UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

PAULO BATISTA DA COSTA

ESTUDO EMPÍRICO SOBRE O IMPACTO DAS CONFIGURAÇÕES DO PROCESSO DE OBTENÇÃO DO ACOPLAMENTO CONCEITUAL

MONOGRAFIA

CAMPO MOURÃO

2018

PAULO BATISTA DA COSTA

ESTUDO EMPÍRICO SOBRE O IMPACTO DAS CONFIGURAÇÕES DO PROCESSO DE OBTENÇÃO DO ACOPLAMENTO CONCEITUAL

Trabalho de Conclusão de Curso de graduação apresentado à disciplina de Trabalho de Conclusão de Curso 2, do Curso de Bacharelado em Ciência da Computação do Departamento Acadêmico de Computação da Universidade Tecnológica Federal do Paraná, como requisito parcial para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Igor Scaliante Wiese

CAMPO MOURÃO

2018

ATA DE DEFESA DO TRABALHO DE CONCLUSÃO DE CURSO

Às **20:00** do dia **19 de novembro de 2018** foi realizada na sala **E003** da UTFPR-CM a sessão pública da defesa do Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação do(a) acadêmico(a) **Paulo Batista da Costa** com o título **Estudo Empírico sobre o Impacto das Configurações do Processo de Obtenção do Acoplamento Conceitual**. Estavam presentes, além do(a) acadêmico(a), os membros da banca examinadora composta por: **Prof. Dr. Igor Scaliante Wiese** (orientador), **Prof. Dr. Marco Aurélio Graciotto Silva** e **Prof. Dr. Reginaldo Ré**. Inicialmente, o(a) acadêmico(a) fez a apresentação do seu trabalho, sendo, em seguida, arguido(a) pela banca examinadora. Após as arguições, sem a presença do(a) acadêmico(a), a banca examinadora o(a) considerou _____ na disciplina de Trabalho de Conclusão de Curso **2** e atribuiu, em consenso, a nota _____ (_____). Este resultado foi comunicado ao(à) acadêmico(a) e aos presentes na sessão pública. A banca examinadora também comunicou ao acadêmico(a) que este resultado fica condicionado à entrega da versão final dentro dos padrões e da documentação exigida pela UTFPR ao professor Responsável do TCC no prazo de **onze dias**. Em seguida foi encerrada a sessão e, para constar, foi lavrada a presente Ata que segue assinada pelos membros da banca examinadora, após lida e considerada conforme.

Observações: _____

_____

_____

_____

_____

_____

Campo Mourão, **19 de novembro de 2018**

_____
Prof. Dr. Marco Aurélio Graciotto Silva
Membro 1

_____
Prof. Dr. Reginaldo Ré
Membro 2

_____
Prof. Dr. Igor Scaliante Wiese
Orientador

A ata de defesa assinada encontra-se na coordenação do curso.

Resumo

Batista da Costa, Paulo. Estudo Empírico sobre o Impacto das Configurações do Processo de Obtenção do Acoplamento Conceitual. 2018. 60. f. Monografia (Curso de Bacharelado em Ciência da Computação), Universidade Tecnológica Federal do Paraná. Campo Mourão, 2018.

Contexto:
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*, *stemming*, etc), com diferentes fatores de ponderação, tais como, TF-IDF (*term frequency–inverse document frequency* e o LSI (*Latent Semantic Indexing*). Portanto, é necessário que se estabeleça um estudo avaliativo sobre o impacto da escolha do modelo de cálculo nos valores de Acoplamento Conceitual obtidos e nos processos de manutenção de software, tais como as recomendações de mudanças conjuntas.

Objetivo:
O objetivo deste trabalho consiste em comparar se os pré-processamentos e fatores de ponderação combinados em diferentes modelos geram valores de Acoplamento Conceitual diferentes e como estes modelos distintos impactam a recomendação de mudanças conjuntas de arquivos de código-fonte.

Método:
Foram selecionados 61 projetos de software livre hospedados no `GitHub`, desenvolvidos nas linguagens de programação `C++`, `Java`, `Javascript`, `Python` e `Ruby`. Das versões *major* mais recentes desses projetos (3 a 6 versões), foram obtidos os valores de Acoplamento Conceitual entre seus arquivos de código-fonte por meio de diferentes combinações de pré-processamento textual (*CamelCase*, *stemming*, uso de comentário de código) e normalizações (TF-IDF, LSI). Os valores de Acoplamento Conceitual resultantes desse processo foram comparados entre si por meio de testes estatísticos, a fim de determinar o impacto das variações dos pré-processamento e normalizações nos valores de Acoplamento Conceitual. Em seguida, esses valores foram utilizados para avaliar a eficiência dos maiores valores de Acoplamento

Conceitual para a recomendação de mudanças conjuntas.

Resultados:
Os resultados indicam que combinações diferentes de parâmetro de pré-processamento impactam em distintos valores de Acoplamento Conceitual entre arquivos de código-fonte. Além disso, para as linguagens `C++` e `Java` os pré-processamentos que resultam em maiores valores de Acoplamento Conceitual são comuns. O mesmo ocorre entre as linguagens `Javascript`, `Python` e `Ruby`. Além disso, os modelos de cálculo que resultam nos maiores valores são aqueles que usam normalização LSI, fato que independe do projeto e da linguagem de programação em que ele é desenvolvido. Além disso, constatou-se que as melhores predições foram realizadas a partir de valores de Acoplamento Conceitual calculados por meio de modelo com fator de ponderação TF-IDF (`UcoUcaNsR1qTfidf`[1]), apesar de não resultarem nos maiores valores de Acoplamento Conceitual entre arquivos de código-fonte.

Conclusões:
A partir deste trabalho foi possível constatar a existência de diferenças entre os valores de Acoplamento Conceitual, calculados por meio de modelos distintos. Em virtude desse fato, foi possível selecionar os modelos de obtenção que resultam nas distribuições de maiores valores de Acoplamento Conceitual. Esses valores foram utilizados para a realização de predições de mudanças conjuntas de artefatos. Apesar dos modelos que resultam nos maiores valores de Acoplamento Conceitual terem como normalização o LSI, foram os cálculos por meio modelos com normalização TF-IDF os que proporcionaram predições mais eficientes. Isso mostra que não há relação direta entre distribuições com maiores valores de Acoplamento Conceitual e melhores predições de mudanças conjuntas. Além disso, altos valores não são medidas de qualidade adequada para o Acoplamento Conceitual.

