Um Estudo Empírico sobre o Impacto dos Pré-processamentos e Normalizações no Cálculo do Acoplamento Conceitual

Paulo Batista da Costa[1], **Igor Wiese**[1], **Igor Steinmacher**[1], **Reginaldo Ré**[1]

[1] Universidade Tecnológica Federal do Paraná - UTFPR
Campus Campo Mourão - PR.

pauloc@alunos.utfpr.edu.br,{igor,igorfs,reginaldo.re}@utfpr.edu.br

Resumo. *O acoplamento é uma das propriedades fundamentais com mais influência sobre a manutenção e evolução do software. Dentre as medidas de acoplamento está o acoplamento conceitual. Na literatura é possível encontrar diversos modelos para cálculo do acoplamento conceitual que combinam pré-processamentos (uso de camelcase, lematização, etc), com diferentes normalizações, tais como, TF-IDF (term frequency–inverse document frequency e o LSI (Latent Semantic Indexing). Este estudo comparou os diferentes modelos de obtenção do acoplamento conceitual entre arquivos de código-fonte. Descobriu-se que a normalização LSI retorna valores mais altos de similaridade entre os artefatos e que os pré-processamentos influenciam de forma diferente as normalizações.*

1. Introdução

A obtenção de similaridade semântica entre arquivos de código-fonte resulta em uma métrica chamada ”acoplamento conceitual”. Essa métrica determina uma relação de interdependência entre arquivos de código-fonte em virtude dos conceitos empregados no desenvolvimento de um projeto de software e é usada com propósitos diferentes, por exemplo, para recomendação de refatoração [Bavota et al. 2011], melhorar a modularidade [Bavota et al. 2010], recomendação de mudanças conjuntas entre artefatos [Kagdi et al. 2013] e uso em técnicas para predizer a ocorrência de defeitos [Kagdi et al. 2013].

Há várias formas de calcular o acoplamento conceitual. Dentre as variações possíveis estão o uso ou descarte do comentário de código-fonte, a remoção ou permanência de termos em relação a frequência que ocorrem, a realização ou não de lematização (do inglês, *stemming*) [Marcus et al. 2008] e o uso de métodos de normalização distintos como o TF-IDF e LSI [Garnier and Garcia 2016, Kagdi et al. 2013]. Um exemplo deste impacto poderia ser observado no uso do *CamelCase*. Esse pré-processamento implica na escrita de palavras compostas, onde cada palavra é iniciada com maiúsculas e unidas sem espaços. Este padrão é utilizado em diversas linguagens de programação como `C++`, `Java`, `Python` e `Ruby` principalmente nas definições de classes e objetos, entretanto, por falta de adoção do padrão um mesmo conceito pode ter sido escrito com alguma variação (*GetCar*, *get_Car*, *getcar*, *Get_Car*) que implicaria no cálculo do acoplamento conceitual.

Apesar da existência de diferentes modelos de cálculo do acoplamento conceitual, não foram encontrados estudos que os comparem em diferentes linguagens de

programação. Portanto, o objetivo deste trabalho consiste em comparar como os pré-processamentos e normalizações combinados em diferentes modelos impactam o valor do acoplamento conceitual obtido. Para a realização deste estudo foram selecionados 61 projetos de software livre, hospedados no `github`, sendo que de cada projeto foram coletadas de 3 a 6 versões. Os projetos selecionados são desenvolvidos em `C++`, `Java`, `Javascript`, `Python` e `Ruby`. Deles foram extraídos as relações de acoplamento conceitual nos arquivos de código-fonte e foram comparadas por meio de testes estatísticos.

Este trabalho esta organizado da seguinte forma. Na seção 2 serão apresentados os conceitos e os trabalhos relacionados. Na seção 3, a metodologia executada. Na seção 4 são apresentados os resultados e a discussão. A seção 5 apresenta as conclusões.

2. Referencial Teórico

2.1. Conceitos

Na Tabela 1 estão descritos os conceitos relacionados ao estudo dos modelos de cálculo do acoplamento conceitual. Na primeira coluna, Conceitos, é apresentado o conceito em si, na segunda coluna, Definição, é apresentada uma breve descrição a respeito do conceito e, na terceira coluna, Tipo, é indicado se o conceito é um pré-processamento (PP) ou uma normalização (NM). Além dos conceitos mostrados na tabela, é importante salientar que o termo **modelo** refere-se a combinação de um conjunto de pré-processamentos (PP) combinado com uma forma de normalização (NM). Portanto, o modelo é o que dá a origem a uma forma de cálculo de acoplamento conceitual.

