: O padrão de projeto *Data Access Object* (DAO) é um padrão de projeto relacionado aos dados e ao banco de dados (CRAWFORD E KAPLAN 2003).

Intenção: Esse padrão busca criar um mecanismo de relação entre o banco de dados e o sistema.

Objetivos: O objetivo do DAO é separar a camada lógica dos recursos, fornecendo um repositório de objetos onde a fonte dos dados é transparente para a regra de negócio. A aplicação delega para o objeto DAO a função de comunicar com o banco de dados. Mudanças na comunicação com o banco de dados afetam somente a camada das classes DAO, sem interferir no funcionamento da regra de negócios.

Diagrama:

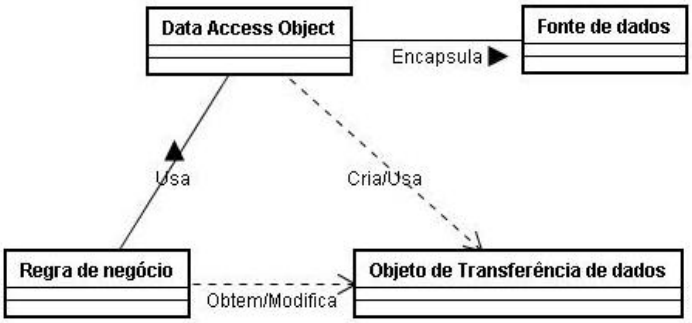

Figura 6 Diagrama de classe do Padrão DAO

2.5.3 Singleton

Classificação: O padrão de projeto *Singleton* pertence ao conjunto de padrões de criação.

Intenção: Garantir a criação de uma única instância de classe acessível de maneira global.

Objetivos: Muitas vezes um recurso global precisa ser acessado por diversas partes do código. Para retirar a responsabilidade de criação destes recursos pelos clientes, e garantir a unicidade, pode-se controlar essas características por meio do padrão *Singleton*, que retorna sempre ao usuário a mesma instância do recurso desejado.

Diagrama:

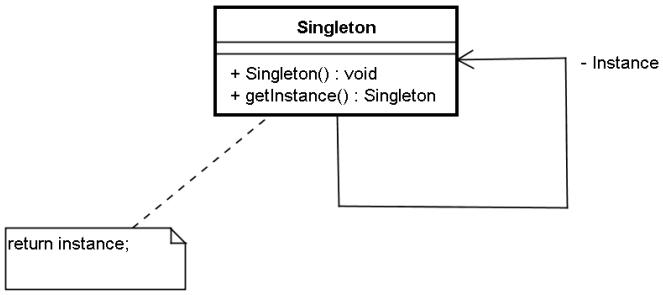

Figura 7 Diagrama de classe do padrão *Singleton*

2.5.4 Factory Method

Classificação: O padrão de projeto *Factory Method* pertence ao conjunto de padrões de criação.

Intenção: Ele fornece um mecanismo para criar objetos a partir de um método sem que seja necessário especificar a classe concreta.

Objetivos: O padrão *Factory Method* proporciona maior abstração, por não trabalhar diretamente com a classe que deseja instanciar. Pode atuar como uma fábrica de objetos genérica, recebendo um parâmetro indicando o tipo de objeto que o cliente deseja e criando de maneira transparente o objeto.

Diagrama:

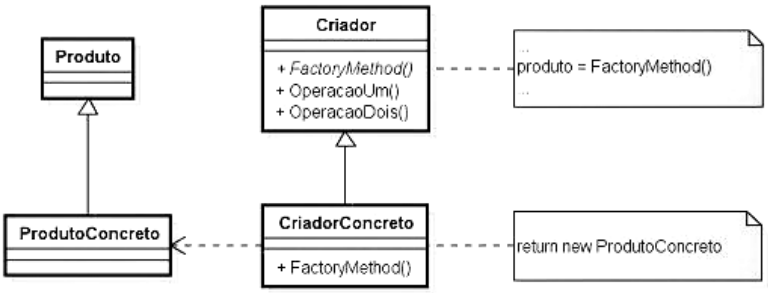

Figura 8 Diagrama de classe do padrão *Factory Method*

2.6 Tecnologias e Ferramentas de Desenvolvimento Web

2.6.1 JavaEE

A plataforma Java Enterprise[1] é um padrão para indústrias de computação que possibilita criar aplicações web empresariais. Ela é composta pelos conjuntos de tecnologias: *Web Services Technologies*, *Web Application Technologies*, *Enterprise Application Technologies* e

[1] http://www.oracle.com/technetwork/java/javaee/overview/index.html. Acessado em 16/10/2011

Management and Security Technologies. O conjunto de tecnologias *Web Application* é composto pelas especificações Java Servlet, JavaServer Faces, JavaServer Pages e Standard Tag Library for JavaServer Pages. Este conjunto de tecnologias dão suporte para a criação de aplicações web utilizando a plataforma Java.

2.6.2 VRaptor

O VRaptor[1] é um f*ramework* MVC Java para aplicações Web focado em desenvolvimento rápido. Ele é simples e intuitivo e possibilita maior produtividade durante o desempenho, tendo uma curva de aprendizado curta. O VRaptor é um framework baseado em outros frameworks, como o Spring, Hibernate e outros, e a sua principal característica é a de utilizar convenções, ao invés de configurações. Por isso, a quantidade de código e os arquivos de configurações são reduzidos, e o desenvolvedor foca na solução.

2.6.3 Tomcat

O Tomcat[2] é um servidor de aplicações, mais especificamente um container de *servlets, e* é uma implementação de software open source para as tecnologias Java Servlet e do JSP do JavaEE. O Tomcar não implementa as especificações de EJB, por isso não é considerado um servidor de aplicações.

2.6.4 MySQL

---

[1] Fonte: http://vraptor.caelum.com.br. Acessado em 16/10/2011.
[2] Fonte: http://tomcat.apache.org/index.html. Acessado em 16/10/2011.

O MySQL[1] é o Sistema Gerenciador de Banco de Dados (SGBD) gratuito de código aberto mais popular no mundo. Possui também uma versão comercial com diversos recursos, e possui diversas ferramentas para modelagem, consultas, e backup dentre outros. É muito utilizado em web sites, e diversas ferramentas de gestão do conhecimento, gerenciamento de conteúdos e ambiente virtual de aprendizagem utilizam ou dão suporte a este SGBD.

2.6.5 Hibernate

O Hibernate[2] é um conjunto de projetos relacionados que permite que os desenvolvedores utilizem modelos estilo POJO em suas aplicações, realizando mapeamento de classes Java para tabelas de banco de dados e permitindo a manipulação diretamente por meio de objetos. Com isso, ele facilita a criação de consultas mais simples e de fácil modificação, e reduz consideravelmente o volume de código necessário.

