PUC Minas
A B A K Ó S
Instituto de Ciências Exatas e Informática

Medindo Acoplamento em Software Orientado a Objeto: A Perspectiva do Desenvolvedor*

Measuring Coupling in Object-Oriented Software Systems: Developer's Perspective

Bruno Rodrigues[1]
Daniel Souza[2]
Eduardo Figueiredo[3]

Resumo

As métricas de acoplamento ajudam a identificar os elementos que possam impactar na qualidade de um *software* orientado a objetos. O alto grau de acoplamento pode afetar alguns atributos externos de qualidade de *software*, como manutenibilidade. Este artigo apresenta os resultados de um estudo com desenvolvedores de *software* de diferentes níveis de experiência em programação que avaliaram o grau de acoplamento de um projeto. Com base nas respostas obtidas, foram relacionadas métricas de acoplamento com o propósito de identificar pontos críticos e melhorar a qualidade e manutenibilidade de sistemas orientados a objetos.

Palavras-chave: Acoplamento. Manutenção de *Software*. Métricas de Acoplamento.

*Submetido em 30/09/2014 – Aceito em 15/10/2014

[1]Mestrando em Sistemas de Informação e Gestão do Conhecimento, pelo Programa de Pós-graduação em Sistemas de Informação e Gestão do Conhecimento pela Universidade Mineira de Educação e Cultura - FUMEC, Brasil– brunorodriguesti@hotmail.com

[2]Analista de Sistemas do Estado de Minas Gerais com pós-graduação em Desenvolvimento de Sistemas, Brasil– danielpucplg@gmail.com

[3]Professor pelo Departamento de Ciência da Computação da Universidade Federal de Minas Gerais (UFMG), Brasil– figueiredo@dcc.ufmg.br

Abstract

Coupling metrics aid in identifying the factors that may have an impact on the quality of object-oriented software systems. The high degree of coupling may affect some external software quality attributes, such as maintainability. This paper presents the results of a study with software developers ranging from different levels of programming experience. Based on answers obtained in this study, metrics of coupling were related with the purpose of identifying critical issues in a software project.

Keywords: Coupling. Software Maintenance. Coupling Metrics.

1 INTRODUÇÃO

A programação orientada a objetos (POO) é uma técnica de programação consolidada no desenvolvimento de sistemas de *software*. Quando adequadamente empregada, POO proporciona um ritmo de desenvolvimento mais rápido e maior qualidade de *software* (Li: Henry, 1993). Um dos benefícios esperados pela adoção de POO é uma melhor manutenibilidade de *software*. Isso traz um ganho em economia de tempo e esforço, ajudando também os programadores a construírem sistemas mais confiáveis e eficientes. No entanto, alguns fatores, entre eles o alto acoplamento, podem interferir negativamente em potenciais ganhos trazidos pelo emprego de POO (BRIAND; WÜST; LOUNIS, 1999). O acoplamento é uma medida da força de associação estabelecida através de uma ligação a partir de uma entidade para outra (BRIAND; DALY; WÜST, 1999). Estas associações entre objetos podem criar um efeito cascata, sendo que o número de invocações entre objetos está relacionado com uma maior probabilidade de alterações em cadeia (BRIAND; WÜST; LOUNIS, 1999).

Sistemas de *software* com baixo acoplamento são mais fáceis de serem mantidos e, consequentemente, geram menor custo para implementação de funcionalidades ou correção de erros. Quanto mais dificuldade se tem para manter o *software*, maior o seu custo (BRIAND; WÜST; LOUNIS, 1999). Assim, reduzir o acoplamento entre as classes do sistema pode facilitar as atividades de manutenção de *software* (GUI; SCOTT, 2009).

Uma variedade de métricas de *software* têm sido propostas e são usadas para avaliar a qualidade de um produto de *software* como a proposta por Chidamber e Kemerer (1994) e Briand e outros (1999b). O uso de métricas de acoplamento para identificar e melhorar os atributos de qualidade de *software* inspiraram diversos estudos como, por exemplo, os trabalhos feitos por Dagpinar e Jahnke (2003), Gui e Scott (2009) e Lanza e Marinescu (2006). Estes e outros trabalhos investigaram como as métricas podem ser usadas para favorecer o desenvolvimento de *software* no paradigma orientado a objetos. Este trabalho, ao contrário dos citados, foca no entendimento dos desenvolvedores para realizar a identificação dos níveis de acoplamento em sistemas orientados a objetos.

Através de um estudo exploratório com 30 programadores que, em sua quase totalidade, atuam na indústria de *software*, foram identificados os componentes com alto acoplamento no sistema *Apache Lucene*. Usando um diagrama simplificado da modelagem do módulo *Search* do *Apache Lucene* e seu código fonte, os desenvolvedores responderam um questionário que tinha como finalidade descobrir o acoplamento neste sistema. Estes desenvolvedores não tiveram acesso a nenhuma métrica durante a execução do estudo. Entretanto, eles tiveram que responder um questionário sobre o raciocínio adotado para identificar os componentes altamente acoplados. O objetivo foi avaliar o entendimento dos desenvolvedores sobre acoplamento de *software*. Em um segundo momento, o presente trabalho coletou, da literatura, métricas de acoplamento comumente adotadas para avaliar manutenibilidade de *software*. Estas métricas

de acoplamento foram relacionadas às respostas dos desenvolvedores; ou seja, o estudo buscou identificar as métricas que os desenvolvedores intuitivamente usaram para avaliar o sistema de *software*.

