MELHORIA DA QUALIDADE DA ESTRUTURA INTERNA DE SISTEMAS DE SOFTWARE POR REDUÇÃO DO NÍVEL DE ACOPLAMENTO ENTRE PACOTES

Francielli Pinto, Heitor Costa

Departamento de Ciência da Computação
Universidade Federal de Lavras - Lavras - MG - Brasil
franbarbarap@gmail.com, heitor@dcc.ufla.br

Abstract. *Software should have high degree of maintainability to be easily extended to meet new needs of the user, fix errors and adapt to new technologies. One of the intrinsic characteristics to be improved is the modular independence: software should have low coupling and high cohesion. With the constant changes in software, this characteristic tends to be inappropriate and software internal structure is deteriorated. Refactorings in the source code should be done to recover modular independence. In this paper, the purpose is to present an approach to restructure software packages, by moving classes among packages to improve the internal structure quality. The approach was used (i) considering and (ii) not considering an order of analysis of classes. Five software systems were used to evaluate the proposed approach, whose classes with high coupling and low cohesion in the package level had seen measures improved and, consequently, software quality improved.*

Resumo. *Sistemas de software devem possuir alto grau de manutenibilidade para que eles sejam facilmente evoluídos de modo a atender novas necessidades do usuário, corrigir erros ou adaptá-los a novas tecnologias. Uma das características intrínsecas a ser melhorada é a independência modular, em que o software deve apresentar baixo acoplamento e alta coesão. Com as constantes alterações no software, essa característica tende a não ser mais adequada e a estrutura interna do software deteriora-se. O que se pode fazer é realizar refatorações no código para que a estrutura interna readquira essa independência modular. Neste artigo, o objetivo é apresentar uma abordagem para reestruturar os pacotes de sistemas de software, por meio da movimentação de classes entre pacotes, de forma a melhorar a qualidade da sua estrutura interna. A abordagem foi utilizada (i) considerando e (ii) não considerando uma ordem de análise das classes. Cinco sistemas de software foram utilizados para avaliar a abordagem proposta, cujas classes com alto acoplamento e baixa*

coesão nos pacotes tiveram suas medidas melhoradas e, consequentemente, tiveram melhora na qualidade dos sistemas.

1. Introdução

Qualquer sistema de software está sujeito a alterações/manutenções por causa de contínuas mudanças nos requisitos de usuários, correções de erros, adaptação a novas tecnologias e melhoria de desempenho. Tendo em vista que a manutenção de software é uma das tarefas que consomem mais tempo e esforço durante o seu ciclo de vida, esse sistema deve possuir alto grau de manutenibilidade [ISO/IEC 25000, 2005]. Manutenibilidade é a facilidade com que um software ou componente de software pode ser modificado para corrigir falhas, melhorar desempenho ou adaptar-se a mudanças no ambiente. Uma das formas para aferir a característica manutenibilidade é por meio de medidas que avaliam propriedades de software diretamente relacionadas a essa característica. Por exemplo, medidas de coesão e de acoplamento podem ser utilizadas para avaliar a manutenibilidade.

Acoplamento é o grau em que as classes de um pacote/módulo estão conectadas a classes de outros pacotes/módulos do sistema, por exemplo. Alto acoplamento indica que alterações nessas classes podem resultar em alterações nas classes a que estão acopladas. Coesão é o grau em que classes de um pacote/módulo estão conectadas a outras classes do mesmo pacote/módulo, por exemplo. Pacotes/módulos estão presentes em sistemas orientados a objetos de forma a agrupar classes que implementam funções relacionadas. Assim, a atividade de manutenção em pacotes altamente coesos e baixamente acoplados é menos árdua do que em pacotes com baixa coesão e alto acoplamento. Dessa forma, um sistema de software deve possuir baixo acoplamento e alta coesão em seus componentes (por exemplo, pacotes) [Ragab; Ammar, 2010; Ahmed *et al*., 2011; Shah *et al*., 2012].

As constantes mudanças no software tendem a deteriorar sua estrutura interna, interferindo no grau de coesão e de acoplamento, o que acarreta em baixa qualidade interna. Para recuperar essa qualidade, a refatoração do código fonte deve ser realizada para reestabelecer o patamar apropriado de coesão e de acoplamento de software com qualidade. Refatoração consiste em alterar a estrutura interna de um sistema sem alterar seu comportamento externo [Fowler, 1999]. Neste artigo, o objetivo é apresentar uma abordagem para recuperar a qualidade da estrutura interna de sistemas de software por meio da redução do acoplamento de seus pacotes, movendo classes de um pacote a outro. Cinco sistemas de software Java open source foram utilizados para avaliar a abordagem proposta.

O restante do artigo está organizado da seguinte forma. Conceitos para o entendimento deste trabalho são brevemente destacados Seção 2. Uma metodologia para melhorar a qualidade da estrutura interna de sistemas de software considerando o acoplamento entre pacotes é apresentada na Seção 3. A utilização da metodologia sugerida

em cinco sistemas de software reais é apresentada na Seção 4. A análise dos resultados obtidos é discutida na Seção 5. Alguns trabalhos relacionados estão resumidos na Seção 6. Conclusões, contribuições e sugestões de trabalhos futuros são apresentadas na Seção 7.