2.6.6 AJAX

O AJAX (Asynchronous JavaScript e XML) é uma técnica para criação rápido de páginas web de forma dinâmica. AJAX permite que páginas web para ser atualizado de forma assíncrona através da troca de pequenas quantidades de dados com o servidor. Isto significa que é possível atualizar partes de uma página web, sem recarregar a página inteira.

Páginas web clássicas, (que não usam AJAX) tem que recarregar a página inteira se algum conteúdo mudar.

---

[1] http://www.mysql.com/products. Acessado em 16/10/2011.
[2] http://www.hibernate.org/hibernate. Acessado em 16/10/2011.

3. METODOLOGIA

3.1 Classificação da Pesquisa

Neste trabalho realizou-se uma pesquisa aplicada, envolvendo a analise de fatores de qualidade em um sistema real.

Quanto ao objetivo, a pesquisa é classificada como descritiva, buscando avaliar os resultados da aplicação de padrões de projetos em um software real.

Para atingir os objetivos foi adotada a abordagem quantitativa, fundamentou-se em métricas de software voltadas à melhoria da manutenibilidade. Os procedimentos metodológicos adotados são detalhados na seção seguinte.

3.2 Procedimentos metodológicos

Para consecução dos objetivos descritos neste trabalho, foi realizado um estudo de caso com um software comercial desenvolvido em Java sem a utilização de padrões de projeto, que está em fase de reestruturação.

Para realizar a avaliação da manutenibilidade do software, pretende-se utilizar modelos de avaliação de qualidade de software que fornecem um processo analítico capaz de avaliar as características de manutenibilidade, antes e após sua reestruturação.

A proposta dessa metodologia é analisar isoladamente a qualidade do código, ao refatorar uma classe, e posteriormente analisar o projeto como um todo, analisando a coesão e o acoplamento. O procedimento consiste em seis etapas descritas a seguir e resumidas na Figura 9:

1. Identificação do sistema a ser estudado: Etapa em que a arquitetura, e características funcionalidades do sistema são levantadas e caracterizadas;
2. Identificação das deficiências do sistema: Etapa em que as características de arquitetura e implementação serão

avaliadas com base nos critérios de qualidade estudados e levantar as deficiências que podem ser trabalhadas;

3. Definição dos padrões de projeto a serem aplicados: Etapa em que cada deficiência será analisa a fim de buscar melhorar a implementação com a adoção de boas práticas de programação e utilização de um ou mais padrões de projeto;
4. Reestruturação do sistema a partir do sistema original: Etapa em que as a reestruturação é realizada a partir das propostas de melhoria e preservando as funcionalidades do sistema original. Realiza-se a criação de um novo sistema adotando aproveitando a estrutura do banco de dados e então se transcrevem os códigos antigos pouco a pouco para o novo projeto, realizando refatoração e aplicando de padrões de projeto.
5. Definição das métricas com base nos fatores de qualidade desejáveis: Nesta etapa serão definidas as métricas que avaliam as características de qualidade que foram abordadas durante a reestruturação;
6. Análise comparativa das medidas obtidas: Nesta etapa, a partir da coleta dos dados por meio de ferramentas e *plugin*, analisar as medidas obtidas de modo que possibilite a comparação do sistema original e da versão reestruturada e apresentação dos dados de forma clara.

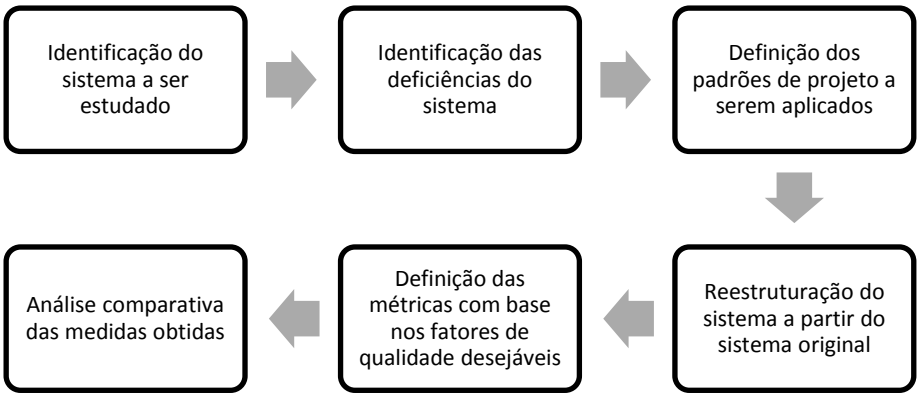

Figura 9 Diagrama da metodologia definida

Para realizar a análise do resultado da aplicação dos padrões de projeto após a reestruturação do software, foram estudadas métricas de software relacionadas à manutenibilidade a fim de se obter dados referentes a um sistema. As mesmas métricas foram analisadas no sistema original e no sistema reestruturado.

Para avaliar o impacto da adoção de padrões de projeto, realizou-se uma comparação dos dados coletados avaliando quantitativamente se houveram melhorias na manutenibilidade do software reestruturado.

4. RESULTADOS

4.1 Identificação do sistema a ser estudado

O SISTEMA é uma aplicação web desenvolvida em Java, utilizando a plataforma Java EE Web Application, e utiliza um banco de dados MySQL. O sistema é executado em um container de aplicações web, Tomcat.

O SISTEMA tem como finalidade possibilitar o gerenciamento e apoio às atividades de um Instituto de Pesquisa, desde o gerenciamento dos dados da equipe e clientes, até o acompanhamento da execução de projetos. Para o estudo, foram selecionadas partes do sistema para realizar a reestruturação. Estas partes são responsáveis por informatizar as seguintes atividades:

- Gerenciamento dos dados do Instituto de Pesquisa e dos usuários do sistema;
- Gerenciamento dos clientes do Instituto de Pesquisa;
- Gerenciamento das pesquisas a serem realizadas;

O SISTEMA foi sendo desenvolvido ao longo do tempo, e suas funcionalidades foram sofrendo modificações. A arquitetura do sistema não é padronizada e tecnologias novas foram sendo adicionadas durante o desenvolvimento. As principais características e tecnologias são apresentadas na Tabela 2.

Tabela 2 Descrição das características do sistema estudado

Características	Sistema original
Tecnologia	Java EE Web Application
Framework MVC	VRaptor (utilizado parcialmente)
Framework ORM	Hibernate (utilizado parcialmente)
Recurso Ajax	Transferência de conteúdo HTML
Padrões de Projeto	DAO (utiliza JDBC e JPA/Hibernate) MVC (camadas não são totalmente separadas)

A estrutura do código do sistema sofreu uma adaptação ao incorporar o *framework* MVC VRaptor. Na Figura 10 é apresentada a organização de pacotes do sistema.