2.2. Trabalhos Relacionados

[Garnier and Garcia 2016], [Antoniol et al. 2000] e [Safeer et al. 2010] utilizaram a normalização TF-IDF para calcular o acoplamento conceitual. Entretanto, eles diferem nos pré-processamentos usados e, principalmente, no propósito (predição de falhas, mapeamento do código e documentação e clusterização de artefatos). Ao contrário, nosso trabalho pretende estudar o cálculo das métricas ao invés de sua aplicação para algum propósito específico.

[Kagdi et al. 2013], [Marcus 2004] e [Lucia et al. 2007] utilizaram o modelo de normalização LSI e também diferentes pré-processamentos. Novamente, os trabalhos usaram o modelo de normalização para propósitos específicos, enquanto [Kagdi et al. 2013] e [Marcus 2004] usaram os seus modelos para prever mudanças conjuntas, [Lucia et al. 2007] construiu uma ferramenta para rastrear artefatos.

3. Metodologia

3.1. Questões de pesquisa

O objetivo desta pesquisa consiste em analisar se o modelo de cálculo do acoplamento conceitual impacta os valores de acoplamento conceitual. As seguintes questões de pesquisa foram investigadas neste trabalho.

QP1: A variação dos parâmetros de pré-processamento influenciam a técnica de normalização usada no modelo de cálculo do acoplamento conceitual?

QP2: É possível determinar o melhor modelo (pré-processamento + técnica de normalização) para cálculo do acoplamento conceitual?

Tabela 1. Definições de Conceitos.

Conceito	Definição	Tipo
Token	Representação de um termo presente em arquivos de código-fonte.	PP
CamelCase	Composição de termos pela junção de duas ou mais palavras com o uso de letras sensíveis ao caso (ou *underscore*).	PP
Tokenização	Processo que divide o código-fonte em um cojunto de termos.	PP
Remoção de quartil	Remover os termos cuja frequência esteja localizada num determinado quartil. Um quartil é a quarta parte de uma amostra, sendo que uma amostra possui quatro quartis [Hyndman and Fan 1996]. Remover o primeiro quartil corresponde a remoção dos 25% termos menos frequentes. Remover o terceiro, corresponde a remoção de 75% dos termos menos frequentes .	PP
Lematização (do inglês, *stemming*)	Reduzir um termo à sua base (forma não flexionada). Exemplo: *Development* ao ser lematizado se torna *Develop*.	PP
TF-IDF	Normalização de frequência dos termos, cujos índices de frequência são proporcionais ao número de aparições de um termo em um arquivo e é condicionado ao número de vezes em que aparece em todos os arquivos da amostra. O índice normalizado para cada termo está condicionado ao número de vezes em que ele ocorre no arquivo e do número total de vezes em que ele ocorre no projeto (todos os arquivos), sendo que quanto mais vezes ele ocorrer no arquivo e no projeto maior será o seu valor normalizado.	NM
LSI	Normalização baseada no princípio que palavras que ocorrem no mesmo contexto (mesmo arquivo) possuem semântica semelhantes. Isso possibilita afirmar a semelhança conceitual entre termos, mesmo que os termos não sejam o mesmo, mas que tenha ocorram no mesmo contexto. O valor do termo normalizado será maior em virtude do número de vezes que ele ocorre no arquivo, além disso ele considera termos que aparecem conjuntamente como similares.	NM
Ingênuo	Utilização das frequências dos termos para o cálculo do acoplamento conceitual sem aplicar qualquer normalização	NM

3.2. Coleta de Dados

A primeira etapa de execução deste estudo é a coleta das versões de alguns software de código aberto. Foram obtidos 61 projetos de software, dos quais 10 são desenvolvidos em `C++`, 11 em `Java`, 12 em `Javascript`, 16 em `Python` e 12 em `Ruby`. Ao todo foram analisadas 299 versões, sendo 49 em `C++`, 50 em `Java`, 55 em `Javascript`, 78 em `Python` e 60 em `Ruby`, sendo que foram analisadas de 3 a 6 versões *major* mais recentes por projeto. As versões *minor* ou *beta* foram descartadas uma vez que ela apresentam pouca variação de código, logo as similaridades entre os arquivos computados pela métrica de acoplamento conceitual deve ser irrelevante. De cada uma das versões dos projetos selecionados foram coletados o código fonte usando o comando `git clone (url do projeto)`.

