JMove: Seus Métodos em Classes Apropriadas

Vitor Sales, Ricardo Terra, Luis Fernando Miranda, Marco Túlio Valente

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
31.270-010 – Belo Horizonte – MG

{vitormsales,terra,luisfmiranda,mtov}@dcc.ufmg.br

Abstract. *This paper presents JMove, a tool to suggest and apply Move Method refactorings based on the structural similarity between methods. Our evaluation on thirteen open-source systems indicates that JMove has been 41% more effective to detect misplaced methods than a state-of-the-art tool.*

Resumo. *Este artigo apresenta JMove, uma ferramenta capaz de sugerir e aplicar refatorações do tipo Move Method com base na similaridade estrutural entre métodos. Em uma avaliação realizada com treze sistemas de código aberto, JMove se mostrou 41% mais eficaz em detectar métodos mal localizados do que uma ferramenta que representa o estado da arte na detecção desse tipo de refatoração.*

1. Introdução

A qualidade de sistemas é sempre um objetivo recorrente em projetos de desenvolvimento de software. Na prática, uma série de diretrizes que promovem a qualidade de sistemas deve ser seguida por processos de desenvolvimento de software, tais como um projeto arquitetural bem definido e o esforço no desenvolvimento de módulos coesos e pouco acoplados.

No entanto, desenvolvedores ao não implementarem métodos nas classes mais apropriadas originam pelo menos dois problemas: (i) *Feature Envy*, que se refere a métodos que utilizam mais – ou que são mais utilizados por – classes diferentes daquela em que está implementado [Fowler 1999]; e (ii) *violações arquiteturais*, que se refere ao fato de um método violar a arquitetura planejada do sistema. Independentemente do problema, uma solução efetiva consiste na aplicação de refatorações *Move Method* [Fowler 1999, Terra et al. 2012]. No entanto, a tarefa de apontar os métodos mal localizados em um sistema não é trivial [Tsantalis and Chatzigeorgiou 2009, Simon et al. 2001, Oliveto et al. 2011, Lanza et al. 2005, Marinescu et al. 2005].

Diante disso, a ferramenta `JMove` implementa uma solução baseada em similaridade estrutural para indicar oportunidades de refatorações *Move Method* [Sales et al. 2013]. Por exemplo, assuma um método de persistência f_p implementado em uma classe de apresentação C_v, `JMove` pode recomendar a movimentação desse método para uma classe de persistência C_p. Em uma avaliação realizada com treze sistemas de código aberto, `JMove` se mostrou 41% mais eficaz em detectar métodos mal localizados do que `JDeodorant`, que constitui a principal ferramenta já proposta com esse objetivo [Tsantalis and Chatzigeorgiou 2009].

O restante deste artigo está organizado como descrito a seguir. A Seção 2 provê uma visão geral da ferramenta `JMove` apresentando seu algoritmo, projeto de implementação, interface gráfica e resultados da avaliação realizada com treze sistemas de código aberto. A Seção 3 discute trabalhos relacionados e a Seção 4 apresenta as considerações finais.

2. A Ferramenta JMove

JMove é um sistema de recomendação de refatorações *Move Method*, como pode ser observado na Figura 1. A partir do código fonte de um sistema Java, a ferramenta calcula a similaridade estrutural de cada método com todas as classes do sistema para recomendar movimentações de métodos que visam melhorar o acoplamento e a coesão do sistema. Ademais, JMove aplica as refatorações escolhidas pelo usuário.

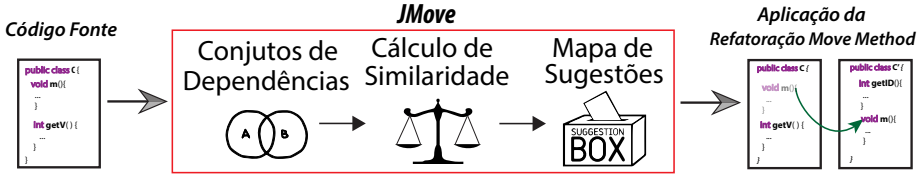


Figure 1. Funcionamento da ferramenta JMove

2.1. Sistema de Recomendação

JMove utiliza um coeficiente de similaridade para identificar métodos mais similares a outras classes do que às classe nas quais estão localizados e, assim, sugerir refatorações *Move Method*. O Algoritmo 1 demonstra o funcionamento da solução proposta.