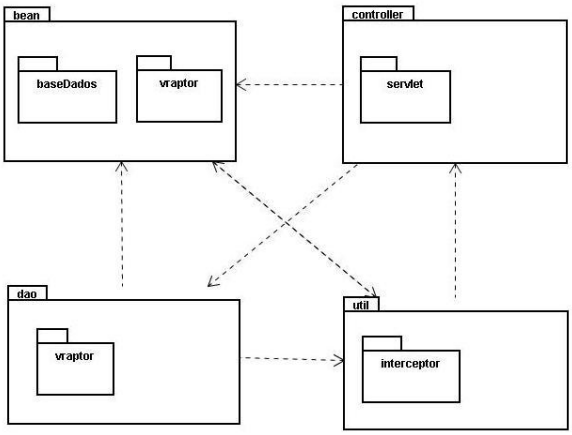

Figura 10 Diagrama de pacotes do SISTEMA original

4.2 Identificação das deficiências do sistema

Após a definição e caracterização do software a ser estudado foi realizado uma análise para identificar falhas, deficiências e possibilidades de melhorias, com base em boas práticas de programação e com a adoção de padrões de projeto.

A seguir são listados os pontos que serão trabalhados visando melhorar a manutenibilidade do software.

4.2.1 Acoplamento entre View e Controller

Foi identificado no sistema uma deficiência da implementação do padrão MVC, onde o elementos de visualização (View) estavam implementados juntamente com classes de controle (Controller). As requisições feitas em AJAX recebem como resposta um conteúdo HTM já formatado, e não os dados. Isso dificulta a manutenção, pois uma mudança na apresentação dos dados deve ser realizada nas camadas de visualização e de controle. Abaixo segue um trecho de código em que este problema foi identificado.

```
(...)
if (retorno.equals("LISTA_USUARIOS")) {
  UsuarioDAO buscarUsuario = new UsuarioDAO();
  UsuarioBean usuarios[] = buscarUsuario.listarUsuario(ip);
  if (usuario != null) {
          String a = "<table id=\"tabelaUsuario\"
border=\"1\" width=\"90%\">";
          for (int i = 0; i < usuarios.length; i++) {
                  a = a
                          + "<tr><td id=\""
                          + usuarios[i].getIdUsuario()
                          + "\" align=\"left\"
style=\"cursor:pointer;\" onClick=\"alterarCor("
                          + usuarios[i].getIdUsuario() +
");altFrame("+usuarios[i].getIdUsuario()+");\">"
                          + usuarios[i].getNome() +
"</td></tr>";
                  }
          a = a + "</table>";
          out.println("document.getElementById(\"" + target
                          + "\").innerHTML='" + a + "';");

  out.print("document.getElementById('selIntAb').style.visib
ility= 'visible';");
  } else{

  out.print("document.getElementById('selIntAb').style.visib
ility= 'hidden';");
          out.println("document.getElementById(\"" + target
                          + "\").innerHTML='';");
          }
}
(...)
```

4.2.2 Modificação da arquitetura durante o desenvolvimento

A adoção do framework MVC VRaptor durante o desenvolvimento do sistema modificou o padrão do código das classes da camada *Controller*, antes implementado utilizando *Servlets*.

Cada *Controller Servlet* é responsável por uma entidade e um parâmetro passado pela requisição define qual ação será executada. O *Controller* possui uma estrutura condicional que define qual bloco de código será executado, dependendo da ação. Toda a lógica do *Controller* fica dentro do mesmo método.

Cada *Controller* usando o VRaptor também é responsável por uma entidade. Entretanto, cada método do *Controller* fica responsável por uma ação que possui um URI diferente, evitando assim uma estrutura condicional, e valorizando a legibilidade do código. Um exemplo comparativo é a Figura 11, que mostra duas classes *Controller*, a primeira escrita utilizando *Servlet*, e a segunda utilizando o framework VRaptor.

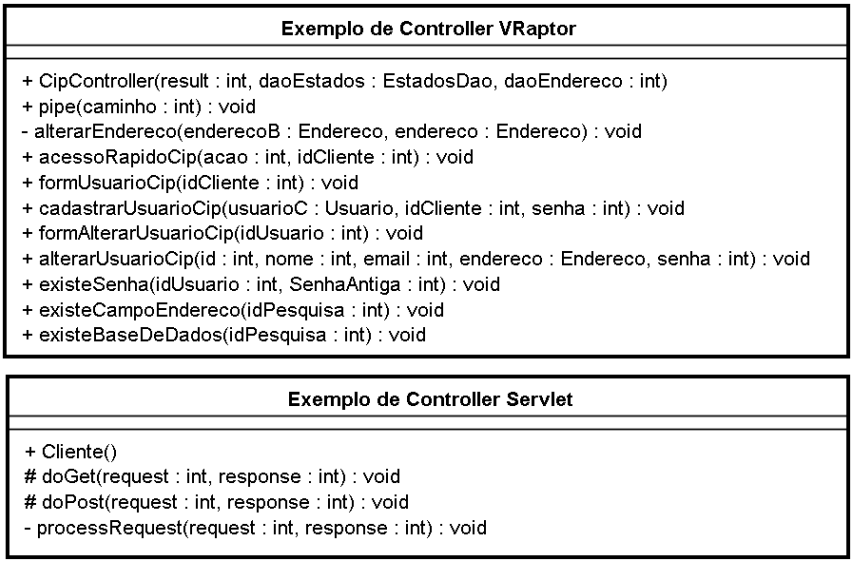

Figura 11 Comparação entre classes controles do sistema original

4.2.3 Duplicação de código

Havia a duplicação de código em várias situações. As entidades *Usuario* e *Cliente*, por exemplo, possuem endereço. Existe uma duplicação do código responsável por alterar o endereço de uma entidade, o que dificulta a manutenção, pois uma alteração na lógica precisa ser feita duas vezes. Este exemplo pode ser observado com o trecho de código a seguir:

```
//Controller Cliente
(...)
 EnderecoBean endereco = new EnderecoBean(idEndereco,
                                          logradouro, numero,
complemento1, complemento2, bairro,
                                          CEP, cidade, UF,
telResidencial, telCelular,
                                          telComercial, fax);
 EnderecoDAO cadastrarEndereco = new EnderecoDAO();
 if (cadastrarEndereco.alterarEndereco(endereco)){
        if (cadastrarCliente.alterarCliente(cliente)){
               (...)
        }
 }
(...)

//Controller Usuario
(...)
 EnderecoBean endereco = new EnderecoBean(idEndereco,
                                          logradouro, numero,
complemento1, complemento2, bairro,
                                          CEP, cidade, UF,
telResidencial, telCelular,
                                          telComercial, fax);
 EnderecoDAO cadastrarEndereco = new EnderecoDAO();

 if (cadastrarEndereco.alterarEndereco(endereco)){
        if (cadastrarUsuario.alterarUsuario(usuarios)){
               (...)
        }
 }
(...)
```