Nossa amostra final é diversa. Todos os projetos estão hospedados no `github` e foram selecionados porque são ativos, que pode ser observado pela quantidade de *forks* e pela popularidade indicada por meio da quantidade de estrelas, e representam diferentes

domínios de aplicação (*frameworks*, ferramentas, etc). Os projetos foram selecionados aleatoriamente dentre os que possuem maiores números de *forks* e estrelas. Dentre eles destacamos Agera Google, Flask, Electron, Angular e a linguagem Ruby.

3.3. Formas de cálculo de Acoplamento Conceitual

Após a coleta das versões, foram executados todos os modelos de obtenção de acoplamento conceitual comparados neste estudo. A Figura 1 representa o fluxo da construção dos modelos de obtenção de valores de acoplamento conceitual e expõem as possíveis variações desse processo. De acordo com a Figura 1, após a leitura dos arquivos de código-fonte, é realizado o processo de *tokenização*, no passo 2. No passo 3, é realizado a remoção de palavras reservadas da linguagem a partir de uma lista. Também são removidos caracteres especiais e números. No passo 4, todo *token* cujo nome é formado por menos do que três caracteres é removido, pois em inglês (idioma no qual os identificadores estão descritos) geralmente as palavras que representam algum conceito possuem três caracteres ou mais. Por exemplo, um identificador de variável com o nome representado pelo *token* 'a' não remete a nenhum conceito, portanto é eliminado nesta etapa.

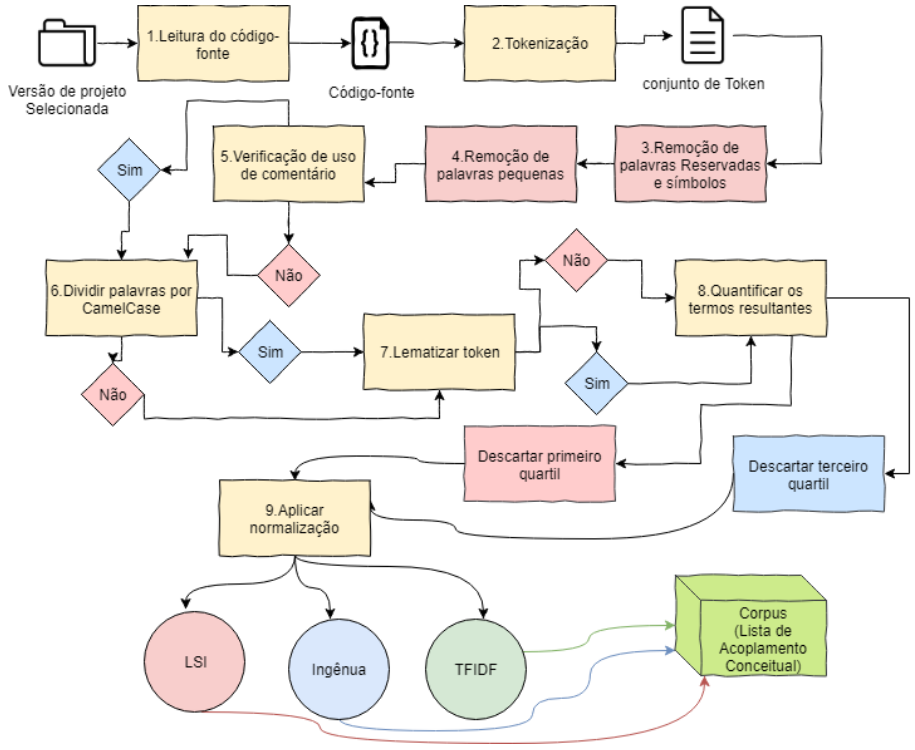

Figura 1. Construção do Corpus para Obtenção de Acoplamento Conceitual

A partir do passo 5 são construídos os modelos comparados neste trabalho. Conforme visto na seção de conceitos, especificamente na Tabela 1, nós comparamos neste trabalho 3 técnicas de normalização (LSI, TF-IDF e ingênuo) com oito combinações de pré-processamento (remoção de comentários de código-fonte, usar comentário de código-fonte, dividir os termos em virtude de *CamelCase*, não dividir os termos por *CamelCase*, lematizar os termos, não lematizar os termos, remoção do primeiro quartil dos termos menos frequentes em arquivos de código-fonte e remoção do terceiro quartil dos termos menos frequentes em arquivos de código-fonte).