Palavras-chaves: Acoplamento Conceitual. Predição de Mudanças Conjuntas. Similaridade.

[1] Uso de comentário de código-fonte, dividir *CamelCase*, não aplicar *stemming*, remover 1º quartil de termos menos frequentes, uso de fator de ponderação TF-IDF

Abstract

Batista da Costa, Paulo. Empirical Study on the Impact of the Conceptual Coupling Obtaining Process Configurations. 2018. 60. f. Monograph (Undergraduate Program in Computer Science), Federal University of Technology – Paraná. Campo Mourão, PR, Brazil, 2018.

Context: Coupling is a fundamental property with a large influence on the software evolution and maintenance. Among coupling measurements there is the Conceptual Coupling. The literature shows that is possible find many models to calculate Conceptual Coupling values that combine preprocessing (using *CamelCase*, steamming, etc), with differents weighting factor methods, such as TF-IDF (term frequency–inverse document frequency) and LSI (Latent Semantic Indexing). Therefore, it is necessary to establish an evaluative study on the impact of choosing the calculation model on the values obtained and software maintenance processes such as recommendations for co-changes.

Objective: The aim of this work consists in comparing how pre-processing and weighting factor methods combined in different models impact the values of Conceptual Coupling, verifying whether different models result in distinct values of Conceptual Coupling. Therefore, it's necessary to stablish an evaluation about the impact of the choice of the calculation model on the values obtained and software maintenance processes such as co-chance recomandations.

Method: We selected 61 free software projects hosted on GitHub, developed in `C++`, `Java`, `Javascript`, `Python`, and `Ruby` programming languages . From the most recent versions of these projects (3 to 6 versions), the Conceptual Coupling values were obtained among their source code files through different combinations of textual preprocessing (*CamelCase*, stemming, use of code comment) and weighting factor methods (LSI, TF-IDF). The results of this process were compared by means of statistical tests in order to determine the impact of the variations of the pre-processing and weighting factor methods on the values of the event. Then, these values were used to evaluate the efficiency of the highest values of Conceptual Coupling for the recommendation of Co-Changes.

Results: The results indicate that different combinations of pre-processing parameter impact in distinct values of Conceptual Coupling between souce code archives. Moreover, for the

languages `C++` and `Java` the pre-processings that result in bigger values of Conceptual Coupling are common. The same occurs between the languages `Javascript`, `Python` and `Ruby`. Moreover, the calculation models that result in the biggest values are those that use LSI weighting factor method, fact that is independent of the project and the programming language that it is developed. Moreover, one evidence is that the best predictions had been carried through values of Conceptual Coupling by models with TF-IDF weighting factor method, although they do not result in the largest values of Conceptual Coupling between source code files.

Conclusions: From this work it was possible to evidence the existence of differences between distributions of values of Conceptual Coupling, calculated by means of distinct models. In virtue of this fact, it was possible to select the attainment models that result in the distributions of bigger values of Conceptual Coupling. These values had been used for the accomplishment of predictions of Co-Changes. Although the models that result in the biggest values of Conceptual Coupling to have as LSI weighting factor method, had been the models with TF-IDF weighting factor method the ones that had provided more efficient predictions. This sample that does not have a direct relation between distributions with bigger values of Conceptual Coupling and better predictions of Co-Changes.

Keywords: Conceptual Coupling. Co-Change Prediction. Similarity

Lista de figuras

3.1 Visão Geral do método empregado neste estudo 25
3.2 Método: Remoção de Símbolos, Caracteres Especiais, Números e Palavras Pequenas 28
3.3 Método: Fluxo de Variações de Combinações de Pré-Processamentos 29
3.4 Método: Pré-processamento, Aplicação de remoção de termos menos frequentes 29
3.5 Método: Aplicação de Fator de Ponderação dos Termos 30

4.1 Gráficos de Distribuição de Valores de Acoplamento Conceitual. 39
4.2 Distribuições de Valores de MAP Obtidos para Projetos por Linguagem . . 45

Lista de tabelas

3.1 Dados de Projetos `C++` Selecionados para o Estudo 26

3.2 Dados de Projetos `Java` Selecionados para o Estudo 27

3.3 Dados de Projetos `Javascript` Selecionados para o Estudo 27

3.4 Dados de Projetos `Python` Selecionados para o Estudo 27

3.5 Dados de Projetos `Ruby` Selecionados para o Estudo 28

3.6 Combinações de Pré-processamento utilizados neste estudo 31

4.1 Pré-processamentos que resulta em maiores valores de Acoplamento Conceitual para cada fator de ponderação por linguagem 36

4.2 Combinações de pré-processamento e fator de ponderação que resultam em maiores valores de Acoplamento Conceitual por linguagem 38

4.3 Modelos de cálculo de valores de Acoplamento Conceitual utilizados em análises da Segunda Questão de Pesquisa 40

4.4 Diferença de tamanho de efeito entre distribuições de valores de MAP obtidas a partir de modelos com fatores de ponderação TF e LSI 41

4.5 Diferença de tamanho de efeito entre distribuições de valores de MAP obtidas a partir de modelos com fatores de ponderação TF-IDF e LSI 42

4.6 Diferença de tamanho de efeito entre distribuições de valores de MAP obtidas a partir de modelos com fatores de ponderação TF-IDF e TF 43

4.7 Diferença de tamanho de efeito entre distribuições de valores de MAP obtidas a partir de modelos com fatores de ponderação TF e TF-IDF 44