4.2.4 Classes DAO sem padrão

As classes relacionadas às entidades foram implementadas utilizando padrão de projeto DAO. Porém, as classes foram implementados utilizando duas tecnologias, antes e depois da adoção do framework VRaptor. Isso fez com que parte do código realizasse consultas por meio de scripts SQL, e outra parte por meio da API JPA/Hibernate. Isso prejudica o reaproveitamento de

código, dificulta a legibilidade do projeto e contribui para a falta de padronização, influenciando na manutenibilidade. A seguir, podem ser vistos dois trechos de códigos que acessam a mesma tabela do banco de dados, mas possuem implementações distintas, utilizando os métodos anteriormente descritos:

```
//DAO Projeto - JDBC
(...)

public ProjetoBean[] listarProjetos(String idCliente){
        Connection conn = null; Statement stmt = null;
        ResultSet rs = null;
        FormataDataBD fdata = new FormataDataBD();
        ProjetoBean[] a = null;
        try{
                String query = "select * from pesquisa where
id_cliente in ("+ idCliente + ") and ativo = 1";
                conn = (Connection) Conexao.getConn();
                stmt = (Statement) conn.createStatement();
                rs = stmt.executeQuery(query);
                int cont = 0;
                while (rs.next()) {
                        cont++;
                }
                if (cont > 0) {
                        ProjetoBean projetos[] = new
ProjetoBean[cont];
                        rs.beforeFirst();
                        int i = 0;
                        while (rs.next()) {
                                ProjetoBean projetoB = new
ProjetoBean(rs.getString("id"), rs.getString("descricao"),
rs.getString("id_cliente"),rs.getString("id_tipo_geometria"),

        rs.getDate("data_inicio"),  rs.getDate("data_fim"),
rs.getString("num_entrevistas"),

        rs.getString("num_entrevistadores"), rs.getString("local"),
rs.getString("status"),rs.getString("nome_base"),
rs.getLong("tam_base"));
                                projetos[i] = projetoB;
                                i++;
                        }
                        return projetos;
                }else{return null;}
        }catch(Exception e){e.printStackTrace();return null;}
}
(...)
```

```
//DAO Projeto - Hibernate
(...)
public List<Pesquisa> getPesquisas(Cliente cliente, int visivel) {
        int idCliente = cliente.getId();
        Criteria criteria =
        getSession().createCriteria(Pesquisa.class);
        criteria.add(Restrictions.eq("visivel", visivel));
        criteria.add(Restrictions.eq("ativo", 1));
        Criteria critCliente = criteria.createCriteria("cliente");
        Criterion clie = Restrictions.eq("id", cliente.getId());
        Criterion id_cliente_pai = Restrictions.eq("idClientePai",
idCliente);
        LogicalExpression le = Restrictions.or(clie,
id_cliente_pai);
        critCliente.add(le);

        return critCliente.list();
}
(...)
```

4.2.5 Falta de Herança

Por se tratar de um software desenvolvido em uma linguagem orientada a objetos, verificou-se um baixo nível de abstração e pouca utilização de herança. Devido aos diversos tipos de usuários que podem utilizar o sistema, o sistema utiliza um atributo da entidade *Usuario* para diferenciar os tipos de usuário. Por ser um software implementado com o paradigma orientado a objetos, deveriam ser implementadas classes diferentes para cada tipo de usuário, que herdassem a classe *Usuario*. Assim, poderia utilizar o polimorfismo ao invés de comparar o atributo em diversas partes do software para diferenciar a entidade *Usuario*. Além de diferenciar os diversos tipos de usuários pelo tipo de instância, também é possível melhorar o reaproveitamento de código. Na Figura 12 é possível ver o diagrama de classe da entidade *Usuario*, onde pode ser observado o atributo *idGrupoUsuario*, que é utilizado para diferenciar os usuários.

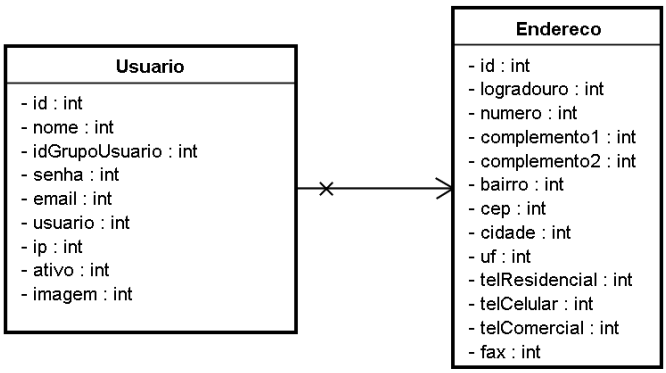

Figura 12 Diagrama de classe da entidade Usuario e Endereco

4.3 Definição dos padrões de projeto

Dentre os padrões de projeto definidos por GAMMA *et al*. (1995) e CRAWFORD E KAPLAN (2003), foram selecionados os padrões que podem ser aplicados no sistema durante a reestruturação com base nas deficiências do software apresentadas:

- MVC: o padrão de arquitetura MVC foi utilizado adotando integralmente o framework MVC, pois é responsável por gerenciar a camada de controle de forma mais eficiente, utilizando convenções de implementação;
- *Singleton*: Foi adotado para implementar uma solução ao problema de duplicação de código, e delegar a responsabilidade de criação de objetos utilizados globalmente a uma única classe;
- *Factory*: Foi escolhido para encapsular o processo de criação de vários objetos com características semelhantes. Assim é possível especificar o tipo de objeto mais específico que será instanciado, em tempo de execução;
- DAO: O padrão de projeto DAO, anteriormente adotado, foi mantido e padronizado, visando encapsular o acesso aos

dados do sistema do restante do código, tornando o banco de dados transparente para as outras camadas.

4.4 Reestruturação do sistema com base no sistema original

Com base nos padrões de projeto selecionados, e utilizando boas práticas de programação, várias modificações foram realizadas no sistema. A princípio a base de dados foi duplicada, a fim de preservar os dados existentes, e um novo projeto foi criado do zero. As funcionalidades começaram a ser implementadas uma a uma, criando novas classes e reaproveitando trechos de códigos existentes, e adaptando às necessidades de melhoria. Cada funcionalidade do sistema original foi reescrita gerando um sistema reestruturado chamado que denominaremos de SISTEMA versão2. Após a implementação de todas as funcionalidades, ocorreram diversas refatorações para se chegar à versão final.

Todo o código da interface foi movido para a camada *View*, e a comunicação com *Controller* com a *View* é feita transmitindo dados, e não mais conteúdo HTML;

A plataforma Java EE Web Application foi mantida, e o framework VRaptor, utilizado parcialmente no sistema, foi adotado para auxiliar a implementar o padrão MVC, criando a camada *Controller* padronizada, responsável por receber as requisições e processar as requisições vindas da interface.