2. *Background*

2.1. Qualidade Software

Em Engenharia de Software, há diversas iniciativas para desenvolver sistemas de software com alto grau de qualidade. A gestão da qualidade é um dos assuntos abordados nessa área, cujo objetivo é garantir a qualidade de software. A qualidade é um conceito antigo e subjetivo. De maneira geral, qualidade pode ser entendida como uma característica/atributo de algum produto resultante da execução de um processo de construção/desenvolvimento. No âmbito da Engenharia de Software, tem-se qualidade sob as perspectivas de produto (ISO/IEC 25000 [ISO/IEC 25000, 2005]) e de processo (ISO/IEC 15504 [El-Emam; Garro, 2000], MPS.BR [MPS, 2005] e CMMI [SEI, 2010]). A qualidade de projeto está relacionada às características definidas pelos projetistas que especificam um produto. A qualidade do processo afeta a qualidade do produto relacionada à conformidade com as especificações [Crosby, 1980].

A norma ISO/IEC 25000 é o resultado da revisão da norma ISO/IEC 9126 ("Engenharia de Software - Qualidade de Produto") [ISO/IEC 9126, 2001] e da norma ISO/IEC 14598 ("Tecnologia de Informação - Avaliação de Produto de Software") [ISO/IEC 14598, 1999]; em ambas, são abordadas a qualidade interna, a qualidade externa e a qualidade em uso com seus respectivos atributos de qualidade. Na ISO/IEC 25000, são definidos atributos de qualidade que se espera em um software. Esses atributos estão distribuídos em oito características, divididas em subcaracterísticas [ISO/IEC 25000, 2005]. Entre essas características, a manutenibilidade tem tomado grande atenção, pois estudos empíricos indicam que mais de 90% dos custos do desenvolvimento de sistemas de software são destinados a sua manutenção [Botava *et al*., 2012]. Manutenibilidade é a facilidade com que um software ou componente de software pode ser modificado para corrigir falhas, melhorar desempenho ou adaptar-se a mudanças no ambiente, nos requisitos e/ou nas especificações funcionais.

A qualidade de atributos externos é difícil de ser mensurada, diferentemente da qualidade de atributos internos (menos árdua para ser estimada). A qualidade de atributos internos pode influenciar a qualidade de atributos externos e, portanto, a estimativa da qualidade de atributos externos é obtida pela medição de atributos internos. Dessa forma, diz-se que os atributos internos são indicadores de atributos externos e a qualidade externa de um sistema de software é obtida indiretamente pela medição de seus aspectos internos

[ISO/IEC 25000, 2005]. Por exemplo, a manutenibilidade é influenciada, entre outros, por atributos relacionados às propriedades acoplamento e coesão.

2.2 Medidas

Medidas são utilizadas para melhor entender os atributos dos modelos de qualidade criados e avaliar a qualidade dos produtos submetidos à engenharia construída, sendo elemento-chave para qualquer processo de engenharia. Dessa forma, no âmbito da Engenharia de Software, medidas têm sido propostas para avaliar a qualidade de sistemas de software [Briand *et al*., 2001; Dagpinar; Jahnke, 2003]. Essas medidas fornecem base quantitativa para o desenvolvimento e para a manutenção de sistemas [Tahvildari; Singh, 2000; Barker; Tempero, 2007]. Além disso, elas são "boas" condutoras para investigar as propriedades de sistemas de software [Bruntink; Deursen, 2004], auxiliando durante o seu ciclo de vida para alcançar a qualidade externa.

Assim como há qualidade sob as perspectivas de produto e de processo e há qualidade interna e externa, há medidas de produto e de processo e medidas internas e externas de software, respectivamente. Nesses casos, essas medidas são indiretas, pois não medem a qualidade em si, mas as manifestações de qualidade, medindo fatores de qualidade e quais fatores afetam a qualidade. O complicador é a precisa relação entre um fator de qualidade e a qualidade em si; por isso, estudos empíricos têm sido realizados na literatura para investigar essa relação, sendo muitas vezes comprovado, por exemplo, coesão e acoplamento como indicadores de manutenibilidade. Medidas internas medem atributos internos de software utilizando a quantidade/frequência de elementos que compõem o sistema, tais como, linhas de código, quantidade de métodos, quantidade de atributos e quantidade de chamadas a métodos. Tendo em vista que medidas internas são indicadores de atributos externos, elas têm por objetivo assegurar a qualidade externa do sistema e oferecer às partes interessadas a possibilidade de avaliar a qualidade do sistema em qualquer fase do seu ciclo de vida [ISO/IEC 25000, 2005].