4.8 Números de Arquivos Recomendados por Modelo de Cálculo de Valores de Acoplamento Conceitual 46

1 Versões de Projetos em `C++` Coletadas 54

2 Versões de Projetos em `Java` Coletadas 54

3 Versões de Projetos em `Javascript` Coletadas 54

4 Versões de Projetos em `Python` Coletadas 55

5 Versões de Projetos em `Ruby` Coletadas 55

6 Significado dos acrônimos dos modelos de cálculo de Acoplamento Conceitual 55

7 Modelos que resultaram em maiores Valores de Acoplamento Conceitual, linguagem `C++` 56

8 Modelos que resultaram em maiores Valores de Acoplamento Conceitual, linguagem `Java` 56

9 Modelos que resultaram em maiores Valores de Acoplamento Conceitual, linguagem `Javascript` 56

10 Modelos que resultaram em maiores Valores de Acoplamento Conceitual, linguagem `Ruby` 56

11 Modelos que resultaram em maiores Valores de Acoplamento Conceitual, linguagem `Python` 57

Sumário

1 Introdução **11**

2 Referencial Teórico **14**

2.1 Fundamentação Teórica 14

2.1.1 Versionamento Semântico 14

2.1.2 Recomendação de Mudança Conjunta 15

2.1.3 Acoplamento Conceitual 16

2.1.4 Pré-Processamentos 16

2.1.4.1 *Token* e *Tokenização* 16

2.1.4.2 *CamelCase* 17

2.1.4.3 *Stemming* 17

2.1.4.4 Remoção de Quartil de Termos Menos Frequentes 17

2.1.5 Fatores de Ponderação 18

2.1.5.1 Fator de Ponderação TF-IDF 18

2.1.5.2 Fator de Ponderação LSI 19

2.1.5.3 Fator de Ponderação TF 19

2.1.6 Cosseno 20

2.1.7 Modelos de Cálculo do Acoplamento Conceitual 20

2.1.8 Avaliação de Recomendações de Mudanças Conjuntas 20

2.2 Trabalhos Relacionados 22

2.2.1 O uso do TF-IDF 22

2.2.2 O uso do LSI 22

2.3 Considerações Finais 23

3 Metodologia **24**

3.1 Objetivo 24

3.2 Método 25

3.2.1 Coleta das Versões de Projetos 25

3.2.2 Modelos de cálculo de Acoplamento Conceitual 27

3.3 Questões de Pesquisa 31

3.4 Considerações Finais 33

4 Resultados e Discussão **35**

4.1 A variação dos parâmetros de pré-processamento influencia a técnica de fator de ponderação usada no modelo de cálculo do acoplamento conceitual? . . . 35

4.2 É possível determinar o modelo (pré-processamento + técnica de fator de ponderação) que resulta os maiores valores de Acoplamento Conceitual? . . . 37

4.3 Modelos que resultam no cálculo dos maiores valores de Acoplamento Conceitual produzem melhores predições de mudanças conjuntas? 40

4.4 Ameaças à Validade 47

4.5 Considerações Finais 49

5 Conclusões **50**

Apêndices **53**

Referências **58**

Capítulo 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 aplicação ou não de *stemming* (MARCUS *et al.*, 2008) e o uso de fatores de ponderação distintos como o TF-IDF e LSI (GARNIER; 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 letras maiúsculas e unidas sem espaços. Este padrão é utilizado em diversas linguagens de programação como `C++`, `Java`, `Javascript`, `Python` e `Ruby` principalmente nas definições de classes e objetos e entretanto, por falta de adoção de um 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 fatores de ponderação 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 os valores de Acoplamento Conceitual entre arquivos de código-fonte e foram comparados por meio de testes estatísticos.

A partir do estudo realizado neste trabalho, foi possível verificar que os pré-processamentos influenciam o cálculo do Acoplamento Conceitual. Além disso, o LSI é o fator de ponderação que compõem os modelos de cálculo com maiores valores de Acoplamento Conceitual-independente da linguagem - para se calcular o Acoplamento Conceitual. Verificou-se também que as linguagens `C++` e `Java` possuem pré-processamentos que resultam em maiores valores de Acoplamento Conceitual comuns, assim como as linguagens `Javascript`, `Python` e `Ruby` e que os pré-processamentos influenciam a obtenção do acoplamento conceitual independentemente da fator de ponderação utilizada.

Em virtude da constatação dessas diferenças entre as distribuições de valores de Acoplamento Conceitual, os modelos que resultaram nos maiores valores para as fatores de ponderação LSI, TF-IDF e TF foram selecionados como base para a realização de predições de mudanças conjuntas presentes entre as versões de software selecionadas. A escolha pelos modelos que proporcionaram as distribuições de maiores valores de Acoplamento Conceitual se deu em virtude da intenção de analisar se maiores valores de Acoplamento Conceitual são melhores para atividades de engenharia de software, como por exemplo, a predição de mudanças conjuntas.

Foi constatado que apesar dos modelos com fator de ponderação LSI serem os que resultaram nas distribuições com os maiores valores de Acoplamento Conceitual, o modelo com fator de ponderação TF-IDF (`UcoUcaNsR1qTfidf`[1]) foi o que proporcionou os maiores números de arquivos recomendados corretamente. Isso significa que as predições por realizadas por meio deste modelo foram aquelas que apresentaram distribuições com maiores valores de MAP, o que significa que resultou em predições de mudanças conjuntas de melhor qualidade.

Isso indica que maiores valores não são medidas adequadas de qualidade do Acoplamento Conceitual. De acordo com a observação realizada neste estudo, é possível afirmar que uma variância maior dos valores das distribuições de Acoplamento Conceitual representam medidas mais adequadas, pois o modelo com fator de ponderação TF-IDF foi aquele que apresentou maiores variações e melhores indicativos de predições de mudanças conjuntas, ou seja, maiores valores de MAP.

As observações a respeito das comparações estabelecidas entre modelos com fatores de ponderação LSI e TF-IDF mostram que em 69,52% dos casos o modelo `UcoUcaNsR1qTfidf` resulta em predições de melhor qualidade (maiores valores de MAP). Por sua vez, o modelo

[1] Uso de comentário de código-fonte, dividir *CamelCase*, remoção do 1º quartil de termos menos frequentes, uso de fator de ponderação TF-IDF

`UcoUcaNsR1qTfidf` foi melhor do que modelos com fator de ponderação TF em 32,18%. Dessa forma, não é possível estabelecer a relação de que maiores valores de Acoplamento Conceitual são medidas mais adequadas para processos de recomendações de mudanças conjuntas.