Os resultados foram analisados e justificados no que se refere às sugestões oferecidas pelos desenvolvedores (i) no que tange à redução do acoplamento entre os objetos analisados e (ii) se estes condizem com técnicas realmente efetivas para redução do acoplamento. Algumas lições aprendidas neste trabalho refletem (i) o conhecimento do uso de classes abstratas e interfaces e (ii) o melhor encapsulamento de objetos para minimizar o acoplamento. Além disso, técnicas de refatoração, como *Extract Method* e *Move Method* (FOWLER, 1999), bem como modelos de arquiteturas como Inversão de Controle e Injeção de Dependência (FOWLER, 1999), foram citados pelos programadores por agregarem conhecimento para diminuir o acoplamento no *software*.

O restante do artigo está organizado da seguinte forma. Na Seção 2 deste trabalho, é apresentado o referencial teórico que fornece base para compreensão e visualização geral do artigo. Na Seção 3, são apresentados os objetivos da pesquisa, o sistema utilizado para análise do nível de acoplamento e a forma como foi conduzido o *survey*. Esta seção também discute a avaliação do nível de experiência dos programadores pesquisados e como o sistema foi analisado por eles. A Seção 4 apresenta os principais resultados do estudo e sua análise. Na Seção 5, são tratadas as ameaças à validade e, na Seção 6, são discutidos os trabalhos relacionados. Por fim, a Seção 7 discute as conclusões obtidas, bem como os projetos para trabalhos futuros.

2 REFERENCIAL TEÓRICO

Grande parte dos esforços de construção de um *software* é realizada em atividades de manutenção (BRIAND; DALY; WÜST, 1999). Quanto mais dificuldade se tem para manter o *software*, maior será o seu custo. Estudos como o de Briand, Wüst e Lounis (1999) e Figueiredo et al. (2008) mostram que o alto grau de acoplamento do *software* impacta negativamente na manutenção, causando efeitos colaterais como mudanças frequentes. Diversas métricas foram propostas, por exemplo, Chidamber e Kemerer (1994), Li e Henry (1993a) e Briand, Wüst e Lounis (1999). Apesar de existirem diversas métricas de acoplamento disponíveis na literatura, observa-se que os desenvolvedores raramente fazem uso destas métricas em projetos reais. Uma possível razão para isso é que as métricas podem não capturar o real entendimento dos desenvolvedores a respeito de acoplamento. Algumas das métricas de acoplamento bastante estudadas na literatura são discutidas a seguir.

As métricas Orientadas a Objeto mais conhecidas pertencem ao conjunto de Chidamber e Kemerer. Este conjunto inclui duas métricas de acoplamento: Acoplamento entre Objetos

(CBO) e Respostas para uma Classe (RFC). CBO é uma métrica que quantifica o número de outras classes que são acopladas a classe corrente. A relação entre as classes pode ir de diversas maneiras. Os relacionamentos entre classes são considerados apenas uma vez, ou seja, múltiplos acessos à mesma classe via métodos ou atributos são contados como um único acesso Chidamber e Kemerer (1994). Por outro lado, RFC é a contagem de métodos que são potencialmente invocados em resposta a uma mensagem recebida por um objeto de uma classe particular (CHIDAMBER; KEMERER, 1994). Isto é computado com a soma do número de métodos de uma classe e o número de métodos externos chamados por ele.

Outras métricas de acoplamento usadas neste artigo são Acoplamento por Mensagens (MPC), Acoplamento baseado em Herança (IH-ICP) (BRIAND; DALY; WÜST, 1999) Acoplamento de Abstração de Dados (DAC), Acoplamento Aferente (Ca), Acoplamento Eferente (Ce) e Número de Associações (NAS). MPC é definido como o número de chamadas únicas efetuadas em uma classe (LI; HENRY, 1993a). A contagem do valor de MPC é calculada por classes incluindo chamadas para todos os métodos, funções e propriedades, sejam em classes ou módulos. Chamadas para definir as propriedades, i.e., métodos *set* e *get*, são contabilizadas separadamente. O IH-ICP conta invocações de métodos dos ancestrais da classe, o ICP o número de chamadas de método de uma classe, ponderadas pelo número de parâmetros. Assim, o IH-ICP considera também invocações de métodos da superclasse (BRIAND; DEVANBU; MELO, 1997). A métrica DAC mede a complexidade de acoplamento causado pelo tipo de dado abstrato, este tipo de acoplamento pode causar violação do encapsulamento (LI; HENRY, 1993b). O Acoplamento Aferente conta o número de classes fora da categoria que dependem de classes dentro da mesma categoria. Enquanto o Acoplamento Eferente conta o número de classes fora da categoria que dependem de classes dentro da mesma categoria (MARTIN, 1994). Já a métrica NAS é uma métrica de projeto que mede o número de associações entre uma classe e seus pares (HARRISON; COUNSELL; NITHI, 1998).