Algoritmo 1 Sistema de Recomendação

Input: Sistema alvo ***S***
Output: Recomendações de refatorações *Move Method*

1: $Recomendations \leftarrow \emptyset$
2: **for all** $method\ f_i \in \boldsymbol{S}$ **do**
3: $C = \textbf{get\_class}(f_i)$ ▷ Retorna classe atual do método
4: $T \leftarrow \emptyset$ ▷ Armazena lista de classes candidatas
5: **for all** $class\ C_i \in \boldsymbol{S}$ **do**
6: **if** $\textbf{similarity}(f_i, C_i) > \textbf{similarity}(f_i, C)$ **then**
7: $T = T + C_i$
8: **end if**
9: **end for**
10: $C' = \textbf{best\_class}(f_i, T)$
11: $Recomendations = Recomendations + move(f_i, C')$
12: **end for**

Suponha um método f de uma classe C e uma possível classe de destino C'. Se a similaridade entre f e C' for maior que entre f e sua classe atual C (linha 6), a classe C' é uma possível candidata a receber f (linha 7). Sendo T a lista de classes candidatas a receber f, a função best_class(f, T) seleciona a classe C' mais adequada (linha 10). Por fim, uma refatoração para mover f da classe C para a classe C' é adicionada à lista de recomendações gerada pela ferramenta JMove (linha 11).

Nesse algoritmo, três funções são usadas: (i) get_class(f), que retorna a classe em que o método f está implementado; (ii) similarity(f, C), que retorna a similaridade entre um método f e uma classe C; e (iii) best_class(f, T), que dentre uma lista de classes candidatas T, retorna a mais apropriada para receber f. As duas últimas funções são descritas na seção seguinte.

2.2. Principais Módulos

A ferramenta JMove implementa uma solução baseada em três módulos principais:

Módulo de Criação de Dependências: Um método é representado pelo conjunto de tipos com os quais estabelece dependência estrutural. Por exemplo, se f chama o método $Bar::g()$, a classe Bar é inserida em seu conjunto de dependências. Além de chamada de métodos, JMove considera qualquer tipo de dependência existente em sistemas orientados a objetos, tais como acesso a atributos, instanciação, declaração de variáveis,

etc. No entanto, tipos primitivos e classes de uso geral (e.g., `java.util.`$*$) são desconsideradas. Essa decisão é similar àquela usada em técnicas de recuperação de texto que excluem *stop words* porque elas raramente ajudam a descrever o conteúdo de um documento [Baeza-Yates and Ribeiro-Neto 2011].

Módulo de Cálculo de Similaridade: A similaridade entre dois métodos f' e f'' é calculada a partir dos seus conjuntos de dependência. Para selecionar o melhor coeficiente de similaridade para esse propósito, foi conduzido um estudo em que 18 diferentes coeficientes de similaridade foram aplicados ao sistema JHotDraw [Sales et al. 2013]. Por ter obtido os melhores resultados, `JMove` adotou o coeficiente *Sokal and Sneath 2* [Sokal and Sneath 1963] definido por:

$$S_{(f',f'')} = \frac{a}{a + 2(b + c)} \tag{1}$$

onde $\boldsymbol{a}$ é o número de dependências comuns aos dois métodos, $\boldsymbol{b}$ é o número de dependências que ocorrem somente no método f' e $\boldsymbol{c}$ é o número de que ocorrem somente no método f''. O valor $S_{(f',f'')} = 1$ indica a máxima similaridade e $S_{(f',f'')} = 0$ indica que os métodos não possuem dependências em comum. É importante mencionar que, conforme apresentado no Algoritmo 1 (Seção 2.1), a similaridade entre um método f e uma classe C – i.e., $\texttt{similarity}(f, C)$ – é obtida pela média aritmética da similaridade de f com os métodos de C.

Módulo de Recomendação: Ao receber uma lista de classes candidatas T para as quais um método f pode ser movido, a função $\texttt{best\_class}(f, T)$ indica a classe mais apropriada e passível de refatoração automática. Basicamente, a partir da classe com maior similaridade, a primeira que satisfizer as precondições da refatoração *Move Method* é selecionada. Excepcionalmente, existem cenários nos quais a classe atual de um método é muito similar à classe sugerida pelo `JMove`, o que pode contribuir com a geração de falsos positivos. Por exemplo, um método de persistência f_p pode apresentar similaridade superior com alguma outra classe de persistência. Para lidar com tais cenários, uma recomendação é dada desde que a similaridade do método com a classe candidata seja pelo menos 25% superior a sua similaridade com a classe atual.