As classes *EstadosController*, *EnderecoController* e *LoggerUtil* foram criadas utilizando o padrão de projeto *Singleton*. O motivo da adoção deste padrão foi para isolar códigos que estavam duplicados ou eram variáveis globais. Com a adoção do *Singleton*, é possível garantir uma única instância de objetos que podem ser acessados em qualquer parte do código que precise de seus recursos, conforme é ilustrado na Figura 13. Visando manter a estrutura de dados original, as classes do modelo que representam

as entidades foram preservadas e as classes DAO sofreram mudanças para adotar o padrão utilizado com o framework Hibernate.

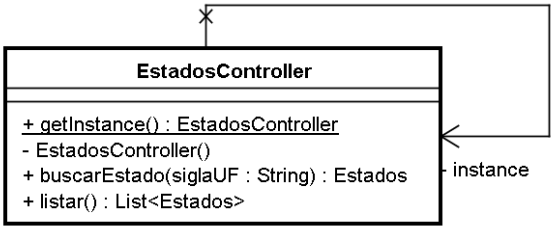

Figura 13 Diagrama de classe de uma implementação do padrão de projeto Singleton

Visando melhorar a abstração do sistema, foram criadas classes específicas, uma classe para cada tipo de usuário, que herdam a classe *Usuario*. Com isso, criou-se a classe *UsuarioFactory*, utilizando o padrão de projeto *Factory Method*, responsável por criar os objetos específicos com base em um argumento passado, usuário, contendo dados genéricos da entidade *Usuario* e também um código que define qual tipo específico deve ser instanciado. Com isso, os detalhes da criação dos objetos ficam encapsulados, e remove do restante do código a responsabilidade de instanciar diferentes tipos de usuários, como pode ser visto na Figura 14.

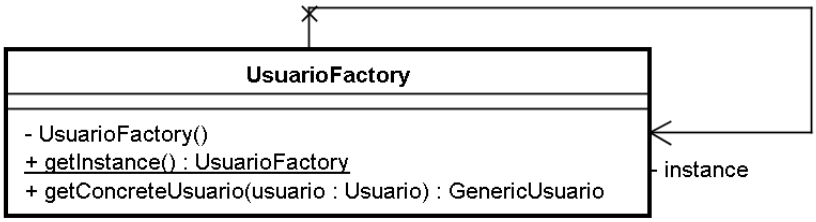

Figura 14 Diagrama de classe da implementação do padrão de projeto *Factory Method*

Após a reestruturação, o SISTEMA versão2 passou a conter as seguintes funcionalidades:

- Geral: Estrutura básica do sistema, contendo menus, cabeçalho, rodapé, e funções presentes em todos os módulos, como alertas.
- Módulo Instituto de Pesquisa (IP): Gerencia os institutos de pesquisa. Cada instituto tem seus próprios clientes, usuários, pesquisas. É possível inserir todos os dados do IP, como endereço, telefone, CNPJ, e será disponível para os usuários.
- Módulo Meus Usuários: Gerencia os usuários do sistema. Os usuários podem ser usuários do Instituto de Pesquisa (IP) ou usuários dos Clientes do Instituto de Pesquisa (CIP).
- Módulo Meus Clientes: Gerencia os clientes do IP. Para cada cliente, são cadastrados dados do cliente, como contato, email, telefone, e os usuários do cliente que terão acesso ao sistema SISTEMA.
- Módulo Meus Dados: Exibe os dados do IP (para usuário IP), do CIP (para usuários CIP), dados do entrevistador ou entrevistado.
- Módulo Minhas Pesquisas: Gerencia as pesquisas do IP, que podem estar associadas ao próprio IP (pesquisa interna), ou associada a qualquer cliente. Definir o estado da pesquisa (disponível para acompanhamento, inscrição ou pesquisa web).

As características do sistema após a reestruturação tornaram-se mais homogêneas, privilegiando a utilização de frameworks, implementando a arquitetura MVC e adotando padrões de projeto de forma correta, conforme apresentado na Tabela 3. A organização dos pacotes permanece utilizando a abordagem de camadas, separando os dados, a camada de acesso aos dados, e a camada de controle, conforme visto na Figura 15.

Tabela 3 Características do Sistema Reestruturado

Características	Sistema Reestruturado
Tecnologia	Java EE Web Application
Framework MVC	VRaptor
Framework ORM	Hibernate
Recurso Ajax	Dados(JSON)
Padrões de Projeto	DAO, Factory, Singleton, MVC

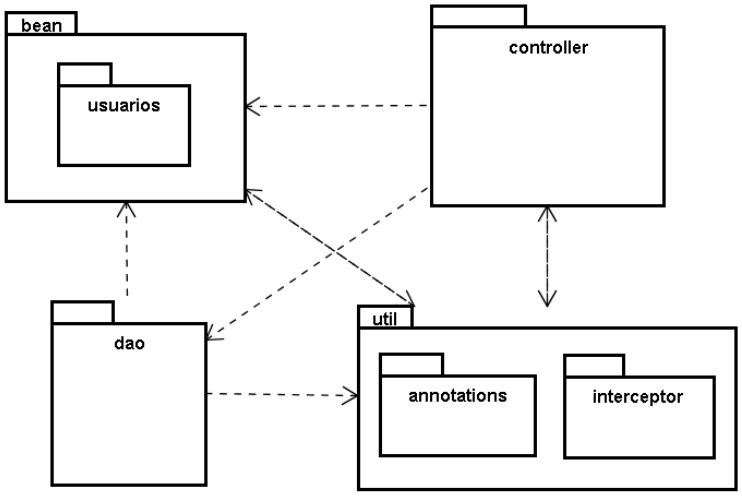

Figura 15 Diagrama de pacotes do SISTEMA versão2.

4.5 Definição das métricas.

A partir das métricas e modelos de avaliação estudados, buscou-se selecionar as métricas que pudessem auxiliar a quantificar o nível de manutenibilidade de um software desenvolvido em Java. Foram selecionadas métricas de quatro grupos: Complexidade, Acoplamento, Manutenibilidade e Tamanho. As métricas selecionadas são apresentadas na Figura 16.