O impacto dos parâmetros de pré-processamento de texto e fatores de ponderação ainda é um assunto pouco estudado quando relacionados à valores de Acoplamento Conceitual. Assim, esse estudo é necessário para que profissionais relacionados ao desenvolvimento de software possam utilizar o melhor conjunto de pré-processamentos e fatores de ponderação para a obtenção dos melhores valores de Acoplamento Conceitual. Desse modo, espera-se que os resultados deste estudo sejam utilizados para a redução de esforços e custos empregados para manutenção, correção e desenvolvimento de software.

Além deste Capítulo introdutório, esta monografia está organizada da seguinte forma: No Capítulo 2 estão descritos os conceitos fundamentais para o entendimento deste trabalho, bem como os trabalhos relacionados. No Capítulo 3 são apresentados o objetivo, as questões de pesquisa e o método empregado para respondê-las. O Capítulo 4 apresenta os resultados e discussões resultantes do estudo realizado e as ameaças à validade. Por último, o Capítulo 5 discute as conclusões desta monografia. Dados complementares deste estudo são encontrados nos apêndices.

Capítulo 2

Referencial Teórico

Neste capítulo são apresentados os fundamentos desta monografia, assim como os trabalhos relacionados. Na Seção 2.1 são expostos os conceitos fundamentais sobre pré-processamentos de arquivos de código-fonte e técnicas de fatores de ponderação de termos utilizados em modelos de cálculo de valores de Acoplamento Conceitual. Além disso, apresenta as definições de Acoplamento Conceitual e Regras de Associação utilizadas para realizar predições de mudanças conjuntas em arquivos de código-fonte. Na Seção 2.2, são apresentados os trabalhos relacionados ao estudo desta monografia e, por fim, a Seção 2.3 aborda as considerações finais a respeito deste capítulo.

2.1. Fundamentação Teórica

Esta monografia aborda um estudo a respeito do impacto das variações de pré-processamentos de texto de código-fonte na obtenção de valores de acoplamento conceitual. Foram encontrados trabalhos que apresentam variações do processo de obtenção de valores de Acoplamento Conceitual (KAGDI *et al.*, 2013; MARCUS, 2004; GARNIER; GARCIA, 2016; SAFEER *et al.*, 2010). No entanto, não foram encontrados estudos que comparem os impactos das combinações entre diferentes formas de pré-processamento de texto e fatores de ponderação nos valores de Acoplamento Conceitual.

2.1.1. Versionamento Semântico

Versionamento Semântico é um conjunto de princípios formulados em 2010 pelo fundador do `GitHub`, Tom Preston-Werner[1], para numerar univocamente as versões de software. De

[1] `http://semver.org`

acordo com esses princípios, a numeração de versões deve seguir o esquema de três dígitos — `MAJOR.MINOR.PATCH` — sendo que cada dígito possui o seguinte significado:

- `MAJOR`: Número que deve ser incrementado quando são realizadas mudanças de API
- `MINOR`: Número que deve ser incrementado quando uma nova funcionalidade é incrementada ao software, porém é mantida a compatibilidade com a versão anterior
- `PATCH`: Número que deve ser incrementado quando são realizadas as correções de possíveis falhas que ocorrem no desenvolvimento do software

O versionamento de software não é uma obrigatoriedade. No entanto, é uma prática incentivada na comunidade de desenvolvedores presentes em repositórios do `GitHub` (RAEMAEKERS *et al.*, 2014) para que cada versão de software possua uma numeração única. Em virtude deste fato, as versões de software utilizadas neste estudo foram selecionadas levando em consideração o conceito de Versionamento Semântico, com o intuito de selecionar apenas as versões definidas como principais (`MAJOR`). Essa prática visa escolher apenas versões que tenham mudanças de Acoplamento Conceitual entre si, evitando as versões que possuem apenas modificações de refatorações.

2.1.2. Recomendação de Mudança Conjunta

A realização de manutenção e desenvolvimento de software implicam em alterações de arquivos de código-fonte. Desse modo, é fundamental ter conhecimento dos possíveis arquivos de código-fonte afetados a cada modificação realizada. De acordo com o trabalho de (ZIMMERMANN *et al.*, 2005) é importante ter conhecimento das relações entre arquivos de código-fonte para manter a qualidade do software, pois isso ajuda na prevenção de falhas causadas por alterações incompletas e auxilia a manutenção necessária durante a evolução do software.

A Recomendação de Mudança Conjunta (do inglês, *co-change recomandation*) é utilizada por desenvolvedores para a predição dos arquivos de código-fonte mais suscetíveis a sofrerem modificações de forma conjunta à aqueles que são alterados durante o desenvolvimento. Para ilustrar, toma-se o seguinte exemplo: Imagine que um desenvolvedor altere o arquivo Y. Dentre os arquivos mais propensos a sofrerem modificação juntamente à esse arquivo estão os arquivos X e Z. Uma Recomendação de Mudança Conjunta ideal serial uma lista que indicasse esses dois arquivos como possíveis arquivos de código-fonte a serem modificados, economizando os esforços por parte dos desenvolvedores na busca pelos arquivos de código-fonte afetados pela modificação do arquivos Y.

2.1.3. Acoplamento Conceitual

Para que seja possível a criação de listas de Recomendação de Mudança Conjunta é necessário que sejam estabelecidas as relações de acoplamento entre arquivos de código-fonte (ISO, 2010). Em engenharia de software, acoplamento é o grau de interdependência entre arquivos de código-fonte, sendo uma medida que indica a força da relação entre eles. Quanto mais forte a relação de acoplamento entre arquivos, maior será a dependência existente entre eles (ISO; IEC, 2005).

Entre os possíveis tipos de acoplamento está o Acoplamento Conceitual (POSHYVANYK; MARCUS, 2006). Ele é obtido por meio da similaridade semântica entre arquivos de código-fonte. 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). Assim, retomando o exemplo ilustrado na Seção 2.1, um arquivo modificado Y teria os arquivos X e Z relacionados em uma lista de Recomendação de Mudança Conjunta caso estes sejam os arquivos que apresentem os maiores valores de Acoplamento Conceitual com o arquivo Y.