3 CONFIGURAÇÃO DO ESTUDO

Para fomentar o entendimento de acoplamento na manutenção de *software*, foi realizado um estudo de caráter exploratório com 30 desenvolvedores sobre acoplamento em um sistema de *software*. Destes desenvolvedores, 28 são profissionais que atuam na indústria de *software* e os outros 2 são atuantes no meio acadêmico.

A pesquisa foi conduzida através da estratégia empírica *survey*. O *survey* é um modo de colher informações de/ou sobre pessoas para descrever, comparar ou explicar seus conhecimentos, atitudes e comportamentos (WOHLIN et al., 2012). Através deste conceito, pretende-se inferir como diferentes programadores reconhecem alto acoplamento em sistemas orientados a objetos, e como pode ser melhorada sua possibilidade de manutenção. Com isso, são colhidas informações de diferentes pontos de vista sobre o acoplamento e como este pode ser minimi-

zado, melhorando a manutenibilidade do sistema, um dos atributos de qualidade de *software*. Para conduzir a pesquisa foram encaminhadas por e-mail as informações sobre a pesquisa e orientações sobre o preenchimento do formulário. Foi declarada também a não divulgação dos nomes dos participantes da pesquisa, tendo como exclusiva finalidade as respostas sobre o acoplamento no *Apache Lucene* e a caracterização geral dos participantes.

Os participantes tiveram acesso ao diagrama simplificado dos principais componentes do módulo *Search* do *Apache Lucene* exibido na Figura 1 e seu código fonte, ambos disponibilizados através de uma interface *Web*. De posse destes dados, foi solicitado que eles analisassem o projeto e respondessem a um formulário, indicando quais as classes que consideraram estar mais acopladas, justificando o motivo de sua escolha. Entre as questões, também foi perguntado sobre a dificuldade que encontraram para descobrir o grau de acoplamento no projeto e como pode ser melhorada a manutenibilidade do projeto analisado de acordo com o nível de acoplamento encontrado.

3.1 Atividades do Estudo

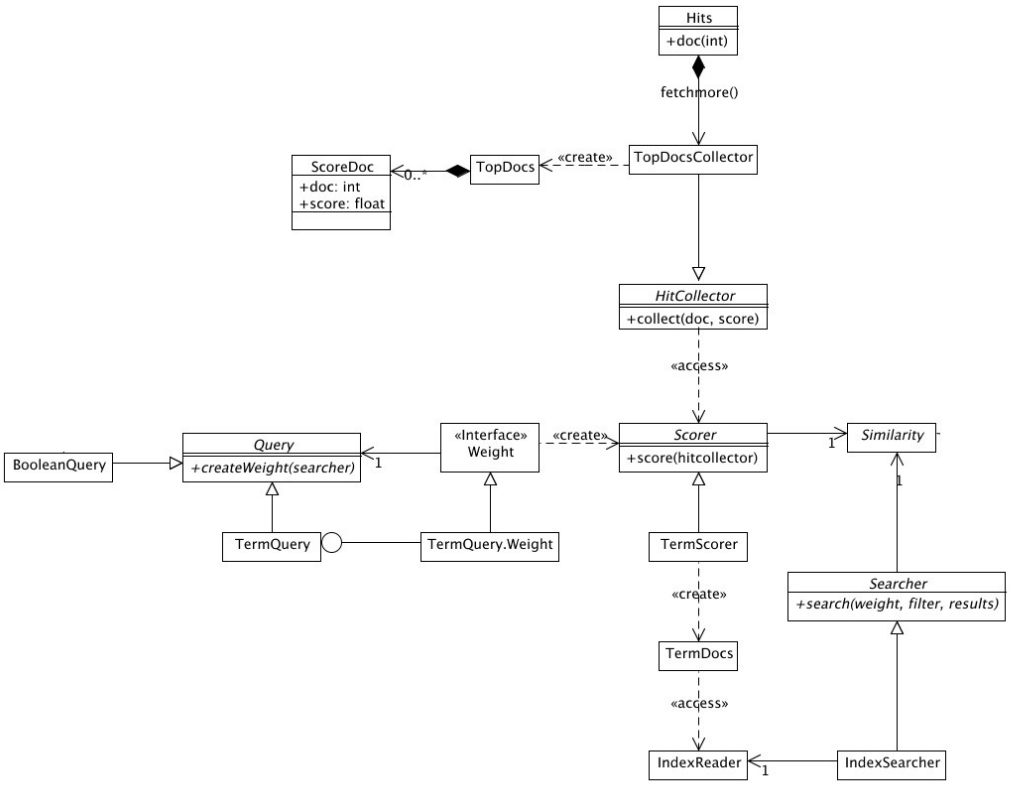
Neste estudo, foi proposta uma análise das principais classes do módulo *Search* da biblioteca *Apache Lucene* através de um questionário aplicado a desenvolvedores de *software* com diversos níveis de experiência profissional. Eles tiveram acesso ao código fonte das classes do sistema e a um Diagrama de Classes simplificado, como mostra a Figura 1.

Os desenvolvedores, então, responderam a um formulário, indicando quais componentes consideraram estar mais acoplados, justificando o motivo de sua escolha e como a manutenibilidade do projeto poderia ser melhorada considerando o nível de acoplamento. Através das respostas dos desenvolvedores, foram identificadas as métricas de acoplamento que intuitivamente os desenvolvedores usam para avaliar a manutenibilidade de sistemas de software orientados a objetos.