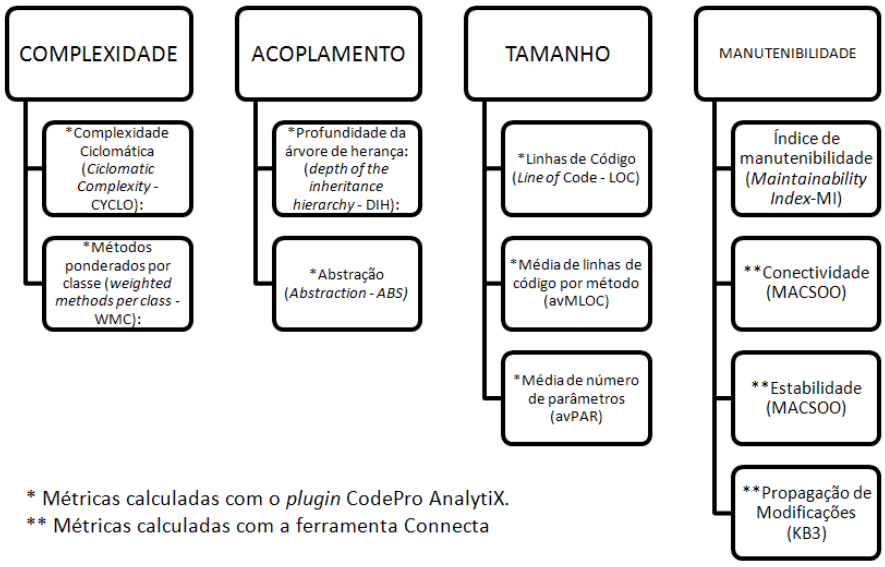

Figura 16 Métricas utilizadas no processo de analise da manutenibilidade

Para coletar as medidas, foram utilizadas as ferramentas CodePro AnalytiX[1] para o IDE Eclipse[2],e a ferramenta *Connecta*[3], conforme é identificado na Figura 16.

4.6 Analise dos resultados obtidos

As duas versões do sistema passaram pelo mesmo tipo de analise utilizando as ferramentas CodePro Analityx e *Connecta*, que forneceram um conjunto de medidas que representam diversos as aspéctos de qualidade. Na tabela Tabela 4 são apresentadas as métricas utilizadas, a característica da

[1] Plugin para a IDE Eclipse que permite coletar métricas de código-fonte de softwares desenvolvidos em Java. http://code.google.com/intl/pt-BR/javadevtools/codepro/doc/index.html (acessado em 06/06/2011).
[2] Ambiente de desenvolvimento integrado desenvolvido pela Eclipse Foundation e suporta diversas linguagens de programação. http://www.eclipse.org/ (acessado em 06/06/2011).
[3] Ferramenta desenvolvida a partir do modelo MACSOO, utilizado para avaliar a conectividade de sistemas desenvolvidos com a linguagem Java. FERREIRA (2011).

métrica, os valores do Sistema original (SISTEMA) e do Sistema versão2 (SISTEMA 2), e a variação entre as duas medidas em porcentagem.

Tabela 4 Resultado geral das medidas coletadas[1]

Métricas	Característica	SISTEMA	SISTEMA 2	Variação
Nº classes	⇔	60	50	-17%
avCYCLO	⇩	1,74	1,36	-22%
WMC	⇩	1180	644	-45%
avDIH	⇔	2,53	2,29	-9%
ABS(%)	⇧	0	6	+6%
LOC	⇩	7195	3446	-52%
avMLOC	⇩	9,21	5,76	-37%
avPAR	⇩	0,89	0,75	-16%
MI	⇧	-38,213	-33,182	+13%
MACSOO (Estabilidade)	⇩	1,188	1,745	+47%
MACSOO (Conectividade)	⇩	0,033	0,041	+24%
K3B	⇩	1,8738	1,7027	-9%

Os resultados apresentam melhorias de forma geral em todas as métricas analisadas. As métricas de complexidade indicam que a reestruturação possibilitou reduzir em 22% a complexidade dos métodos, o que também pode ser verificado pelo número médio de linhas por método com uma redução de 37%. Além disso, o tamanho total do sistema em linhas de código também foi reduzido em 52%. O Nível de abstração do SISTEMA era 0%, e após a reestruturação possui 6%. Isso indica um melhor uso da orientação a objetos, o que também é indicado pela profundidade de hierarquia de herança (DIH), que teve um aumento de 9%.

[1] ⇧ = quanto maior valor melhor; ⇩ = quanto menor melhor; ⇔ = valor de referencia, maior ou menor não tem significado.

Os resultados da avaliação proposta pelo modelo MACSOO avalia a manutenibilidade do sistema, e quanto menor o valor, menor o esforço em realizar modificações. O valor de estabilidade calculado pelo modelo MACSOO obteve um aumento de 47%, e a métrica de conectividade um aumento de 24%. Além disso, a métrica K3B, que avalia a propagação de modificações ao se modificar uma determinada classe reduziu 9%. Embora os valores de estabilidade e conectividade tenham aumentado, indicando que o SISTEMA versão2 está mais acoplado e que uma modificação em uma classe afeta uma parte maior do sistema, temos que levar em conta que a conectividade ainda é baixa, e é resultante da interação entre as partes do sistema.

Para melhor interpretação dos dados relacionados à Estabilidade e Conectividade, foram buscados na literatura dados coletados de aplicações para comparar com o sistema estudado. Na Tabela 5 é apresentado os valores da Estabilidade em porcentagem, que indica o impacto da manutenção em relação ao total de classes do sistema. Também é exibido o grau de Conectividade, que se refere à razão entre o número de conexões existentes e o maior numero de conexões possíveis. Também há um conceito dado por um especialista em relação à qualidade do aplicativo, e que não tem influência alguma das medidas.

Ao comparar os dados obtidos com os encontrados na literatura, vemos que mesmo a conectividade e a estabilidade aumentando, os valores ainda são baixos, e similares a aplicações consideradas com valores bons. Uma possível explicação é que a conectividade é presente em sistemas que possuem muitas ligações, e utilizam muitos recursos externos. Na OO normalmente é desejado que houvesse uma divisão de responsabilidades, o que faz com que existam ligações entre diversas partes do sistema. Um alto número de ligações é ruim, mas algumas são necessárias.

O sistema possuía classes muito coesas, e pouco acopladas, entretanto possuía uma duplicação de código muito grande, e pouca divisão de responsabilidades.

Tabela 5 Comparação com as medidas encontradas por FERREIRA *et al.*(2008)

Aplicação	Nº classes	Estabilidade (%)	Conectividade (%)	Conceito
SISTEMA	60	1,967	1,188	-
SISTEMA 2	50	3,490	1,745	-
1	64	4,2	3	Ótimo
4	29	7,2	6	Excelente
6	14	9,3	7	Bom
9	14	19,3	16	Razoável
11	5	34	20	Fraco

Além da análise global, também foram realizados cálculos por pacote, analisando separadamente as classes de cada pacote, a fim de verificar quais as medidas de cada grupo de classes com características em comum. Na Tabela 6 podemos observar os dados coletados, das duas versões do SISTEMA. Os dados mostram que o pacote *Controller* possuía a maior complexidade Ciclomática, 4,29 em média. Após a reestruturação, obteve-se o valor médio de 1,92. Isso indica que a proposta de trabalhar com o framework MVC, padronizando as classes *Controller* e separando lógicas complexas em vários métodos obteve bons resultados. A métrica WMC também mostra que a complexidade dos métodos também foi reduzida em todos os pacotes.