2.1.4. Pré-Processamentos

2.1.4.1. *Token* e *Tokenização*

Token pode ser definido como a menor parte individual do texto de um arquivo de código-fonte, podendo representar um dos possíveis tipos de elementos:

- **Identificador:** Identificadores são nomes dados a elementos de software, tais como variáveis, funções, métodos, classes e etc.
- **Palavra Reservada:** Palavra pertencente ao conjunto de instruções de uma linguagem. Exemplo: `for`, `if` e `while` na linguagem de programação `Java`.
- **Operador:** Qualquer símbolo de operações matemáticas ou lógicas.
- **Separador:** Qualquer elemento utilizado para separar elementos de software distintos. Exemplos de separadores são parenteses, colchetes, vírgula, ponto e dentre outros.
- **Literal:** São constantes que expressam valores. Eles podem ser de tipos como números (inteiros, ponto flutuante, notações científicas) ou palavras.

Todo trabalho que envolve a realização da obtenção de valores de Acoplamento Conceitual, realiza o processo de leitura dos arquivos de código-fonte e posteriormente a *Tokenização* (do inglês, *tokenization*) (KAGDI *et al.*, 2013; POSHYVANYK; MARCUS, 2006), processo que divide cada arquivo código-fonte em sequência de *token*. Dessa forma, cada arquivo de código-fonte pode ser associado à uma lista de *token*, sendo que cada *token* se refere à um termo presente no arquivo de código-fonte.

2.1.4.2. *CamelCase*

CamelCase é forma de escrever palavras por meio da composição de termos pela junção de duas ou mais palavras com o uso de letras sensíveis ao caso (ou *underscore*) (LUCIA *et al.*, 2007a; KAGDI *et al.*, 2013; POSHYVANYK; MARCUS, 2007). Para exemplificar, imagine que em um arquivo de código-fonte da linguagem `Java` seja encontrado um método cujo identificador é nomeado por `getCar`. Esse termo é composto pelo verbo `get` e pelo substantivo `Car` (ambos em inglês). Dessa forma o termo `getCar` que é um termo composto, escrito com o uso da forma de *CamelCase*.

2.1.4.3. *Stemming*

O procedimento que reduz as palavras flexionadas à sua base comum ou raíz é denominado *stemming* (LOVINS, 1968). Por exemplo, a palavra `Development` é reduzida por *stemming* ficando como `Develop`. É um pré-processamento aplicado em trabalhos de obtenção de valores de Acoplamento Conceitual entre arquivos de código-fonte (LUKINS *et al.*, 2008; POSHYVANYK, 2009; UJHAZI *et al.*, 2010).

2.1.4.4. Remoção de Quartil de Termos Menos Frequentes

Uma das tarefas durante o pré-processamento de arquivos de código-fonte para a obtenção de valores de acoplamento conceitual é a remoção de termos que representem conceitos menos relevantes (LUCIA *et al.*, 2007a; LINSTEAD *et al.*, 2007), pois eles aparecem com menor frequência nos arquivos de código-fonte e podem ser considerados como ruídos nesse processo. Uma possível abordagem é a remoção de quartis de termos menos frequentes. Um quartil é a quarta parte de uma amostra, sendo que uma amostra possui quatro quartis (HYNDMAN; FAN, 1996). Neste estudo foram realizadas remoções de termos menos frequentes em duas proporções: Remover o primeiro quartil, o que corresponde a remoção dos 25% termos menos frequentes; Remover o terceiro quartil, o que corresponde a remoção de 75% dos termos menos frequentes.

2.1.5. Fatores de Ponderação

A obtenção de valores de Acoplamento Conceitual entre arquivos de código-fonte exige uma etapa após a aplicação de pré-processamentos no texto que consiste em determinar valores para a relevância de cada termo presente em cada arquivo de um projeto. Isso implica na quantificação da frequência em que cada termo ocorre em cada arquivo de código-fonte e no projeto como um todo. Essas frequências de termos precisam ser normalizadas para verificar a relação de relevância de cada termo tanto para o arquivo de código-fonte em que ele aparece, como para o projeto de software como um todo. Dentre as fatores de ponderação utilizadas nos trabalhos relacionados ao Acoplamento Conceitual estão o TF-IDF (SUN *et al.*, 2010; ANTONIOL *et al.*, 2000; GARNIER; GARCIA, 2016; SAFEER *et al.*, 2010) e o LSI (KAGDI *et al.*, 2013; MARCUS, 2004; LUCIA *et al.*, 2007b).

2.1.5.1. Fator de Ponderação TF-IDF

O TF-IDF (acrônimo de *Term Frequency - Inverse Document Frequency*) é uma fator de ponderação 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 menor no projeto será o seu valor normalizado.

O TF-IDF é uma fator de ponderação que reflete o grau de importância de uma palavra de um arquivo que pertence a uma coleção de arquivos (GARNIER; GARCIA, 2016). O valor do TF-IDF cresce proporcionalmente ao número de aparições de um termo em um arquivo e é condicionado ao número de vezes em que aparece em todos os arquivos do corpus. Dessa forma, é proporcional ao número de vezes em que aparece no arquivo considerando todas as aparições no conjunto de arquivos.

O cálculo TF-IDF é dividido em duas partes, sendo a primeira o *term frequency*, ou frequência do termo, que é o peso da aparição do termo no arquivo e é calculada por meio da divisão do número de vezes em que o termo aparece no arquivo pelo número total de termos do arquivo:

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t \in d} ft', d} \tag{2.1}$$

A segunda parte do cálculo se refere ao grau de importância do termo em todos os arquivos do conjunto, é calculado pelo logaritmo do valor da divisão entre o número de

arquivos pelo número de arquivos que contém o termo:

$$idf(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{2.2}$$

Desse modo, multiplica-se o valor de TF pelo respectivo valor de IDF do termo. Com isso se tem o valor de TF-IDF para o termo, no aquivo:

$$tfidf(t, D) = tf(t, D) * idf(t, D) \tag{2.3}$$

2.1.5.2. Fator de Ponderação LSI

É um fator de ponderação baseado no princípio de 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.