3.2 Sistema Utilizado

O *Apache Lucene* é uma biblioteca com recursos de motor de busca de texto, escrita em Java, sendo adequada para quase toda aplicação que requer pesquisa de texto completo (APACHE LUCENE, 2014). A escolha do *Apache Lucene* para este estudo foi motivada por se tratar de um projeto de *software* livre escrito na linguagem de programação *Java*, sendo que o Java é uma das linguagens de programação mais usadas, segundo a pesquisa do Tiobe Software (2014).

Figura 1 – Principais classes do módulo *Search* do *Apache Lucene*.



Fonte: Apache Lucene

3.3 Caracterização dos Participantes

Para classificar os níveis de experiência dos desenvolvedores que participaram deste estudo, foi aplicado um questionário sobre sua experiência profissional. Deste modo obteve-se o conhecimento dos participantes em relação a (i) programação orientada a objetos e *Java*, (ii) participação em projetos de desenvolvimento de *software*, (iii) experiência em anos de desenvolvimento e (iv) os tipos dos projetos em que participaram. De acordo com o questionário respondido por estes, pode-se estimar sua experiência de desenvolvimento de *software*. Assim, o nível de experiência dos desenvolvedores pesquisados pode ser classificado de acordo com a Tabela 1. Os participantes também indicaram seu entendimento a respeito de acoplamento por meio de uma pergunta sobre o que eles consideram acoplamento de *software*. Os dados completos do questionário aplicado estão disponíveis online[4].

Tabela 1 – Experiência dos programadores em anos.

Nível	# de Programadores (anos em experiência)	# de Programadores (conhecimento em POO e Java)
Juniores	7 (1 a 3 anos)	2 (básico)
Intermediários	16 (3 a 5 anos)	20 (intermediário)
Experientes	5 (6 a 10 anos)	6 (avançado)

Fonte: Dados da Pesquisa

[4]Resultados disponíveis em: http://migre.me/lYKHF

4 RESULTADO E DISCUSSÃO DOS RESULTADOS

Os resultados do estudo apontaram que os componentes *TopDocsCollectoreIndexReader* do *Apache Lucene* são os mais acoplados na visão dos desenvolvedores que participaram desta pesquisa, conforme Tabela 2. Observa-se que os participantes puderam escolher mais de um componente como acoplado e não tiveram acesso a nenhuma medição de acoplamento. Os participantes apenas indicaram tais componentes por meio de inspeção de código e do diagrama das principais classes da biblioteca.

Tabela 2 – Componentes indicados como Acoplados.

Componentes	Quantidade de Indicações
TopDocsCollector	13
IndexReader	12
Scorer	9
TermScorer	9
IndexSearcher	9
HitCollector	7
TermDocs	7
Query	7
TermQuery	6
Searcher	5
Hits	5
ScoreDoc	4
BooleanQuery	4
TopDocs	4
Similarity	2

Fonte: Dados da Pesquisa

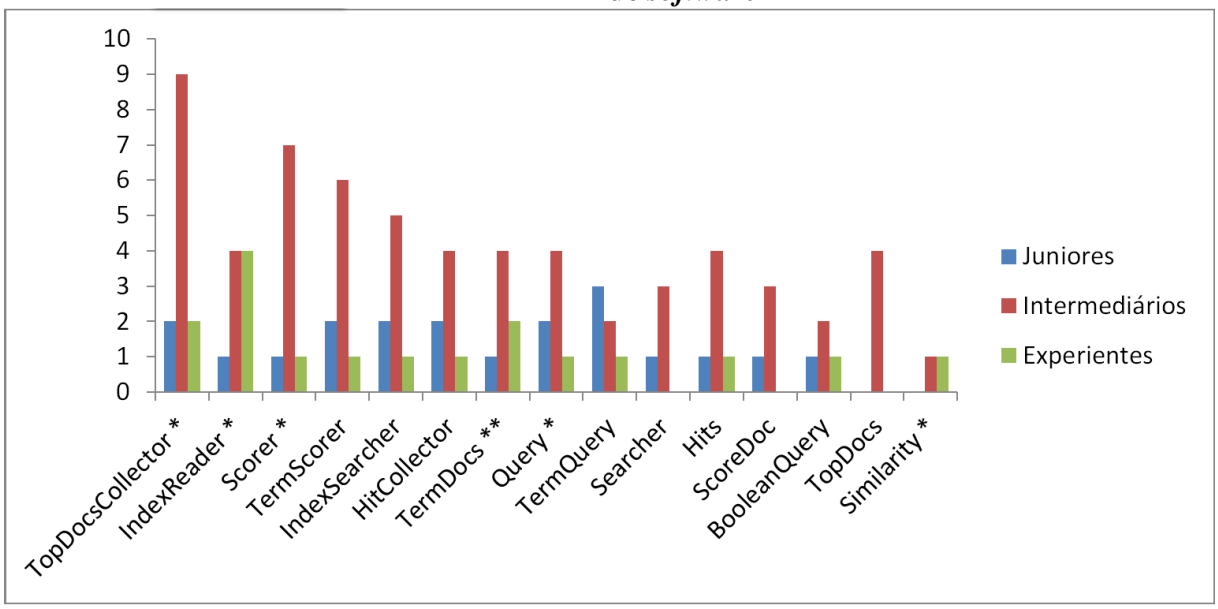
4.1 Identificação do Grau de Acoplamento pelos Programadores