Dentre as medidas de acoplamento e de coesão, podem ser destacadas *Coupling Between Objects* (CBO) e *Lack of Cohesion in Methods* (LCOM), respectivamente. A medida CBO é a quantidade de classes a que uma determinada classe está acoplada, medindo o acoplamento entre classes. O acoplamento entre duas classes normalmente se dá quando métodos de uma classe chama métodos ou instancia variáveis de outra classe [Basili *et al*., 1996; Benlarbi *et al*., 2000]. A medida LCOM é a diferença entre quantidade de pares de métodos com grau zero de similaridade e a quantidade de pares de métodos com grau de similaridade diferente de zero, medindo a falta de coesão de métodos de uma classe [Basili *et al*., 1996]. As medidas CBO e LCOM possuem variações na obtenção de seus valores. As variações da medida LCOM diferem-se na forma como consideram a similaridade entre métodos e/ou na forma como calculam a medida. As variações da

medida CBO diferem-se em termos de granularidade ou em quais tipos de acoplamento entre classes considera no cálculo da medida.

3. Abordagem Metodológica para Melhorar a Qualidade da Estrutura Interna de Software

Neste trabalho, foi considerada como medida de acoplamento de uma classe `C` em um pacote `P` a quantidade de classes de outros pacotes utilizadas por `C`. Para isso, foi utilizada a medida CBO com a semântica original [Chidamber; Kemerer, 1994] sutilmente modificada. Além disso, como medida de coesão, foi utilizada a quantidade de classes do mesmo pacote de uma classe `C` que ela utiliza, sendo que uma classe utiliza outra quando "chama" (envia mensagens para) métodos ou há instâncias (objetos) de outra classe. Assim, para cada classe, foram verificados os tipos dos seus atributos, o tipo de retorno, o tipo de parâmetros e o corpo de cada método declarado na classe para identificar as classes utilizadas.

Para fins de ilustração e de cálculo das medidas, foi considerada a representação de um grafo direcionado que permite arestas direcionadas de um vértice para outros vértices. Os vértices representam as classes do sistema de software a ser analisado e as arestas representam relacionamentos entre classes. O nome da classe foi atribuído como identificador aos vértices; quanto às arestas, elas podem ou não ter identificador (nome) do pacote destino (Figura 1). Por exemplo, uma aresta com o identificador $\texttt{pacoteP}_2$ saindo do vértice $\texttt{classeC}_1$ para o vértice $\texttt{classeC}_5$ indica que a $\texttt{classeC}_1$ "usa" a $\texttt{classeC}_5$ e que a $\texttt{classeC}_5$ pertence ao pacote $\texttt{pacoteP}_2$ que não contém a $\texttt{classeC}_1$. Assim, a $\texttt{classeC}_1$ está "acoplada" ao pacote $\texttt{pacoteP}_2$. A aresta para representar relacionamento entre duas classes (por exemplo, $\texttt{classeC}_3$ e $\texttt{classeC}_2$) de um mesmo pacote não possui identificador.

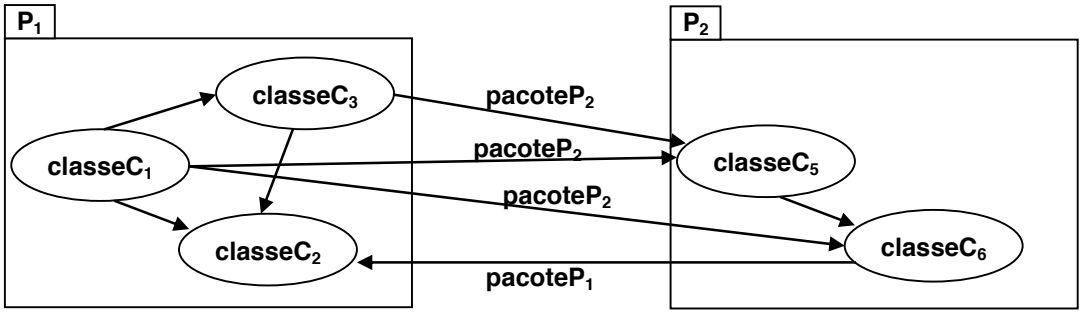

Figura 1 - Grafo Hipotético para representar Conexões entre Classes de um Software

Dessa forma, o cálculo das medidas de coesão e de acoplamento é facilitado, pois a coesão de um pacote corresponde à quantidade de arestas sem identificador e o

acoplamento entre pacotes corresponde à quantidade de arestas com identificador. Além disso, quanto à implementação da eventual movimentação de classes entre pacotes para aumentar a coesão e/ou diminuir o acoplamento, ela é facilitada, pois, para mover uma classe de um pacote a outro, o identificador das arestas envolvidas nessa movimentação é alterado. Considerando os pacotes apresentados na Figura 1, caso a $\texttt{classeC}_1$ seja movimentada do pacote $\texttt{P}_1$ para o pacote $\texttt{P}_2$, o identificador das arestas da $\texttt{classeC}_1$ para a $\texttt{classeC}_3$ e para a $\texttt{classeC}_2$ é alterado para $\texttt{pacoteP}_1$ e o identificador das arestas da $\texttt{classeC}_1$ para a $\texttt{classeC}_5$ e para a $\texttt{classeC}_6$ é "apagado".