Nesse tipo de fator de ponderação, é criada uma matriz contendo a frequência dos termos por documento (linhas representam um único termo e as colunas representam os documentos) e é aplicado sobre ela a técnica matemática conhecida como SVD (WALL *et al.*, 2003) (acrônimo do inglês, *Singular Value Decomposition*) que reduz o número de linhas ao passo que preserva as similaridades conceituais entre as colunas. Em seguida, as palavras são comparadas por meio do método do cosseno (KAGDI *et al.*, 2013; GARNIER; GARCIA, 2016; POSHYVANYK, 2009), sendo que cada vetor é composto por uma linha da tabela. Valores próximos a 1 representam termos muito similares, enquanto os próximos de 0 representam maiores diferenças.

2.1.5.3. Fator de Ponderação TF

O termo TF referencia o uso da frequência dos termos (do inglês, *Term Frequency*) para o cálculo dos valores de Acoplamento Conceitual entre arquivos de código-fonte sem a aplicação de qualquer processo de fator de ponderação. Dessa forma, são quantificados o número de vezes que cada termo é encontrado em cada arquivo e esses números são utilizados no cálculo sem a realização de qualquer outra consideração.

2.1.6. Cosseno

Cosseno é a métrica de similaridade entre dois vetores não nulos e denota o angulo entre eles. O cosseno de 0° é 1, valor máximo, e é menos do que 1 para todo o intervalo [0°,360°). No contexto de Acoplamento Conceitual, o valor do cosseno entre vetores que representam quantificações de termos de arquios de código-fonte é sempre positivo, assumindo valores entre 0 (nenhuma similaridade) e 1 (vetores idênticos, similaridade máxima). Esse cálculo é utilizado para mensurar o valor do Acoplamento Conceitual entre arquivos de código-fonte em trabalhos de engenharia de software (KAGDI *et al.*, 2013; GARNIER; GARCIA, 2016; POSHYVANYK, 2009).

Dado dois vetores de termos, A e B, a similaridade calculada por cosseno, $cos(\theta)$, é representado por:

$$similaridade = cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}||||\mathbf{B}||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2.4}$$

2.1.7. Modelos de Cálculo do Acoplamento Conceitual

Neste trabalho, o termo **Modelo** é usado para referenciar a uma possível forma de calcular os valores de Acoplamento Conceitual entre arquivos de código-fonte. Dessa forma, cada combinação de parâmetros de pré-processamento textual e fator de ponderação de termos é denominada **Modelo de Cálculo dos Valores de Acoplamento Conceitual**. Por exemplo, podemos afirmar que dentre os possíveis modelos de cálculo de valores do Acoplamento Conceitual entre arquivos de código-fonte há um que é formado pela combinação da fator de ponderação TF-IDF, separação de termos por *CamelCase*, *stemming* e remoção do terceiro quartil de termos menos frequentes.

2.1.8. Avaliação de Recomendações de Mudanças Conjuntas

A eficiência de recomendações de mudanças conjuntas pode ser avaliada em virtude das medidas de *Precisão* e *Sensibilidade* (*Recall*) (MARCUS; MALETIC, 2003; MARCUS *et al.*, 2008; KAGDI *et al.*, 2010). A *Precisão* é o resultado da divisão do número de recomendações corretas e o total de elementos recomendados. A *Sensibilidade* é o valor da razão do número de recomendações corretas pelo total de elementos esperados.

Sendo F o número de elementos recomendados (recuperados) e E o número de elementos recomandados corretamante, o cálculo da *Precisão* é definido por:

$$Precisão = \frac{|F \cap E|}{|F|} \tag{2.5}$$

A *Sensibilidade* é definida pela equação:

$$Sensibilidade = \frac{|F \cap E|}{|E|} \tag{2.6}$$

Além da Precisão e da Sensibilidade, a medida F também está relacionada à qualidade de recomendações de mudanças (KAGDI *et al.*, 2013). Ela é calculada por meio dessas duas medidas, sendo o seu cálculo definido por:

$$\text{MEDIDA F} = 2 \times \left(\frac{Precisão \times Sensibilidade}{Precisão + Sensibilidade} \right) \tag{2.7}$$

Em (GARNIER; GARCIA, 2016), é mostrado o uso da Precisão Média (*Average Precision*) (AP), uma métrica indicada para a avaliar recomendações. O cálculo considera a ordem dos arquivos da lista de recomendação. Desse modo, quanto mais próximo da primeira colocação da lista estiver uma recomendação correta, melhor será a qualidade da recomendação.

Na equação que calcula o AP, *P(k)* é a precisão dos primeiros k arquivos de código-fonte da lista de recomendações e $\Delta r(k)$ é a diferença da *Sensibilidade* calculada em $k-1$ primeiros arquivos e a dos k primeiros arquivos. F é a lista ordenada de recomendações. Os valores da Precisão média variam de 0 a 1. A equação é definida por:

$$AP = \sum_{k=1}^{|F|} P(k) * \Delta r(k) \tag{2.8}$$

A medida da eficiência global é a Média das Precisões Médias (do inglês, *Mean Average Precision*) MAP (GARNIER; GARCIA, 2016), é calculada sobre o conjunto de todas a predições realizadas. Esse cálculo é a média de todos os AP calculados. Para uma consulta Q, o calculo do MAP é:

$$MAP = \frac{1}{|Q|} \sum_{q \in Q} AP(q) \tag{2.9}$$

O MAP é uma medida que avalia a qualidade da recomendações de mudanças. Predições que aparecem primeiro nas listas de predições são consideradas melhores (GARNIER; GARCIA, 2016), pois ele leva em consideração a posição na lista de recomendação em que o arquivo recomendado aparece. Os valores de MAP variam de 0 a 1.

2.2. Trabalhos Relacionados

Nesta seção, são apresentados os principais trabalhos relacionados ao calculo de valores de Acoplamento Conceitual. Eles foram divididos em duas categorias principais, de acordo com a fator de ponderação empregada em cada trabalho, sendo assim divididos em TF-IDF e LSI.