Dos componentes identificados como mais acoplados no sistema, cinco deles são classes abstratas e um é interface. Na Figura 2, as classes abstratas estão anotadas com um asterisco e a interface com dois asteriscos. Entretanto, as classes abstratas promovem aos projetos de sistemas maior reusabilidade de seus componentes (Gamma et. al. 1993), assim como as interfaces na programação orientada a objetos, que têm por finalidade minimizar o grau de acoplamento do sistema (DU BOIS; DEMEYER; VERELST, 2004). As classes abstratas e interfaces apoiam a modularidade de *software* porque elas podem ser facilmente testadas e mantidas (GUI; SCOTT, 2009), sendo, portanto, componentes úteis na programação orientada a objetos não afetando negativamente a manutenibilidade de sistemas.

A Figura 2 mostra que os desenvolvedores juniores foram mais cautelosos ao indicar classes abstratas e interfaces como alto acoplamento no *Apache Lucene*. A Figura 3 reforça

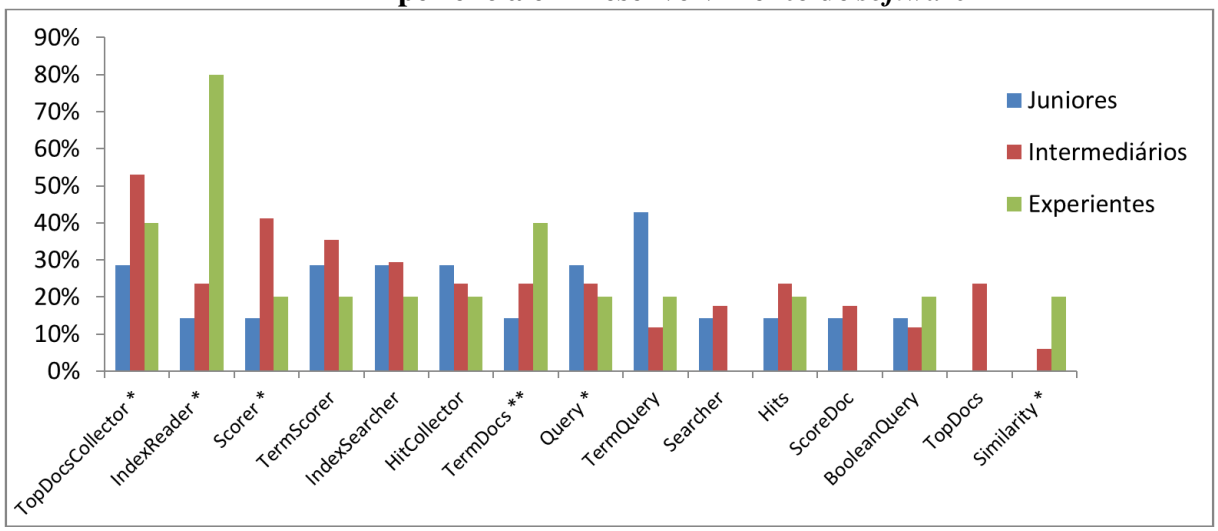
essa afirmativa, onde pode ser vista a porcentagem de desenvolvedores que indicou somente as classes abstratas e interfaces como alto acoplamento.

Figura 2 – Componentes mais acoplados, agrupados pela experiência em desenvolvimento de *software*



Fonte: Dados de Pesquisa

Figura 3 – Porcentagem dos componentes indicados como acoplados, agrupado pela Experiência em Desenvolvimento de *software*



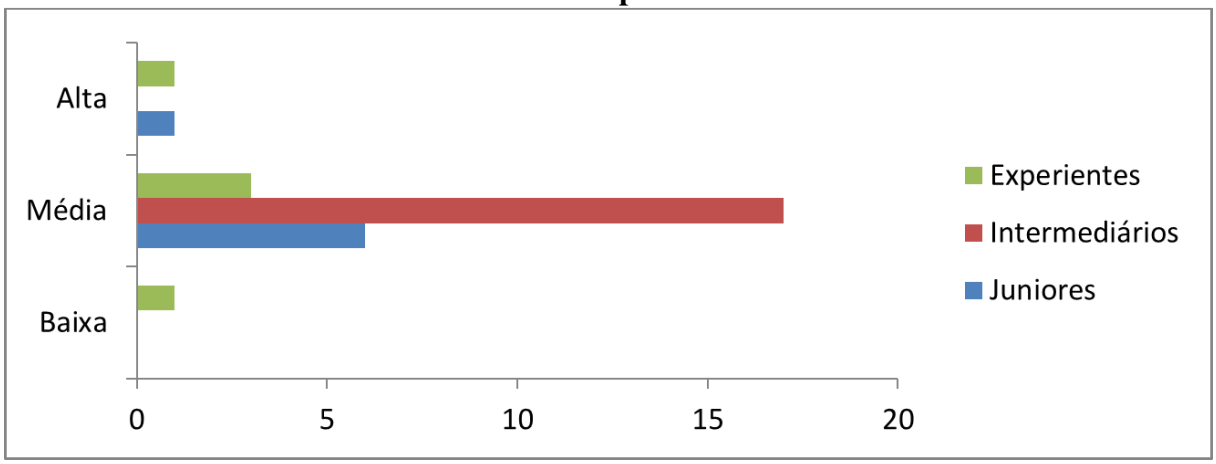
Fonte: Dados de Pesquisa

4.2 Dificuldade de Identificação do Acoplamento

A dificuldade encontrada pelos programadores para identificar classes com alto grau de acoplamento no sistema foi classificada como média por noventa por cento dos participantes pesquisados. Pela Figura 4, pode-se perceber que desenvolvedores mais experientes se dividem também entre baixa e alta dificuldade para se descobrir acoplamento no sistema, enquanto