É importante mencionar que no passo 8 os termos são quantificados de acordo com a frequência em que ocorrem nos documentos de código-fonte. A partir deste passo, todo documento de código-fonte é representado por um vetor cuja as posições possuem o número de ocorrência para cada termo presente no código-fonte. Em seguida, há a remoção de termos menos frequentes que variam entre o primeiro e o terceiro quartil. Essa remoção é feita para remover os termos menos significantes (termos menos frequentes possuem conceitos menos relevantes para o arquivo), e com ela há a possibilidade de 16 variações do processo de obtenção do acoplamento conceitual.

No passo 9, o último, cada vetor passa pelo processo de normalização, gerando uma matriz onde as linhas e colunas são as combinações par-a-par entre todos os arquivos de código fonte de uma versão com os seus respectivos valores de acoplamento conceitual. Apesar de haverem 16 variações de conjuntos de pré-processamento, para este estudo foram escolhidos 6 configurações apresentadas na tabela 2. Essas configurações foram escolhidas a partir da literatura.

Tabela 2. Combinações de Pré-processamento utilizados neste estudo

Utilizar Comentário	Dividir por *CamelCase*	Lematizar	Remover quartil
sim	sim	sim	remover 3º quartil
sim	sim	não	remover 1º quartil
sim	não	não	remover 3º quartil
não	sim	sim	remover 3º quartil
não	não	sim	remover 1º quartil
não	não	não	remover 1º quartil

3.4. Análise dos Resultados

Primeiramente foi verificado se existe diferença estatística entre as distribuições dos valores de acoplamento conceitual obtidos para pares de arquivos de código fonte usando um modelo de obtenção. Essa verificação foi realizada por meio da aplicação do teste de hipótese *Mann–Whitney U test*. Esse teste foi aplicado para verificar se a distribuição desses valores é diferente para um α = 0.05, α representa o nível de significância. Se o valor-*p* resultante da execução do for menor do que o α, rejeita-se a *hipótese nula*, que neste contexto, pressupõe que as distribuições dos valores de acoplamento gerados entre os pares de arquivos de uma determinada versão são idênticas para os dois modelos comparados.

Após verificar se existe diferença entre as distribuições dos valores de acoplamento conceitual, será aplicado o teste *Cohen's D* (tamanho de efeito), cuja finalidade é dimensionar o tamanho da diferença verificada entre as distribuições avaliadas. Para interpretar o valor do *Cohen's D* obtido, usaremos a escala que indica: ($\delta < 0.1$) insignificante; ($0.1 \leq \delta < 0.2$) muito pequeno; ($0.2 \leq \delta < 0.5$) pequeno; ($0.5 \leq \delta < 0.8$) médio; ($0.8 \leq \delta < 1.20$) grande;($1.20 \leq \delta < 2.0$) muito grande; e, ($\delta \geq 2.0$) enorme. São considerados os melhores modelos aqueles que apresentam o tamanho de efeito maior quando são comparados par a par.

4. Resultados

Foram realizadas 45747 comparações entre diferentes modelos de acoplamento conceitual para responder as questões de pesquisa. Esse número de comparações resulta da

comparação par a par de 18 modelos de obtenção de acoplamento conceitual para as 299 versões de software. Para cada versão foram realizadas 153 comparações. Os resultados das duas questões de pesquisa investigadas neste trabalho são apresentados à seguir.

4.1. QP1: A variação dos parâmetros de pré-processamento influencia a técnica de normalização usada no modelo de cálculo do acoplamento conceitual?