Para identificar as classes a serem movimentadas entre pacotes, são consideradas as classes que apresentam o valor da medida de acoplamento maior que o valor da medida de coesão. Considerando uma classe `C`, é identificado qual pacote `C` está mais acoplada ("usa" mais classes) e é verificada se a quantidade de classes do pacote identificado que `C` utiliza (acoplamento) é maior que o valor da coesão da `C`. Se o valor do acoplamento é maior que o valor da coesão, `C` é movimentada para o pacote identificado; caso contrário, `C` permanece no pacote, pois pode aumentar o acoplamento entre os pacotes ao movê-la, o que não condiz com os objetivos da reestruturação. Isso deve ser realizado para todas as classes.

A ordem de análise das classes para movimentá-las entre os pacotes pode interferir no resultado final, pois, ao mover uma classe de um pacote para outro, pode impactar negativa ou positivamente nos valores de acoplamento e de coesão de outras classes, não sendo possível ser revertido. Assim, algumas situações devem ser consideradas para estabelecer uma ordem de análise das classes:

- Inicialmente, a análise das classes mais utilizadas por classes "fora do pacote" que ela pertence deve ser realizada, pois a quantidade de classes impactadas é maior ao movê-las. A ideia é a viabilidade de reverter a movimentação ao impactar negativamente `n` classes, pois a probabilidade da classe impactada ainda não ter sido verificada é maior e ela ainda poderá ser movimentada de forma a melhorar suas medidas;
- Em seguida, a análise das classes mais utilizadas por classes "dentro do pacote" que ela pertence deve ser realizada, pois, ao movimentar uma classe utilizada por `n` classes do mesmo pacote que pertence, `n` classes são impactadas negativamente tendo aumento no acoplamento e redução na coesão. A ideia é a mesma do caso anterior;
- Por fim, a análise das classes que utilizam menos classes, pois elas são as menos impactadas durante o processo de reestruturação.

Assim, devem ser seguidos passos para estabelecer uma ordem de análise das classes que apresente melhores resultados e, em seguida, aplicar a abordagem (Algoritmo 1).

```
PARA cada classe FAÇA:

      Computa coesão;
      Computa acoplamento;

FIM_PARA;


PARA cada classe FAÇA:

      SE acoplamento > coesao ENTÃO:

            pacoteA : = pacote a qual a classe está mais acoplada;

            SE quantidade de classes que a classe usa em pacoteA > coesão ENTÃO:

                  Move classe para pacoteA;

                  PARA cada classe FAÇA:

                        Recomputa coesão;
                        Recomputa acoplamento;

                  FIM_PARA

            FIM_SE;

      FIM_SE;

FIM_PARA;
```

Algoritmo 1 - Abordagem para Reestruturação das Classes de um Software

4. Avaliação da Metodologia

Para avaliar a metodologia proposta, foram utilizados cinco sistemas de software Java *open source*:

- `BlueJ`[1]. Ambiente integrado Java cujo objetivo é o aprendizado da linguagem, foi desenvolvido na Universidade de Sidney e Universidade de Monash. BlueJ suporta exibição de algumas funções para o usuário, tais como, gráfica da estrutura de classe, edição gráfica e textual, editor de texto, compilador, máquina virtual e depurador;
- `JabRef`[2]. Gerenciador de referência bibliográfica, cujo formato de arquivo nativo é o BibTex, formato padrão Latex;
- `JActor`[3]. *Framework* de ator de alto desempenho projetado para facilitar a escalabilidade vertical. Dentre as funções, podem ser destacadas o envio de até 150 milhões de mensagens por segundo, a inclusão de um construtor de máquina de estado

[1] http://www.bluej.org/
[2] http://jabref.sourceforge.net/
[3] http://sourceforge.net/projects/jactor/

para manter o código de forma clara e simples e a criação de até um bilhão de atores por segundo em um único segmento;

- `JFreeChart`[4]. Biblioteca para a construção de tipos de gráficos mais utilizada, com mais de 2,1 milhões de downloads;
- `JHotDraw`[5]. *Framework* GUI para gráficos técnicos e estruturados.

Inicialmente, foi obtido o valor de algumas medidas adicionais desses sistemas utilizando a ferramenta `Google AnalytiX`[6] para caracterizá-los (

Tabela 1). Em seguida, foi verificada a utilização da metodologia proposta nos sistemas sem estabelecer uma ordem de análise das classes (chamada de Abordagem 1). Posteriormente, foi verificada a utilização da metodologia proposta nos sistemas com uma ordem de análise das classes, seguindo os critérios de verificação descritos na seção anterior (chamada de Abordagem 2). Em ambas as verificações, as medidas das classes e as duas abordagens, antes e após a reestruturação, foram analisadas. A aplicação da metodologia e obtenção dos resultados se deu por meio de uma ferramenta desenvolvida pelos próprios autores.