nenhum desenvolvedor intermediário indicou ser alta ou baixa a dificuldade de encontrar acoplamento no sistema.

Figura 4 – Experiência em desenvolvimento de *software* x dificuldade de encontrar acoplamento



Fonte: Dados de Pesquisa

4.3 Discussão sobre as Classes mais Acopladas

Ao responderem o questionário aplicado sobre o acoplamento no *Apache Lucene*, os desenvolvedores participantes da pesquisa justificaram que conseguiram identificar as classes com alto nível de acoplamento principalmente pela dependência entre classes ou módulos. Tais justificativas podem ser equiparadas às definições clássicas de acoplamento (CHIDAMBER; KEMERER, 1994), tais como uso de métodos de outra classe, a instanciação de atributos, uso de herança, chamadas de outras classes, uso de classes abstratas, classes que disponibilizam os métodos para outras classes e uso de associações. Através das justificativas dadas pelos programadores, pode-se considerar que, intuitivamente, eles consideraram conceitos encontrados nas definições de quatro métricas de acoplamento referenciadas na literatura: CBO, RFC, MPC, DAC, Ca, Ce, NAS e IH-ICP apresentadas na Seção 2. Assim, este estudo conclui que estas métricas aplicam os conceitos mais representativos para encontrar acoplamento na biblioteca *Apache Lucene* e, possivelmente, em outros sistemas Orientados a Objetos.

4.4 Melhorando a Capacidade de Manutenção do Sistema

Para melhorar a capacidade de manutenção do sistema, diminuindo o acoplamento prejudicial ao sistema, os participantes pesquisados sugerem (i) transformar as classes mais acopladas em interfaces, a fim de diminuir o nível de acoplamento destas classes; (ii) aplicar refatoração *Move Method* (TSANTALIS; CHATZIGEORGIOU, 2009) e *Move Field* nas classes mais acopladas, movendo seus métodos e ou atributos para outras classes; (iii) dividir as classes

mais acopladas em duas ou mais classes por meio da refatoração *Extract Class* e fazer uso da Injeção de Dependência e Inversão de Controle (FOWLER, 1999).

A ideia básica de Injeção de Dependência é ter um objeto separado, um montador, que preenche um atributo/variável na classe de listagem com uma implementação apropriada para a interface usada (FOWLER, 1999). Mesmo com a Injeção de Dependência executada, pode haver ainda acoplamento entre classes. Uma solução para isso é usar Inversão de Controle, que se trata de um container que armazena interfaces implementadas (FOWLER, 1999). Com isso, programa-se para interface, reduzindo o acoplamento e potencializando a qualidade do *software*. A interface fornece um limite entre as implementações de uma abstração e seus clientes, limita a quantidade de detalhes de implementação visível para os clientes e também especifica a funcionalidade que as implementações devem fornecer (CANNING et al., 1989). A interface então diminui o acoplamento de uma classe em relação a outra, tornando o código mais fácil de ser mantido.

Foram sugeridas, também, alternativas para redução do acoplamento, a utilização de serviços e padrão de projeto MVC. Apesar de serem úteis ao processo em questão, não se aplicam a este trabalho, visto que as classes selecionadas para pesquisa se referem em específico a apenas um módulo do sistema *Apache Lucene*.

5 AMEAÇAS À VALIDADE DA PESQUISA

Alguns fatores podem comprometer o resultado do presente trabalho. Com base nas justificativas de encontrar as classes mais acopladas no *Apache Lucene*, dois dos participantes não souberam responder como descobriram as classes mais acopladas. Treze não deram sugestões concretas e/ou não souberam responder como melhorar a capacidade de manutenção no sistema através da identificação do nível de acoplamento das classes.

Para esta pesquisa, apenas foram disponibilizados para análise o digrama de relacionamento das principais classes da API com seus respectivos código fontes. Apesar de ser possível identificar as classes mais acopladas no sistema com estes artefatos, outras partes da documentação do projeto, como o diagrama de classes, poderiam facilitar a identificação das classes mais acopladas. Devido à complexidade da API, apenas as principais classes de sua arquitetura foram analisadas. Contudo, também não foi utilizada nenhuma IDE ou ferramenta automatizada que facilitasse a análise do código fonte pelos participantes. Isto poderia deixá-los mais confortáveis para a análise, facilitando sua leitura e navegação pelos arquivos, no entanto perderíamos o fator humano na análise.