O MI também pode ser melhor analisado ao observar por pacotes. O MI das classes do pacote *Controler* do SISTEMA possuem o valor -19,51. Após a reestruturação, das classes do pacote *Controler* passaram a possuir o valor 0,76. O valor ainda é baixo, comparando-se com outros pacotes, entretanto classes que possuem regras de negócios tendem ser maiores e mais complexas, o que compromete a manutenibilidade.

Tabela 6 Resultado das métricas por pacote das duas versões do SISTEMA[1]

Métricas	**Caracte-rística**	Bean		Controller		Dao		Util	
		S1	S2	S1	S2	S1	S2	S1	S2
avCYCLO	⇩	1,03	1,13	**4,29**	**1,92**	2,18	1,29	2,32	1,83
WMC	⇩	463	345	**429**	**187**	118	44	100	68
avDIH	⇔	2,04	2,23	3,41	2,09	2,50	2,88	2,12	2,11
ABS(%)	⇧	**0,00**	**9,50**	0,00	0,00	0,00	0,00	0,00	11,10
LOC	⇔	2206	1585	2871	1217	**1637**	**308**	481	336
avMLOC	⇩	4,00	4,16	**25,57**	**10,43**	**17,37**	**6,47**	9,25	6,12
avPAR	⇩	0,53	0,49	1,98	1,34	1,44	1,36	1,10	0,81
MI	⇧	-10,50	-3,43	**-19,51**	**0,76**	**-6,05**	**31,66**	21,96	31,19

Na Figura 17 podemos observar um gráfico que compara o SISTEMA versão2 com o SISTEMA original. As medidas possuem valor 1 (um), ou 100%, enquanto as medidas da versão2 é a razão entre a medida do o original e a versão2. Sendo assim, valores menores que 1 (um) representam diminuição, e valores maiores que 1 (um) representam aumento.

[1] S = Sistema; Sv2 = Sistema versão 2; ⇧ = quanto maior valor melhor; ⇩ = quanto menor melhor; ⇔ = valor de referencia, maior ou menor não tem significado.

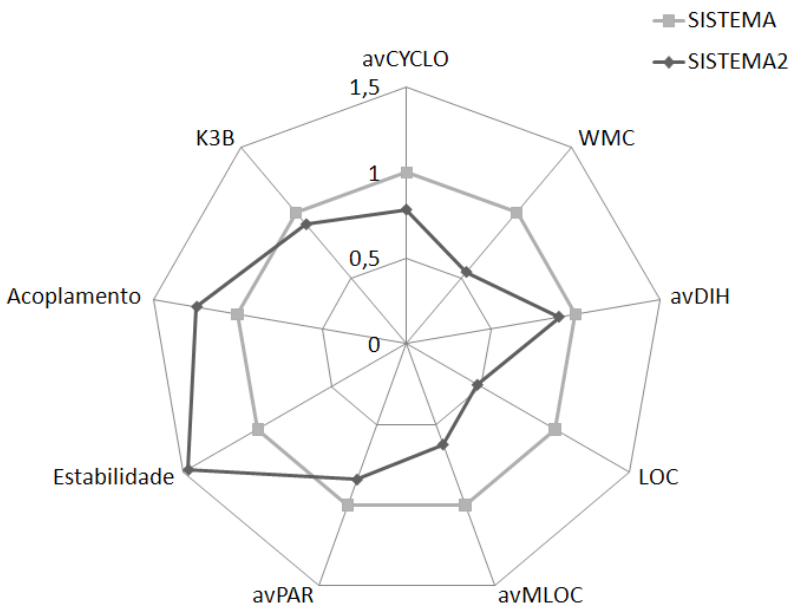

Figura 17 Proporção entre as medidas do SISTEMA e do SISTEMA versão2

Podemos observar que as únicas métricas que apresentaram valores maiores foram a estabilidade e a conectividade, obtidas pelo modelo MACSOO. O aumento deste valor indica melhoria da manutenibilidade, embora tenham aumentado, ainda podem ser considerados valores bons, e estão na faixa de valores de aplicações definidas como boas, segundo FERREIRA(2008). As outras métricas obtiveram diminuição no SISTEMA versão2, e em todos estes casos, a diminuição era desejada.

4.7 Dificuldades encontradas

Durante o trabalho de reestruturação, coleta e analise dos dados, foram necessários alguns estudos para conseguir chegar ao resultado apresentado.

O trabalho de reestruturação utilizou dos recursos foltados para refatoração da ferramenta de desenvolvimento Eclipse, que auxilia a renomear variáveis, métodos e classes, acrescentar ou remover parâmetros, mover classes e extrair trechos de código para um novo método. Diversos recursos foram utilizados para analizar códigos com baixa qualidade, como o pluguin CodePro Analytix que fornece, além das métricas, uma busca por códigos códigos duplicados, e exibe um gráfico de conexões de todo o sistema ou de um pacote específico.

A ferramenta para calcular a Conectividade e a propagação de modificações (K3B), chamada Connecta, foi adquirida com Kecia Aline M. Ferreira que explicou o seu funcionamento. A ferramenta foi desenvolvida por FERREIRA et. al(2008) para validação do modelo MACSOO porposto no trabalho. A ferramenta possui uma interface muito fraca, e ainda não está totalmente finalizada.

Foi encontrada uma certa dificuldade de utilizar a ferramenta, pois necessita de algumas bibliotecas, e deve ser executada por linha de comando, para então carregar a interface gráfica. Diferentemente do plugin CodePro Analytix que analisa o código-fonte, a ferramenta Connecta analiza os bytecodes, que são os códigos Java compilados. Isso dificulta a analise, pois deve-se compilar o projeto todo, e então utilizar a ferramenta fora do ambiente de programação. Essa dificuldade pode atrapalhar sua utilização no dia a dia do desenvolvimento, por demandar de muito tempo para conseguir obter as medidas.

5. CONCLUSÃO

Para avaliar impacto da utilização de padrões de projetos na manutenibilidade de um software comercial em Java, foram aplicados padrões de projeto em um sistema comercial desenvolvido em Java e mediu-se o nível de manutenibilidade do sistema original e do sistema reestruturado, comparando-os.

Após a análise dos dados coletados percebe-se que a maioria das medidas do sistema reestruturado superaram as medidas do sistema original. A partir desta analise, pode-se concluir que a reestruturação do sistema proporcionou notáveis melhorias na qualidade do software estudado. Considerando que as métricas de software aplicadas podem ser usadas para mensurar o grau de esforço para realizar a manutenção em um sistema, pode-se afirmar que o sistema reestruturado possui melhor manutenibilidade, isto é, exige menor esforço para sua modificação.