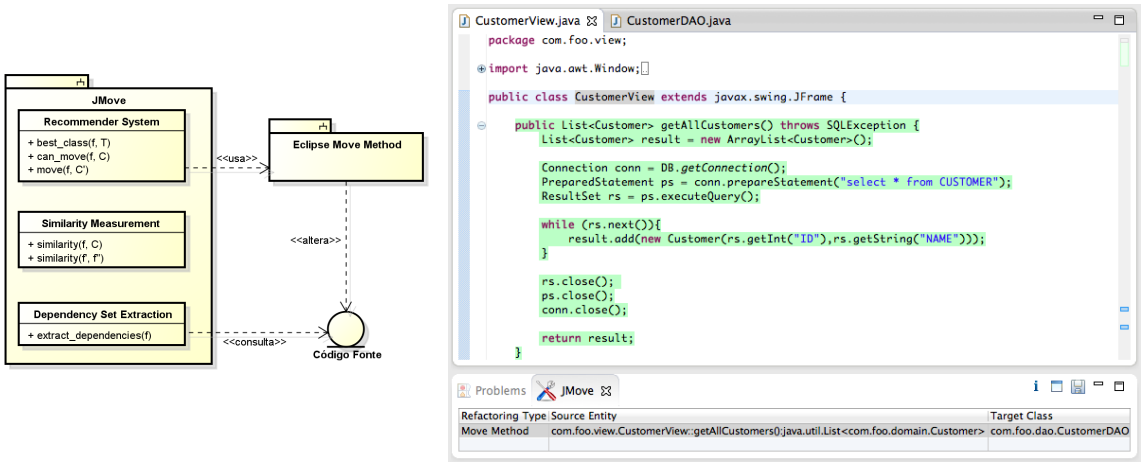
2.3. Arquitetura e Interface

`JMove` foi implementado como um *plugin* para o IDE Eclipse, conforme ilustrado na Figura 2. Basicamente, a ferramenta `JMove` implementa o Algoritmo 1 (Seção 2.1) e os módulos descritos na Seção 2.2. Como um exemplo motivador, a Figura 2(b) ilustra uma sugestão fornecida pela ferramenta `JMove` em que o método de persistência `getAllCustomers()` foi equivocadamente implementado em uma classe de apresentação. `JMove` detecta essa falha e indica que esse método deveria ser movido da classe de apresentação `CustomerView` para a classe de persistência `CustomerDAO`.

O módulo *Dependency Set Extraction*, conforme pode ser observado na Figura 2(a), oferece um serviço que extrai o conjunto de dependências de todos os métodos do sistema a partir do código fonte – $\texttt{extract\_dependencies}(\texttt{f})$. Nesse exemplo, o método `getAllCustomers()` é representado pelo seguinte conjunto de dependências:

$$f = \{\texttt{Customer}, \texttt{DB}, \texttt{SQLException}, \texttt{Connection}, \texttt{PreparedStatement}, \texttt{ResultSet}\}$$

Em seguida, o módulo *Similarity Measurement* é responsável por calcular a similaridade de todos os métodos com todas as classes – por meio dos métodos $\texttt{similarity}(\texttt{f}, \texttt{C})$



(a) Arquitetura (b) Interface

Figure 2. Arquitetura (2a) e interface (2b) do JMove

e $\texttt{similarity}(\texttt{f}', \texttt{f}'')$. Nesse exemplo, a lista de classes candidatas T para o método $\texttt{getAllCustomers}()$ inclui as seguintes classes:

$$T = \{\ [\texttt{CustomerDAO}, 0.94],\ [\texttt{ProductDAO}, 0.86],\ [\texttt{DepartmentDAO}, 0.86],\ \ldots\ \}$$

Por fim, o módulo *Recommender System* seleciona a classe mais apropriada para cada um dos métodos passíveis de refatoração – por meio da função $\texttt{best\_class}(\texttt{f}, \texttt{T})$. Como precondição, conforme descrita na Seção 2.2, é necessário verificar se o método pode ser movido antes de indicar a refatoração – o que é realizado pela função $\texttt{can\_move}(\texttt{f}, \texttt{C})$. Nesse exemplo, a classe mais similar é de fato a classe de persistência `CustomerDAO`. Se o desenvolvedor acatar a sugestão, `JMove` solicita ao *engine* de refatoração do Eclipse que aplique a refatoração *Move Method* – $\texttt{move}(\texttt{f}, \texttt{C}')$.