6 TRABALHOS RELACIONADOS

O uso de métricas para melhorar a qualidade e premeditar a manutenibilidade de projetos de *software* são temas bastante recorrentes em pesquisas como trabalhos realizados por Lanza et al. (2006), Li e Henry (1993a) e Gui e Scott (2009). Li e Henry (1993a) usaram as métricas de *software* com o intuito de analisar a manutenibilidade e qualidade de *software*. Eles avaliaram as métricas em dois sistemas comerciais orientados a objetos. Comparando as métricas procedurais com métricas OO, eles concluíram que existe uma forte relação entre métricas e o esforço de manutenção em sistemas orientados a objetos. Ou seja, o esforço de manutenção pode ser premeditado pela combinação de métricas coletadas do código fonte (LI; HENRY, 1993b). Dallal (2013) utilizou as métricas de coesão, acoplamento e tamanho para investigar as relações entre as classes e características de manutenção de três sistemas de código aberto escritos em *Java*, utilizando a análise de regressão multivariada para construir modelos para prever a manutenibilidade das classes dos sistemas. Dubey e Rana (2011) revisaram a literatura para propor um modelo de manutenibilidade baseada na relação das métricas orientadas a objetos e manutenibilidade, onde explorou diversos atributos da manutenção de *software*.

Dagpinar e Jahnke (2003) realizaram um estudo de manutenibilidade de *software* OO, analisando o histórico de versão de dois sistemas com base nas métricas orientados a objetos. Neste trabalho, os autores relacionam métricas como tamanho, herança, coesão e acoplamento buscando relação na manutenibilidade dos *softwares*. Os resultados indicam que as métricas de acoplamento de exportação não são bons indicadores de manutenção. Por outro lado, não houve nenhuma relação significativa entre as métricas de acoplamento indiretas e manutenção, no entanto há a forte relação entre as métricas de importação direta de acoplamento e características de manutenibilidade (DAGPINAR; JAHNKE, 2003). Briand, Devanbu e Melo (1997) apresentaram uma proposta de minimizar o acoplamento em projetos orientados a objetos, particularmente o uso da herança. Eles investigaram o impacto da qualidade de diferentes projetos em C++.

Preocupados com os impactos do acoplamento sobre a manutenção do *software*, Burrows et al. (2010) investigaram as métricas de acoplamento que fortemente impactam na qualidade externa. Contrastando com o presente artigo, os autores exploraram o paradigma da Orientação a Aspectos. Observa-se que grande parte das métricas de acoplamento encontradas para Orientação a Aspectos são adaptações das métricas existentes na Orientação a Objetos.

7 CONCLUSÃO E TRABALHOS FUTUROS

Percebe-se, neste trabalho, que as métricas de acoplamento CBO, RFC, MPC, IH-ICP, DAC, Ca Ce NAS são intuitivamente utilizadas pelos desenvolvedores de *software* para identi-

ficar acoplamento prejudicial a sistemas orientados a objeto, portanto, são potencialmente úteis quando coletadas e quantificadas as medições das referidas métricas. Caso o projetista tenha como intuito a diminuição do esforço na fase de manutenção do sistema, ele deve ter estas métricas de acoplamento como foco à minimização dos valores medidos.

Ressalta-se que o uso de classes abstratas e interfaces são mecanismos de polimorfismo e reuso na orientação a objetos, o que é excelente para a manutenção do *software*, quando usadas de maneira adequada. Pela presente pesquisa, é fácil perceber que, institivamente, as classes abstratas e interfaces podem ser confundidas como alta dependência causada pelo forte acoplamento.

Como maneira de se aplicar a redução do acoplamento citado acima, foram mencionadas, pelos profissionais pesquisados, técnicas de desenvolvimento de *software*, tais como o uso de interfaces, o uso de refatoração de código, tais como *ExtractMethod* e *MoveMethod*, injeção de dependência e inversão de controle (FOWLER, 1999). A qualidade do código deve ser mantida para garantir um *software* manutenível, para isso o bom uso da orientação a objeto e padrões de projeto devem ser aplicados. É interessante que a equipe de desenvolvimento faça uma constante avaliação do nível de acoplamento do sistema através das métricas aqui relatadas, para que os esforços de refatoração sejam aplicados de maneira mais precisa.

O baixo acoplamento aumenta a qualidade do *software* produzido, possibilitando maior reuso dos componentes na programação orientada a objetos, seja por alcançarmos classes mais especializadas ou pelo uso de polimorfismo. Ressalta-se que baixo acoplamento é diferente de nenhum acoplamento. Para que dois componentes se comuniquem, sempre haverá uma ligação. Baixo acoplamento significa que um único cenário não afeta um grande número de componentes estruturais (KAZMAN et al., 1996). Ao buscar facilidade de manutenção a longo prazo, deve-se lutar para que essa ligação seja a menor e mais simples possível.

A facilidade de manutenção em sistemas orientados a objetos não tem como único atributo o acoplamento. Para complementar este trabalho, um estudo sobre a coesão também deve ser feito em trabalhos futuros. A coesão pode ser definida como sendo uma medida de grau em que os elementos de um módulo permanecem juntos (BRIAND; WÜST; LOUNIS, 1999). As arquiteturas com baixo acoplamento e alta coesão são mais fáceis de serem modificadas, uma vez que os efeitos das alterações são mais locais se comparados com a ausência dessas qualidades .

Referências

APACHE LUCENE. [S.l.], 2014. Disponível em: <http://lucene.apache.org/>. Acesso em: 30 de jul. 2014.