Tabela 1 - Medidas Adicionais

Medidas	Sistemas de Software				
	BlueJ	JabRef	JActor	JFreeChart	JHotDraw
Quantidade de linhas de código (incluindo linhas em branco)	186.126	144.157	8.887	226.623	57.020
Quantidade de classes	786	675	89	602	304
Quantidade de pacotes	182	113	18	40	36
Quantidade de métodos	8.459	5.632	320	7.913	3.377
Quantidade de atributos	3.902	4.210	114	2.887	900
Quantidade de linhas de código por método	9,98	12,44	5,25	9,60	7,84
Quantidade de métodos por classe	6,57	3,59	3,04	12,58	7,01
Quantidade de atributos por classe	2,35	2,20	0,99	3,34	1,57

5. Análise dos Resultados

A seguir, são apresentados os resultados obtidos e a análise desses resultados na avaliação da abordagem proposta. Para a análise dos resultados, foram observadas as variáveis:

- **A** - Após a reestruturação, quantidade de classes sem alteração em suas medidas, porém apresentam a medida de acoplamento maior que a medida de coesão;

[4] http://www.jfree.org/jfreechart/
[5] http://www.jhotdraw.org/
[6] https://developers.google.com/java-dev-tools/codepro/doc/analytix

- **B** - Após a reestruturação, quantidade de classes sem alteração em suas medidas, porém apresentam a medida de coesão maior que a medida de acoplamento;
- **C** - Após a reestruturação, quantidade de classes com aumento na medida de acoplamento e redução da medida de coesão e com a medida de acoplamento maior que a medida de coesão;
- **D** - Após a reestruturação, quantidade de classes com aumento na medida de acoplamento e redução da medida de coesão, porém apresentam a medida de coesão maior que a medida de acoplamento;
- **E** - Após a reestruturação, quantidade de classes com aumento na medida de coesão e redução na medida de acoplamento, porém continuam com a medida de acoplamento maior que a medida de coesão;
- **F** - Após a reestruturação, quantidade de classe com aumento na medida de coesão e redução na medida de acoplamento e com a medida de coesão maior que a medida de acoplamento.

Para comparação das duas abordagens e interpretação dos resultados, bem como das variáveis descritas anteriormente, pode ser observado:

- Considerando que é melhor quanto maior a coesão e menor o acoplamento, verifica-se que quanto menor o valor de A melhor, pois, de acordo com a abordagem proposta, se a medida de acoplamento é maior que a medida de coesão, o ideal seria mover a classe para reduzir o acoplamento e aumentar coesão. Isso nem sempre é possível por causa da possibilidade de, ao mover uma classe para o pacote a que está mais acoplada, o acoplamento pode aumentar e a coesão pode reduzir;
- O valor de B não foi considerado na comparação, pois considerando que, com a Abordagem 1, B = 2 e com a Abordagem 2, B = 1, significando que uma classe com a Abordagem 1 não teve suas medidas alteradas, sofreu alterações nas suas medidas com a Abordagem 2. Essa alteração pode ter sido positiva ou negativa;
- Quanto ao valor de C, o ideal é o seu valor ser igual a zero, uma vez que não se objetiva aumentar o acoplamento. Isso nem sempre é possível, pois, ao mover uma classe de um pacote a outro, outras classes têm suas medidas alteradas. Essa alteração pode ser positiva ou negativa. Logo, interpreta-se que é melhor o valor de C estar mais próximo de zero;
- A interpretação do valor de D é a mesma para o valor de C, porém comparando as duas, é preferível valor de D ser maior que o valor de C do que o contrário, pois é preferível ter maior quantidade de classes com medida de coesão maior que medida de acoplamento do que o contrário;
- A interpretação do valor de E se difere da interpretação para o valor de C e valor D, pois a abordagem visa à redução do acoplamento e ao aumento da coesão. Logo, quanto

maior o valor de E, quanto maior a quantidade de classes com aumento de coesão e redução de acoplamento, melhor;
- A interpretação do valor de F é a mesma para o valor de E, ressaltando que é preferível ter valor de F maior que valor de E.

Na

Tabela 2, são apresentados resultados obtidos com a Abordagem 1 (sem ordem de análise das classes). Os seguintes resultados podem ser destacados:
- No BlueJ, ~18% (C + D) das classes tiveram aumento do acoplamento e redução da coesão com a reestruturação sugerida e ~32% (E + F) das classes tiveram redução do acoplamento e aumento da coesão. O resultado final foi positivo;
- No JabRef, ~7% (C + D) das classes tiveram aumento do acoplamento e redução da coesão e ~50% (E + F) das classes tiveram redução do acoplamento e aumento da coesão. O resultado final foi positivo;
- No JActor, ~27% (C + D) das classes tiveram aumento do acoplamento e redução da coesão e 33% (E + F) das classes tiveram redução do acoplamento e aumento da coesão. A quantidade de classes que sofreram impactos negativos com a reestruturação é próxima a quantidade de classes que sofreram impactos positivos, porém inferior, indicando resultado final positivo;
- NJFreeChart, ~16% (C + D) das classes tiveram aumento do acoplamento e redução da coesão e ~33% (E + F) das classes tiveram redução do acoplamento e aumento da coesão. O resultado final foi positivo;
- No JHotDraw, ~5% (C + D) das classes tiveram aumento do acoplamento e redução da coesão e ~30% (E + F) das classes tiveram redução do acoplamento e aumento da coesão. O resultado final foi positivo.