2.4. Avaliação

A ferramenta `JMove` foi avaliada com 13 sistemas do Qualitas.*class* Corpus [Terra et al. 2013], uma versão compilada do Qualitas Corpus [Tempero et al. 2010]. Similarmente à técnica adotada em alguns estudos [Moghadam and Cinneide 2012, Oliveto et al. 2011], um percentual de 3% dos métodos de cada sistema foram movidos de suas classes originais para outras classes. Tanto os métodos movidos quanto as classes de destino foram selecionados de forma aleatória, porém respeitando as seguintes diretrizes: (i) uma classe poderia apenas ceder ou receber um único método, a fim de não degradar a arquitetura do sistema por completo; e (ii) as movimentações realizadas devem respeitar as precondições da refatoração *Move Method* a fim de poderem ser revertidas.

Assumindo uma versão modificada S' de um sistema S, espera-se que `JMove` recomende que todos os métodos movidos retornem às suas classes de origem e, assim, seja possível restaurar o sistema S original. Mais especificamente, o objetivo dessa avaliação é comparar os resultados obtidos pelo `JMove` com `JDeodorant` [Tsantalis and Chatzigeorgiou 2009], uma ferramenta bastante conhecida nessa linha de pesquisa. A Tabela 1 apresenta os resultados obtidos.

Em uma análise global considerando os 380 métodos movidos, `JMove` indicou que 318 (83,7%) estavam mal localizados e que deveriam ser movidos, enquanto que `JDeodorant` indicou apenas 225 (59,2%). Esse resultado indica que `JMove` foi 41% mais eficaz que o `JDeodorant`. Por fim, é importante mencionar que houve "acertos parciais" em que as

Table 1. Resultado da Avaliação

Sistema	Métodos Movidos	Sugestões Fornecidas		*Recall*	
		JDeodorant	JMove	JDeodorant	JMove
Ant 1.8.2	25	21 + 161*	21 + 136*	84%	84%
ArgoUml 0.34	37	23 + 30*	33 + 48*	62%	89%
Cayenne 3.0.1	47	18 + 105*	38 + 164*	38%	81%
DrJava r5387	18	13 + 293*	15 + 98*	72%	83%
FreeCol 0.10.3	17	10 + 281*	13 + 201*	59%	76%
FreeMind 0.9.0	12	7 + 60*	11 + 65*	58%	92%
JMeter 2.5.1	25	15 + 102*	21 + 64*	60%	84%
JRuby 1.7.3	41	24 + 399*	34 + 357*	59%	83%
JTOpen 7.8	39	23 + 427*	35 + 162*	59%	90%
Maven 3.0.5	24	13 + 56*	16 + 42*	54%	67%
Megamek 0.35.18	35	21 + 243*	29 + 292*	60%	83%
WCT 1.5.2	29	14 + 72*	25 + 44*	48%	86%
Weka 3.6.9	31	23 + 327*	27 + 313*	74%	87%
Total	**380**	**225 + 2.556***	**318 + 1.986***	**59,2%**	**83,7%**

n^* = número de sugestões não relacionadas à avaliação

ferramentas indicaram que o método estava mal localizado, mas não indicaram corretamente a classe de destino. Nesse aspecto, `JDeodorant` obteve 21 "acertos parciais" contra 15 do `JMove`. Além disso, o `JDeodorant` disparou um número superior de sugestões para 9 dos 13 sistemas. Mais especificamente, `JDeodorant` disparou 20% mais sugestões do que o `JMove`.

Observações relevantes: Pelo menos duas observações devem ser realizadas: (i) o experimento conduzido avalia apenas *recall* e considera também "acertos parciais"; e (ii) as demais sugestões não relacionadas à avaliação não são necessariamente falsos positivos, i.e., não se sabe se os métodos estavam mal localizados ou se a sugestão é um falso positivo. Como trabalho futuro, pretende-se conduzir uma avaliação de precisão com a participação de especialistas de cada sistema para verificar as demais sugestões.

3. Ferramentas Relacionadas

Muito esforço tem sido dedicado a proposição de técnicas para apoiar engenheiros de software durante refatorações *Move Method*. Diferentemente do `JMove` que se baseia na similaridade estrutural entre métodos, as ferramentas `iPlasma` – e sua evolução comercial `inFusion`[1] – se baseiam em um conjunto de métricas para detecção de *Feature Envy*, uma das principais manifestações da necessidade de refatorações *Move Method* [Lanza et al. 2005, Marinescu et al. 2005]. Por outro lado, a ferramenta `MethodBook` adota *Relational Topic Model* (RTM) – técnica usada, por exemplo, pelo módulo de sugestões de amizades do Facebook – para identificar oportunidades de refatorações *Move Method* [Oliveto et al. 2011].