5.1 Trabalhos Futuros

Como trabalhos futuros, sugere-se pesquisar a relação de cada padrão e seu impacto na manutenibilidade, definindo em quais situações eles devem ser empregados, analisando quais estão diretamente relacionados com a melhoria da manutenibilidade.

Pretende-se também realizar simulações de manutenção no software, a partir de alteração nos requisitos e estabelecimento de novos, que deverão ser implementados no sistema original e no reestruturado. A partir dessa alteração, pode-se avaliar o esforço para tal manutenção e comparar os resultados da aplicação das métricas, visando comprovar se as métricas de software são bons indicativos para avaliar a melhoria da manutenibilidade.

REFERÊNCIAS BIBLIOGRÁFICAS

ALMEIDA, L. T.; DE MIRANDA, J. M**. Código Limpo e seu Mapeamento para Métricas de Código Fonte**, 2010. Monografia (Graduação em Ciência da Computação) – Universidade de São Paulo, São Paulo, 2010. http://www.ime.usp.br/~cef/mac499-10/monografias/lucianna-joao/arquivos/monografia.pdf. Acessado em 04/07/2011.

BENNETT, K. H.; RAJLICH, V. T. Software Maintenance And Evolution: A
Roadmap. **Proceedings Of The Conference On The Future Of Software Engineering**, P.73-87, Limerick, Irlanda, 2000. http://www.cs.wayne.edu/~vip/publications/Bennett.ACM.2000.Roadmap.pdf. Acessado em 04/07/2011.

CHIDAMBER, S. R.; KEMERER, C. F. A Metrics Suite for Object-Oriented Design, **IEEE Transactions on Software Engineering**, vol. 20, pp. 476-493, 1994. http://www.pitt.edu/~ckemerer/CK%20research%20paper p/MetricForOOD_ChidamberKemerer94.pdf. Acessado em 04/07/2011.

CRAWFORD, W. E.; KAPLAN, J. **J2EE Design Patterns**. O'Reilly Media, 2003.

FERREIRA, K. A. M.; BIGONHA, M. A. S.; BIGONHA R. S. Reestruturação de Software Dirigida por Conectividade para Redução de Custo de Manutenção. **Revista de Informática Teórica e Aplicada - RITA/UFRGS**, v. 15, n. 2. Porto Alegre, RS, Brasil, 2008. http://seer.ufrgs.br/rita/article/view/rita_v15_n2_p155-180/4490. Acessado em 04/07/2011.

FERREIRA, K. A. M. **Um modelo de predição de amplitude da propagação de modificações contratuais em software orientado por objetos**, 2011. Tese (Doutorado em Ciência da Computação) – Universidade Federal de Minas Gerais, 2011. http://www.bibliotecadigital.ufmg.br/dspace/bitstream/1843/SLSS-8GYFSX/1/keciaalinemarquesferreira.pdf. Acessado em 04/07/2011.

FOWLER, M.; BECK, K.; BRANT, J.; OPDYKE, W.; ROBERTS, D. **Refactoring: Improving the Design of Existing Code**. Addison-Wesley Professional, 1 edition. 1999.

GAMMA E.; HELM R.; JOHNSON R.; VLISSIDES J. **Design Patterns: Elements of Reusable Object-Oriented Software**. Addison-Wesley, 1995.

HALSTEAD, M. H. **Elements of Software Science**, Elsevier Science Inc. New York, NY, USA 1977.

HERNANDEZ, J.; KUBO, A.; WASHIZAKI, H. **Selection of Metrics for Predicting the Appropriate Application of design patterns**, 2010. http://patterns-wg.fuka.info.waseda.ac.jp/asianplop/proceedings2011/asianplop2011_submission_20.pdf. Acessado em 04/07/2011.

HIGO, Y.; KUSUMOTO, S.; INOUE, K. A metric-based approach to identifying refactoring opportunities for merging code clones in a Java software system. **Journal of Software Maintenance and Evolution: Research and Practice**. v. 20, n.6, p.435-461. Novembro, 2008. http://www.stage-project.jp/kanri/data/ronbun/kj5.pdf. Acessado em 04/07/2011.

IEEE Std. 14764-2006. **Software Engineering - Software Life Cycle Processes - Maintenance**, 2006. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1703974. Acessado em 04/07/2011.

IEEE Std 1219-1998. **IEEE Standard for Software Maintenance**, 1998. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00720567. Acessado em 04/07/2011.

IEEE Std 610.12. **Standard glossary of software engineering terminology**, 1990. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=159342. Acessado em 04/07/2011.

ISO/IEC 9126-1:2001. **Software engineering - Product quality – Part1: Quality model**, 2001.

MÄKELÄ, S. E LEPPÄREN, V. A Software Metric for Coherence of Class Roles in Java Programs. **Proceedings of the 5th International Symposium on Principles and Practice of Programming in Java**. ACM, New York, NY, USA. 2007. http://www.tucs.fi:8080/publications/attachment.php?fname=inpMaLe07c.ppd. Acessado em 04/07/2011.

OMAN, P.; HAGEMEISTER, J. Metrics for assessing a software system's maintainability. **Software Maintenance, Proceerdings, Conference on**, pp.337-344, Nov 1992. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=242525. Acessado em 04/07/2011.

PRESSMAN, R. S. **Engenharia de Software** . 6 ed. McGraw-Hill,2006.

PRESSMAN, R. S. **Software Engineering - A practitioners Approach**. 5 ed. McGraw-Hill,2001.

REPELI, L. R. C. **Refatoração de sistemas Java utilizando padrões de projeto: um estudo de caso**. Dissertação (Mestrado em Ciência da Computação) – Universidade Federal de São Carlos, São Carlos, 2006. http://www.bdtd.ufscar.br/htdocs/tedeSimplificado//tde_busca/arquivo.php?codArquivo=907. Acessado em 04/07/2011.

SOMMERVILLE, I **Engenharia de software**. Pearson Addison Wesley, 8. ed. Rio de Janeiro, 2007.

VOKÁČ, M.; TICHY, W.; SJØBERG, D. I. K.; ARISHOLM, R.; ALDRIN, M. A. Controlled Experiment Comparing the Maintainability of Programs Designed with and without Design Patterns—A Replication in a Real Programming Environment, **Journal Empirical Software Engineering**, v.9 n.3, p.149-195, 2004. http://www.springerlink.com/content/m370523qvm4489h4/fulltext.pdf. Acessado em 04/07/2011.

XIONG, C.J.; XIE, M. E; NG, S.H. Optimal software maintenance policy considering unavailable time. **Journal of Software Maintenance and Evolution: Research and Practice**, v.23, n.1, p.21-33, 2001. http://onlinelibrary.wiley.com/doi/10.1002/smr.467/pdf. Acessado em 04/07/2011.