Tabela 2 - Resultados (Abordagem 1)

Variável	Sistemas de Software				
	BlueJ	JabRef	JActor	JFreeChart	JHotDraw
A	67 (8%)	29 (4%)	6 (7%)	71 (12%)	9 (3%)
B	332 (42%)	263 (39%)	30 (34%)	237 (39%)	187 (62%)
C	118 (15%)	32 (5%)	20 (22%)	75 (13%)	7 (2%)
D	21 (3%)	16 (2%)	4 (5%)	19 (3%)	10 (3%)
E	70 (9%)	15 (2%)	4 (5%)	86 (14%)	24 (8%)
F	178 (23%)	320 (48%)	25 (28%)	114 (19%)	67 (22%)
Total	**786**	**675**	**89**	**602**	**304**

Na Tabela 3, são apresentados resultados obtidos com a Abordagem 2 (existe ordem de análise das classes). Os seguintes resultados podem ser destacados:

- No Bluej, ~21% (C + D) das classes com aumento do acoplamento e redução da coesão e ~38% (E + F) das classes com redução do acoplamento e aumento da coesão;
- No JabRef, ~9% (C + D) das classes com aumento do acoplamento e redução da coesão e ~52% (E + F) das classes com redução do acoplamento e aumento da coesão;
- No JActor, 31% (C + D) das classes com aumento do acoplamento e redução da coesão e ~38% (E + F) das classes com redução do acoplamento e aumento da coesão;
- No JFreeChart, ~24% (C + D) das classes com aumento do acoplamento e redução da coesão e ~35% (E + F) das classes com redução do acoplamento e aumento da coesão;
- No JHotDraw, ~7% (C + D) das classes com aumento do acoplamento e redução da coesão e ~30% (E + F) das classes com redução do acoplamento e aumento da coesão.

Tabela 3 - Resultados (Abordagem 2)

Variável	Sistemas de Software				
	BlueJ	JabRef	JActor	JFreeChart	JHotDraw
A	26 (3%)	8 (1%)	3 (3%)	27 (5%)	5 (2%)
B	298 (38%)	250 (37%)	25 (28%)	214 (36%)	185 (61%)
C	146 (19%)	55 (8%)	20 (23%)	122 (20%)	11 (3%)
D	22 (2%)	14 (2%)	7 (8%)	24 (4%)	12 (4%)
E	76 (10%)	9 (2%)	3 (3%)	81 (13%)	19 (6%)
F	218 (28%)	339 (50%)	31 (35%)	134 (22%)	72 (24%)
Total	**786**	**675**	**89**	**602**	**304**

Os dados apresentados na

Tabela 2 e na Tabela 3 foram analisados e a Abordagem 1 e Abordagem 2 foram contrastadas, podendo ser observado:

- BlueJ - redução de ~62,5% no valor de A; aumento de ~23,7% no valor de C; aumento de ~4,8% no valor de D; aumento de ~8,6% no valor de E; aumento de ~22,5% no valor de F. Pode-se verificar que, a diferença entre a quantidade de classes sem alteração em suas medidas com a Abordagem 1 e a quantidade de classes sem alteração em suas medidas com a Abordagem 2 é igual a 75 classes ((67 - 26) + (332 - 298) = 75). Dessa forma, diz-se que 75 classes sem alteração em suas medidas com a Abordagem 1 tiveram alteração com a Abordagem 2, sendo 29 classes ((146 - 118) + (22 - 21) = 29) com aumento de acoplamento e redução de coesão e 46 classes ((76 - 70) + (218 - 178) = 46) com aumento de coesão e redução de acoplamento. Assim, os resultados apresentados pela Abordagem 2 são melhores no projeto BlueJ;
- JabRef - redução de ~72,4% no valor de A; aumento de ~71,9% no valor de C; redução de 12,5% no valor de D; redução de 40% no valor E; aumento de ~5,9% no valor de F. A diferença entre a quantidade de classes sem alteração em suas medidas com a Abordagem 1 e a quantidade de classes sem alteração em suas medidas com a

Abordagem 2 é igual a 34 classes, sendo 21 classes com aumento de acoplamento e 13 classes com aumento de coesão;