Em linhas de pesquisa similares, a ferramenta `Crocodile` emprega a distância *Jaccard* entre métodos e atributos [Simon et al. 2001]. Na prática, a ferramenta exibe as distâncias calculadas em uma perspectiva tridimensional para que o desenvolvedor identifique manualmente as oportunidades de refatoração. Inspirados nesse trabalho, Tsantalis e Chatzigeorgiou verificam os acessos a atributos e métodos para detectar casos de *Feature Envy* e usam o coeficiente de Jaccard para definir uma métrica de coesão considerada pré-requisito para ocorrer uma sugestão de refatoração *Move Method* [Tsantalis and Chatzigeorgiou 2009]. A ferramenta `JDeodorant` – que implementa essa abordagem – indica quais métodos devem ser movidos, além da informação da potencial classe de destino. `JMove` também se baseia em um coeficiente de similaridade, mas considerando o conjunto de tipos com os quais um método estabelece dependência estrutural, e não a quantidade de acessos a atributos ou métodos.

[1] www.intooitus.com/products/infusion

4. Conclusão

Métodos implementados em classes não apropriadas – sejam manifestações de *Feature Envy* ou violações arquiteturais – levam a uma diminuição direta na qualidade do sistema em termos de extensibilidade, flexibilidade e manutenibilidade [Terra et al. 2012]. Diante de tal cenário, é recomendado o emprego de refatorações *Move Method* no intuito de reorganizar o sistema [Fowler 1999]. No entanto, a tarefa de apontar os métodos mal localizados em um sistema não é uma atividade trivial [Tsantalis and Chatzigeorgiou 2009, Simon et al. 2001, Oliveto et al. 2011, Lanza et al. 2005, Marinescu et al. 2005].

Diante disso, a ferramenta `JMove` implementa uma solução baseada em similaridade estrutural para indicar oportunidades de refatorações *Move Method* [Sales et al. 2013]. Em uma avaliação realizada com treze sistemas de código aberto, `JMove` se mostrou 41% mais eficaz para detectar métodos mal localizadas do que `JDeodorant` – uma ferramenta bastante conhecida para detecção de *Feature Envy* [Tsantalis and Chatzigeorgiou 2009].

A ferramenta `JMove` está publicamente disponível em: `http://java.labsoft.dcc.ufmg.br/jmove`

Agradecimentos: Este trabalho foi apoiado pela FAPEMIG, CAPES e CNPq.

References

Baeza-Yates, R. and Ribeiro-Neto, B. (2011). *Modern Information Retrieval*. Pearson, 2nd edition.

Fowler, M. (1999). *Refactoring: improving the design of existing code*. Addison.

Lanza, M., Marinescu, R., and Ducasse, S. (2005). *Object-Oriented Metrics in Practice*. Springer-Verlag New York, Inc.

Marinescu, C., Marinescu, R., Mihancea, P. F., and Wettel, R. (2005). iPlasma: An integrated platform for quality assessment of object-oriented design. In *International Conference on Software Maintenance (ICSM) (Industrial and Tool Volume)*, pages 77–80.

Moghadam, I. H. and Cinneide, M. O. (2012). Automated refactoring using design differencing. In *Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 43–52.

Oliveto, R., Gethers, M., Bavota, G., Poshyvanyk, D., and De Lucia, A. (2011). Identifying method friendships to remove the feature envy bad smell (nier track). In *Proceedings of the 33rd International Conference on Software Engineering (ICSE)*, pages 820–823.

Sales, V., Terra, R., Miranda, L., and Valente, M. T. (2013). Recommending move method refactorings. Technical report, Federal University of Minas Gerais.

Simon, F., Steinbrückner, F., and Lewerentz, C. (2001). Metrics based refactoring. In *5th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 30–38.

Sokal, R. R. and Sneath, P. H. A. (1963). *Principles of Numerical Taxonomy*. Freeman.

Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., and Noble, J. (2010). Qualitas corpus: A curated collection of java code for empirical studies. In *2010 Asia Pacific Software Engineering Conference (APSEC2010)*, pages 336–345.

Terra, R., Miranda, L. F., Valente, M. T., and Bigonha, R. S. (2013). Qualitas.class Corpus: A compiled version of the Qualitas Corpus. *Software Engineering Notes*, pages 1–4.

Terra, R., Valente, M. T., Czarnecki, K., and Bigonha, R. S. (2012). Recommending refactorings to reverse software architecture erosion. In *Proceedings of the 2012 16th European Conference on Software Maintenance and Reengineering*, pages 335–340.

Tsantalis, N. and Chatzigeorgiou, A. (2009). Identification of move method refactoring opportunities. *IEEE Transactions on Software Engineering*, pages 347–367.