Para responder essa questão de pesquisa, primeiramente foram comparados os valores de acoplamento conceitual obtidos entre todos os pares de arquivos de código fonte encontrados em cada versão de um projeto. Nesta QP, as normalizações foram fixadas, variando somente os pré-processamentos listados na Tabela 2. Não foram realizadas variações quanto a ordem da aplicação dos pré-processamentos do texto do código-fonte, fato que será estudado em trabalhos futuros.

De acordo com a Tabela 3, identificou-se para as linguagens `Javascript`, `Python` e `Ruby`, que o melhor conjunto de pré-processamentos é: usar comentário de código-fonte, dividir termos por *CamelCase*, não lematizar, remover termos menos frequentes com valores inferiores ao primeiro quartil. O mesmo conjunto de pré-processamento é o melhor para as linguagens `C++` e `Java` combinadas com as normalizações TF-IDF e ingênua.

No entanto, o mesmo não ocorreu para as linguagens `C++` e `Java` com o uso do LSI. Neste caso, o melhor conjunto de pré-processamento é formado pelo uso de comentário de código, não divisão de termos por *CamelCase*, não utilização da lematização e remoção do terceiro quartil de termos menos frequentes. Dessa forma, há uma maior remoção de termos menos frequentes nos arquivos de código-fonte, além do fato de que os termos permanecem inalterados em virtude de *CamelCase*.

Tabela 3. Melhores pré-processamentos para cada normalização por linguagem

Linguagem	Pré-processamento	Normalização
`Python`, `Javascript`, `Ruby`	Uso de comentário, divisão por *CamelCase*, Sem Lematização, remover primeiro quartil de termos menos frequentes	LSI,TF-IDF,Ingênua
`C++`, `Java`	Uso de comentário, divisão por *CamelCase*, Sem Lematização, remover primeiro quartil de termos menos frequentes	TF-IDF,Ingênua
`C++`, `Java`	Uso de comentário, não divide por *CamelCase*, Sem Lematização, remover terceiro quartil de termos menos frequentes	LSI

Ao comparar os modelos com normalização LSI entre si, verificou-se que em 76,94% das comparações, os valores de acoplamento conceitual tinham um tamanho de efeito muito pequeno. Isso significa dizer que a maioria dos modelos com normalização LSI retorna valores muito semelhantes de acoplamento conceitual, independentemente da linguagem de programação analisada. O mesmo não aconteceu com os modelos TF-IDF e ingênuo. No caso do TF-IDF, 60,28% das comparações retornaram um tamanho de efeito muito pequeno. No modelo ingênuo, o pré-processamento tem uma influência maior, pois 53,93% das comparações apresentaram tamanho de feito muito pequeno.

Dessa forma, é possível afirmar que a variação dos parâmetros de pré-processamento de texto de código-fonte causam diferenças entre os valores de acoplamento conceitual, especialmente quando a normalização usada é o TF-IDF e o modelo ingênuo. Nota-se que para as linguagens `C++` e `Java` os conjuntos dos melhores pré-processamentos em virtude das normalizações são os mesmos entre si e diferentes do que ocorre nas demais linguagens deste estudo. Isto possivelmente está relacionado as diferentes similaridade de sintaxe do código-fonte escrito em `C++` e `Java`.

4.2. QP2: É possível determinar o melhor modelo (pré-processamento + técnica de normalização) para cálculo do acoplamento conceitual?

Para responder esta questão de pesquisa foram comparados todos os modelos construídos entre si em cada versão de cada linguagem.

De acordo com a Tabela 4, pode-se observar que as linguagens `C++` e `Java` possuem a melhor combinação de pré-processamento e normalização em comum. O mesmo pode ser dito entre as linguagens `Javascript`, `Python` e `Ruby`. Em todas as linguagens os modelos baseados em LSI obtiveram o melhor desempenho. Do total de comparações, os melhores modelos obtiveram em 27,55% dos casos uma vantagem com tamanho de efeito muito pequeno, 19,8% com tamanho de efeito pequeno, 14,91% com tamanho de efeito grande e 35,45% com tamanho de efeito muito grande.