- JActor - redução de 50% no valor de A; sem alteração no valor de C; aumento de 125% no valor de D; redução de 75% no valor E; aumento de 24% no valor de F. A diferença entre a quantidade de classes sem alteração em suas medidas com a Abordagem 1 e a quantidade de classes sem alteração em suas medidas com a Abordagem 2 é igual a 8 classes, sendo 3 classes com aumento de acoplamento e 5 classes com aumento de coesão, apresentando pequena melhora nos resultados;
- JFreeChart - redução de ~62% no valor de A; aumento de ~62,7% no valor de C; aumento de ~26,3% no valor de D; redução de ~5,8% no valor E; aumento de ~17,5% no valor de F. A diferença entre a quantidade de classes sem alteração em suas medidas com a Abordagem 1 e a quantidade de classes sem alteração em suas medidas com a Abordagem 2 é igual a 70 classes, sendo 52 classes com aumento de acoplamento e 15 classes com aumento de coesão;
- JHotDraw - aumento de ~44,4% no valor de A; aumento de ~57,1% no valor de C; aumento de 20% no valor de D; redução de ~20,8% no valor E; aumento de ~7,5% no valor de F. A diferença entre a quantidade de classes sem alteração em suas medidas com a Abordagem 1 e a quantidade de classes sem alteração em suas medidas com a Abordagem 2 é igual a 6 classes, sendo todas com aumento de acoplamento.

Em resumo, pode-se dizer que a Abordagem 1 e a Abordagem 2 aumentam a coesão e reduzem o acoplamento, apresentando melhora significativa na qualidade da estrutura interna dos sistemas de software analisados. Além disso, conclui-se que realmente a ordem em que as classes são verificadas influencia no resultado final, porém em alguns projetos, a Abordagem 1 apresentou melhores resultados e em outros, a Abordagem 2 apresentou melhores resultados. Isso impede de assumir uma metodologia padrão no que diz respeito à ordem em que as classes são verificadas.

6. Trabalhos Relacionados

Em um estudo [Alkhalid *et al*., 2011], foi analisado a utilização da técnica de clusterização para realizar refatorações nos pacotes para melhorar a qualidade de software, aumentando a coesão e diminuindo o acoplamento entre pacotes. Foram utilizados quatro algoritmos hierárquicos clusterização e duas técnicas de refatoração. Foi verificado se a utilização de um dos algoritmos de clusterização para refatorar sem/com alteração na quantidade de pacotes melhora a qualidade de software. Observou-se que há aumento na coesão e diminuição do acoplamento entre pacotes, assim há melhora na qualidade de software.

Em outro estudo [Bavota *et al*., 2012], é proposta uma prática para remodularizar os pacotes para a aumentar a coesão entre pacotes. Essa remodularização consiste em

decompor pacotes de baixa coesão, indicados pelo engenheiro de software, em pacotes menores com maior coesão. São utilizadas medidas de acoplamento para medir a coesão entre pacotes, pois pacotes com alto acoplamento possuem baixa coesão. Refatorações são sugeridas e podem ou não ser aceitas. Um estudo de caso é realizado em cinco sistemas para avaliar a prática proposta, sendo observado que as refatorações sugeridas são significativas e que, aplicando-as, obtêm-se pacotes mais coesos sem deteriorar o acoplamento entre pacotes.

Uma prática foi proposta [Shah *et al*., 2012], utilizando a técnica de refatoração Move Class, que consiste em mover classes entre pacotes, para diminuir o acoplamento entre pacotes, aumentando a qualidade de software. A prática consiste em um grafo de dependência, em que os vértices representam classes e as arestas representam dependências entre classes. Refatorações são realizadas no modelo para permitir a avaliação dos impactos dessas refatorações na qualidade do software antes de serem aplicadas no código fonte. Essas refatorações são aplicadas utilizando um *plug-in* do Eclipse desenvolvido pelos autores. Estudos de caso são realizados em três sistemas *open source* para avaliar a prática proposta e é verificada melhora na estrutura destes sistemas.

O presente trabalho difere-se dos demais apresentados anteriormente no aspecto que propõe a reestruturação interna de sistemas de software em nível de pacotes; para isso, foram utilizadas algumas medidas de coesão e de acoplamento, de modo a melhorar essas medidas. A abordagem proposta foi utilizada em cinco sistemas de software *open source* e os resultados obtidos foram analisados, observando melhora significativa na qualidade interna desses sistemas após a reestruturação.

7. Conclusão

Visando melhorar a qualidade da estrutura interna de sistemas de software, no presente trabalho foi apresentada uma abordagem para reestruturar sistemas de software por meio da redução de acoplamento entre pacotes, mais especificamente, movimentado classes entre pacotes para aumentar a coesão e diminuir o acoplamento. Para a identificação de classes a serem movidas entre pacotes, foram utilizadas medidas de coesão e de acoplamento. A medida CBO foi ajustada de modo a contabilizar as classes acopladas de outros pacotes. Como medida de coesão, foi considerada a quantidade de classes do mesmo pacote que uma classe "chama" métodos ou instancia variáveis. Uma estrutura de grafo foi utilizada para representação e cálculo das medidas.