2.2.1. O uso do TF-IDF

O trabalho de (GARNIER; GARCIA, 2016) utiliza valores de Acoplamento Conceitual para localização de falhas em código-fonte. Inicialmente, os autores selecionaram 20 projetos desenvolvidos na linguagem de programação `C#`. Eles usam os textos de *issues* presentes em repositórios de *issues* utilizados pelos desenvolvedores desses projetos para mapear os arquivos que contenham as falhas de acordo com a similaridade entre o texto da *issue* e texto do arquivo de código-fonte. Dessa forma, quanto maior o valor do Acoplamento Conceitual entre uma *issue* e um arquivo de código-fonte, maiores são as chances do erro reportado por aquela *issue* estar relacionado ao arquivo de código-fonte.

Por sua vez, (ANTONIOL *et al.*, 2000) usa o Acoplamento Conceitual como um meio de mapear as relações entre a documentação e os arquivos de código-fonte. Quanto maior o valor do Acoplamento Conceitual entre a documentação e o arquivo de código-fonte, maiores as chances deles estarem relacionados entre si. Uma peculiaridade deste trabalho é que, ao invés da aplicação da *stemming* como ocorre em outros trabalhos, neste os autores decidem apenas em converter palavras do plural para suas formas no singular.

Em (SAFEER *et al.*, 2010), os autores propõem a utilização do Acoplamento Conceitual como métrica para realizar a *clusterização* de arquivos de código-fonte. Diferentemente da organização estrutural, como a vista em `Java` por exemplo (pacotes, classes e afins), essa organização consiste em agrupar conjuntos de arquivos mais similares entre si de acordo com o Acoplamento Conceitual. Dessa forma, arquivos de código-fonte com altos valores de Acoplamento Conceitual entre si tendem a fazerem parte do mesmo módulo dentro da organização proposta.

2.2.2. O uso do LSI

Os trabalhos de (KAGDI *et al.*, 2013), (MARCUS, 2004) e (LUCIA *et al.*, 2007b) utilizaram o modelo de fator de ponderaçãoLSI e também diferentes pré-processamentos. Novamente, os trabalhos usaram o modelo de fator de ponderação para propósitos específicos, enquanto

(KAGDI *et al.*, 2013) e (MARCUS, 2004) usaram os seus modelos para prever mudanças conjuntas, (LUCIA *et al.*, 2007b) construiu uma ferramenta para rastrear artefatos.

Em (KAGDI *et al.*, 2013) os autores propõem a combinação do cálculo de valores de Acoplamento Conceitual com outra métrica denominada Acoplamento Evolucionário. Este tipo de acoplamento considera que arquivos que mudam conjuntamente com certa frequência estão relacionados entre si. Dessa forma, o autor combina os dois tipos de acoplamento para a realização de predições de mudanças conjuntas. Nesse trabalho, os autores conseguiram melhores resultados de predições de mudanças em relação ao uso exclusivo do Acoplamento Evolucionário, tendo análises que chegaram 21% melhor quando a granularidade do estudo era método de classes java (análise aplicada no código-fonte do projeto `ArgoUML`) e de 9% a 30% melhor quando a granularidade do estudo é a de arquivo de código-fonte (análise realizada no projeto `iBatis`).

O trabalho apresentado por (MARCUS, 2004) propõem o uso do Acoplamento Conceitual combinado com o Acoplamento Estrutural. Dessa forma, além de usar a similaridade de conceito entre arquivos de código-fonte, o fator da estrutura é proposto para ser levado em consideração. Suponha um projeto `Java`, estruturado em pacotes e classes. Dois arquivos são mais similares entre si se pertencerem ao mesmo pacote e possuírem um alto valor de Acoplamento Conceitual. Já ao contrário dos demais, o trabalho proposto por (LUCIA *et al.*, 2007b) utiliza exclusivamente valores de Acoplamento Conceitual para rastrear artefatos de código-fonte.

2.3. Considerações Finais

Este capítulo apresentou os conceitos fundamentais para o entendimento do trabalho desta monografia. Diferentemente dos trabalhos relacionados que propõem o uso do Acoplamento Conceitual como forma de melhoria em algum propósito específico, o principal objetivo deste trabalho consiste no estudo empírico do cálculo de Acoplamento Conceitual. Todos os conceitos apresentados fazem parte do método criado para este estudo.

Capítulo 3

Metodologia

Este capítulo descreve o método utilizado neste trabalho. Na Seção 3.1 é descrito o objetivo, bem como o problema relacionado ao assunto desta pesquisa. Na Seção 3.2 estão descritos os passos que compõem o método deste estudo. Na Seção 3.3 são apresentadas as questões de pesquisas. Por fim, na Seção 3.4 são realizadas as considerações finais a respeito da metodologia.

3.1. Objetivo

O objetivo deste trabalho consiste em comparar como os pré-processamentos e fatores de ponderação combinados em diferentes modelos impactam os valores de Acoplamento Conceitual obtidos, verificando se diferentes modelos de obtenção resultam em valores distintos de Acoplamento Conceitual.

Esse estudo se faz necessário pela ausência de trabalhos que comparem os diferentes modelos de obtenção de valores de Acoplamento Conceitual utilizados em diversos estudos presentes na literatura (KAGDI *et al.*, 2013; BAVOTA *et al.*, 2010, 2011). Em virtude do fato de que o Acoplamento Conceitual é uma métrica relevante em cenários como o de recomendações de mudanças conjuntas entre artefatos (KAGDI *et al.*, 2013), recomendação de refatorações (BAVOTA *et al.*, 2011), melhorar modularidade (BAVOTA *et al.*, 2010) e uso em técnicas para predizer a ocorrência de defeitos (KAGDI *et al.*, 2013), é importante que seja feito um estudo que verifique se há diferenças entre os valores de Acoplamento Conceitual obtidos por meio de diferentes modelos de obtenção e, se possível, que auxilie na escolha da melhor combinação de pré-processamentos e fatores de ponderação para a obtenção de valores de Acoplamento Conceitual.

Além da constatação da diferença entre as distribuições de valores de Acoplamento Conceitual calculadas por modelos distintos, foi necessário a realização de uma avaliação desses valores de forma empírica em predições de mudanças conjuntas. Isso foi feito para constatar se as distribuições com maiores valores de Acoplamento Conceitual resultam em predições de mudanças conjuntas mais eficientes, buscando os melhores valores de Acoplamento Conceitual para a utilização em tarefas de engenharia de software.