BRIAND, Lionel C.; DALY, John W.; WÜST, Jürgen K. A Unified Framework for Coupling Measurement in Object-Oriented Systems. **IEEE Transactions on Software Engineering**, IEEE Press, Piscataway, NJ, USA, v. 25, n. 1, p. 91–121, jan 1999.

BRIAND, Lionel C.; DEVANBU, Prem; MELO, Walcelio. An Investigation into Coupling Measures for C++. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 19. **Proceedings...** New York, NY: ACM, 1997. (ICSE '97), p. 412–421.

BRIAND, Lionel C.; WÜST, J.; LOUNIS, H. Using Coupling Measurement for Impact Analysis in Object-Oriented Systems. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE, 1999. **Proceedings...** [S.l.], 1999. (ICSM '99), p. 475–482.

BURROWS, R. et al. The Impact of Coupling on the Fault-Proneness of Aspect-Oriented Programs: An Empirical Study. In: INTERNATIONAL SYMPOSIUM ON SOFTWARE RELIABILITY ENGINEERING, 21, 2010. **Proceedings...** Washington, DC: IEEE, 2010. (ISSRE), p. 329–338.

CANNING, P. S. et al. Interfaces for Strongly-typed Object-Oriented Programming. In: CONFERENCE PROCEEDINGS ON OBJECT-ORIENTED PROGRAMMING SYSTEMS, LANGUAGES AND APPLICATIONS. **Proceedings...** New York, NY, USA: ACM, 1989. (OOPSLA '89), p. 457–467. ISBN 0-89791-333-7.

CHIDAMBER, S. R.; KEMERER, C. F. A Metrics Suite for Object Oriented Design. **IEEE Transactions on Software Engineering**, v. 20, n. 6, p. 476–493, Jun 1994.

DAGPINAR, M.; JAHNKE, J. H. Predicting Maintainability with Object-Oriented Metrics - An Empirical Comparison. In: WORKING CONFERENCE ON REVERSE ENGINEERING, 10, 2003. **Proceedings...** [S.l.], 2003. (WCRE 2003), p. 155–164.

DALLAL, Jehad Al. Object-oriented class maintainability prediction using internal quality attributes. **Journal Info. Soft. Technol.**, Butterworth-Heinemann, Newton, MA, USA, v. 55, n. 11, p. 2028–2048, nov 2013.

DU BOIS, Bart; DEMEYER, Serge; VERELST, Jan. Refactoring - Improving Coupling and Cohesion of Existing Code. In: WORKING CONFERENCE ON REVERSE ENGINEERING, 11. **Proccedings...** Washington, DC: IEEE Computer Society, 2004. (WCRE '04), p. 144–151.

DUBEY, Sanjay Kumar; RANA, Ajay. Assessment of Maintainability Metrics for Object-Oriented Software System. **SIGSOFT Software Engineering Notes**, ACM, New York, NY, USA, v. 36, n. 5, p. 1–7, set. 2011.

FIGUEIREDO, Eduardo et al. Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 30. **Proccedings...** New York, NY: ACM, 2008. (ICSE '08), p. 261–270.

FOWLER, M. **Refactoring**: Improving the design of existing code. 1. ed. [S.l.]: Addison-Wesley, 1999.

GUI, Gui; SCOTT, Paul D. Measuring Software Component Reusability by Coupling and Cohesion Metrics. **Journal of Computers**, v. 4, n. 9, p. 18–21, 2009.

HARRISON, R.; COUNSELL, S.; NITHI, R. Coupling Metrics for Object-Oriented Design Software. In: INTERNATIONAL SOFTWARE METRICS SYMPOSIUM METRICS, 5. **Proceedings...** [S.l.], 1998. p. 150–157.

KAZMAN, R. et al. Scenario-Based Analysis of Software Architecture. **Software, IEEE**, v. 13, n. 6, p. 47–55, Nov 1996.

LANZA, M. et al. **Object-Oriented Metrics in Practice**. [S.l.]: Springer Science & Business Media, 2006.

LI, W.; HENRY, S. Maintenance Metrics for the Object Oriented Paradigm. In: INTERNATIONAL SOFTWARE METRICS SYMPOSIUM, 1, 1993. **Proceedings...** [S.l.], 1993. p. 52–60.

LI, W.; HENRY, S. Maintenance Metrics for the Object Oriented Paradigm. In: PROC. IEEE METRICS. **Proceedings...** [S.l.], 1993. p. 52–60.

MARTIN, R. OO Design Quality Metrics an Analysis of Dependencies. In: WORKSHOP PRAGMATIC AND THEORETICAL DIRECTIONS IN OBJECT-ORIENTED SOFTWARE METRICS, (OOPSLA '94). **Proceedings...** [S.l.], 1994.

TIOBE SOFTWARE. **Tiobe Programming Community Index**. [S.l.], 2014. Disponível em: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>. Acesso em: 02 de set. 2014.

TSANTALIS, N.; CHATZIGEORGIOU, A. Identification of Move Method Refactoring Opportunities. **IEEE Transactions on Software Engineering**, v. 35, n. 3, p. 347–367, Maio 2009.

WOHLIN, C. et al. **Experimentation in Software Engineering**. [S.l.]: Springer, 2012.