Cinco sistemas de software Java *open source* foram utilizados para avaliar a abordagem proposta. Após alguns testes, foi verificada que a ordem de análise das classes de um sistema de software pode interferir no resultado final e, dessa forma, foram assumidas duas versões da abordagem: i) uma em que as classes são verificadas de forma

aleatória; e ii) outra em que as classes são verificadas de acordo com padrões pré-determinados. Após a avaliação das duas versões, foi verificada que a abordagem proposta é capaz de melhorar as medidas de acoplamento e de coesão de classes com baixa coesão e alto acoplamento. Isso representa melhora na qualidade interna dos sistemas de software e ainda que a ordem com que as classes são analisadas interfere no resultado final, porém nenhuma conclusão pode ser tirada com relação à maneira de ordenar as classes a serem analisadas, pois em alguns sistemas de software analisados uma versão da abordagem apresentou melhores resultados e, em outros, a outra versão da abordagem apresentou melhores resultados.

Sugestões de trabalhos futuros estão relacionados a utilização de outras medidas de coesão e de acoplamento para melhorar os resultados obtidos com a reestruturação interna e uma verificação mais minuciosa dos impactos gerados com a movimentação das classes entre pacotes, de modo a reduzir os impactos negativos e aumentar os impactos positivos, ou seja, chegar a um melhor resultado possível para determinado projeto. A avaliação da metodologia proposta por um especialista da área também pode ser abordada.

Referências

Ahmed, M. A.; Abubakar, A.; Alghamdi, J. S. (2011) A Study on Uncertainly Inherent in Class Cohesion Measuments. In: Journal of Systems Arctecture, v. 57, pp. 474-484.

Alkhalid, A.; Alshayeb, M.; Mahmoud, S. A. (2011) Software Refactoring at the Package Level using Clustering Techniques. In: IET Software, v. 5. pp. 276-284.

Barker, R.; Tempero, E. (2007) A Large-Scale Empirical Comparison of Object-Oriented Cohesion Metrics. In: Asia-Pacific Software Engineering Conference, pp. 414-421.

Basili, V. R.; Briand, L. C.; Melo, W. L. (1996) A Validation of Object-Oriented Design Metrics as Quality Indicators. In: IEEE Transactions on Software Engineering, v. 22, pp.751-761.

Bavota, G.; De Lucia, A.; Marcus, A.; Oliveto, R. (2012) Using Structural and Semantic Measures to Improve Software Modularization. In: Empirical Software Engineering.

Benlarbi, S.; El Emam, K.; Goel, N.; Raí, S. (2000) Thresholds for Object-Oriented Measures. In: International Symposium on Software Reliability Engineering, pp. 24-38.

Briand, L. C.; Wust, J.; Lounis, H. (2001) Replicated Case Studies for Investigating Quality Factors in Object-Oriented Designs. In: Empirical Software Engineering, v.6, pp.11-58.

Bruntink, M.; van Deursen, A. (2004) Predicting Class Testability using Object-Oriented Metrics. In: IEEE International Workshop on Source Code Analysis and Manipulation, pp. 136-145.

Chidamber, S. R.; Kemerer, C. F. (1994) A Metrics Suite for Object Oriented Design. In: IEEE Transaction on Software Engineering, v. 20, n. 6, pp. 476-493.

Crosby, P. (1980) Quality Is Free: The Art of Making Quality Certain. McGraw-Hill. 270p.

Dagpinar, M.; Jahnke, J. H. (2003) Predicting Maintainability with Object-Oriented Metrics - An Empirical Comparison. In: Working Conference Reverse Engineering, pp. 155-164.

El-Emam, K.; Garro, I. (2000) Estimating the Extent of Standards Use: The Case of ISO/IEC 15504. In: Journal of Systems and Software. v. 53, Is. 2, pp. 137-143.

Fowler, M. (1999) Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional. 464p.

ISO/IEC 14598. (1999) Information Technology - Software Product Evaluation.

ISO/IEC 25000. (2005) Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE) - Guide to SquaRE.

ISO/IEC 9126-1. (2001) Software Engineering - Product Quality - Part 1: Quality Model.

MPS (2005) Melhoria de Processo do Software Brasileiro. Guia Geral, v. 1.

Ragab, S. R.; Ammar, H. H. (2010) Object Oriented Design Metrics and Tools a Survey. In: International Conference on Informatics and Systems, pp. 1-7.

SEI. CMMI for Development (CMMI-DEV) Version 1.3. Software Engineering Institute. Technical Report CMU/SEI-2010-TR-033. Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 2010.

Shah, S. M. A.; Dietrich, J.; McCartin, C. (2012) Making Smart Moves to Untangle Programs. In: European Conference on Software Maintenance and Reengineering, pp. 359-364.

Tahvildari, L.; Singh, A. (2000) Categorization of Object-Oriented Software Metrics. In: Canadian Conference on Electrical and Computer Engineering, v. 1, pp. 235-239.