Tabela 4. Melhores combinações de pré-processamento e normalização por linguagem

Linguagem	Pré-processamento	Normalização
`C++`, `Java`	Uso de comentário de código-fonte, não dividir em virtude de *CamelCase*, não lematizar termos, remoção do terceiro quartil de termos menos frequentes.	LSI
`Javascript`, `Python`, `Ruby`	Uso de comentário de código-fonte, dividir em virtude de *CamelCase*, não lematizar termos, remoção do primeiro quartil de termos menos frequentes	LSI

Em termos práticos, em relação a linguagem `C++`, o melhor modelo de acoplamento conceitual apresentou maiores similaridades entre os arquivos de código fonte em 6 projetos (27 versões). No caso da linguagem `Java`, o melhor modelo foi superior em 10 projetos (41 versões). Para a linguagem `Javascript`, foram 7 projetos (30 versões). Na linguagem `Python`, o melhor modelo foi superior em 10 projetos (43 versões). Por último, a linguagem `Ruby` apresentou seu melhor modelo com maiores valores de acoplamento em 8 projetos (33 versões).

Conclui-se que de uma forma geral, usar a normalização baseada no LSI fornece valores maiores da similaridade de acoplamento conceitual entre os arquivos de código fonte. Já o pré-processamento depende da linguagem do projeto que se deseja analisar.

5. Conclusões

Este trabalho estudou o impacto da combinação de diferentes pré-processamentos e técnicas de normalização usadas na literatura. Além disso, ao invés de realizar estudos somente aplicando o acoplamento conceitual com arquivos de código fonte da linguagem Java e C++, foram avaliadas três novas linguagens (ruby, python e Javascript).

Os resultados indicam que os pré-processamentos influenciam o cálculo do acoplamento conceitual, e que o LSI é a melhor normalização (RQ2) - independente da linguagem - para se calcular o acoplamento conceitual. Verificou-se também que as linguagens `C++` e `Java` possuem pré-processamentos comuns, assim como as linguagens `Javascript`, `Python` e `Ruby` e que os pré-processamentos influenciam a obtenção do acoplamento conceitual independentemente da normalização usada (RQ1).

Em trabalhos futuros, serão investigadas questões derivadas deste estudo, tais como: A ordem da aplicação dos pré-processamentos afetam os valores de acoplamento conceitual obtidos? Por quais razões certas combinações de pré-processamento são melhores do que outras de acordo com a linguagem, domínio do projeto e características do código-fonte? Além disso, os melhores modelos de obtenção de acoplamento conceitual encontrados neste estudos serão utilizados em tarefas de predição de mudanças conjuntas e predição de defeitos para se comparar os impactos na prática.

Referências

Antoniol, G., Canfora, G., Casazza, G., and Lucia, A. D. (2000). Information retrieval models for recovering traceability links between code and documentation. In *Proceedings 2000 International Conference on Software Maintenance*, pages 40–49.

Bavota, G., De Lucia, A., Marcus, A., and Oliveto, R. (2010). Software re-modularization based on structural and semantic metrics. In *Reverse Engineering (WCRE), 2010 17th Working Conference on*, pages 195–204. IEEE.

Bavota, G., De Lucia, A., and Oliveto, R. (2011). Identifying extract class refactoring opportunities using structural and semantic cohesion measures. *Journal of Systems and Software*, 84(3):397–414.

Garnier, M. and Garcia, A. (2016). On the evaluation of structured information retrieval-based bug localization on 20 c# projects. In *Proceedings of the 30th Brazilian Symposium on Software Engineering*, pages 123–132. ACM.

Hyndman, R. J. and Fan, Y. (1996). Sample quantiles in statistical packages. *The American Statistician*, 50(4):361–365.

Kagdi, H., Gethers, M., and Poshyvanyk, D. (2013). Integrating conceptual and logical couplings for change impact analysis in software. *Empirical Software Engineering*, 18(5):933–969.

Lucia, A. D., Fasano, F., Oliveto, R., and Tortora, G. (2007). Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 16(4):13.

Marcus, A. (2004). Semantic driven program analysis. In *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*, pages 469–473. IEEE.

Marcus, A., Poshyvanyk, D., and Ferenc, R. (2008). Using the conceptual cohesion of classes for fault prediction in object-oriented systems. *IEEE Transactions on Software Engineering*, 34(2):287–300.

Safeer, Y., Mustafa, A., and Ali, A. N. (2010). Clustering unstructured data (flat files)-an implementation in text mining tool. *arXiv preprint arXiv:1007.4324*.