3.2. Método

Nesta Seção são apresentados os passos do método realizado para a obtenção dos valores de Acoplamento Conceitual que são utilizados para avaliar e responder as questões de pesquisa. A Figura 3.1 mostra uma visão geral do processo.

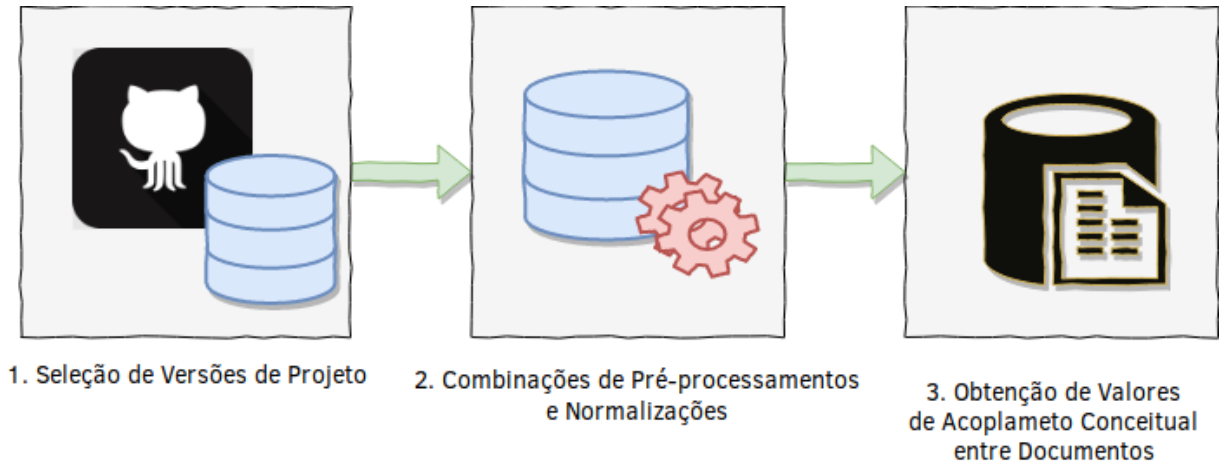


Figura 3.1. Visão Geral do método empregado neste estudo

3.2.1. Coleta das Versões de Projetos

A primeira etapa de execução do estudo desta monografia é a coleta das versões de um conjunto composto por 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 elas apresentam pouca variação de código, logo as similaridades entre os arquivos computados pela métrica de acoplamento conceitual devem ser irrelevantes. De cada uma das versões dos projetos selecionados foram coletados o código fonte usando o comando `git clone (url do projeto)`.

A 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`, `React` e `Angular`. As informações a respeito das versões de projetos selecionadas para este estudo estão disponíveis, resumidamente, nos apêndices. Para informações mais detalhadas, como número de *forks*, estrelas e data de último acesso aos repositórios dos projetos estão disponíveis em armazenamento remoto e podem ser encontrados em (COSTA, 2018a)[1].

As Tabelas 3.1, 3.2, 3.3, 3.4 e 3.5 contém dados referentes aos projetos selecionados para o estudo realizado. Nelas, a primeira coluna da esquerda pra direita estão os nomes dos projetos. A coluna **NV** contém o número de versões, **NARQ** o número de arquivos por versão, **NLC** o número de linhas de código-fonte por versão, **CPV** o número de *commit* por versão, **ACPV** o número médio de arquivos modificados entre versões, **ACMIN** e **ACMAX** são respectivamente o número mínimo e máximo de arquivos alterados entre versões de software. A Tabela 3.1 contém dados de projetos `C++`, a Tabela 3.2 projetos `Java`, a Tabela 3.3 projetos `Javascript`, a Tabela 3.4 projetos `Python` e a Tabela 3.5 projetos `Ruby`.

Tabela 3.1. Dados de Projetos `C++` Selecionados para o Estudo

PROJETO	NV	NARQ	NLC	CPV	ACPV	ACMIN	ACMAX
`electron`	5	533.2	73960.0	1144.75	3942.0	701	5451
`flatbuffers`	5	46.0	23771.8	94.75	429.75	188	628
`gource`	5	58.4	17469.4	70.0	188.25	18	425
`leveldb`	5	123.2	25449.2	16.0	62.75	10	160
`liteide`	5	678.4	123110.6	140.75	678.5	347	830
`mosh`	4	88.0	14881.25	269.0	631.66	153	1313
`muduo`	5	236.2	22635.6	48.0	167.0	30	514
`osquery`	5	463.4	88912.8	64.25	231.75	187	321
`watchman`	5	137.6	35146.8	242.25	2294.5	66	7974
`zeal`	5	59.2	8429.6	150.5	910.0	94	1975

[1] link para acessar informações dos projetos coletados: `https://doi.org/10.5281/zenodo.1495566`

Tabela 3.2. Dados de Projetos `Java` Selecionados para o Estudo

PROJETO	NV	NARQ	NLC	CPV	ACPV	ACMIN	ACMAX
agera	5	123.6	16902.4	49.0	181.66	105	299
Dexter	4	30.25	2346.25	81.0	221.66	41	419
epoxy	5	349.8	41843.0	13.0	154.25	99	220
fresco	5	760.0	105917.6	132.25	1437.75	294	2107
guice	3	478.0	72118.33	399.0	17053.5	15571	18536
moshi	5	59.6	14380.8	46.0	306.75	176	581
okhttp	5	279.6	73847.6	68.5	516.75	216	721
okio	5	48.8	11551.0	31.0	174.75	144	248
picasso	5	67.6	9961.0	88.0	378.25	206	549
stetho	5	219.6	22482.0	96.0	426.0	173	936
tinker	4	218.25	43496.0	75.66	359.33	150	764

Tabela 3.3. Dados de Projetos `Javascript` Selecionados para o Estudo

PROJETO	NV	NARQ	NLC	CPV	ACPV	ACMIN	ACMAX
angular	5	298.4	17844.4	764.5	5318.25	2227	10477
async	5	39.8	10488.4	189.75	1262.25	121	4629
bluebird	5	212.0	50887.2	97.25	732.25	307	1224
cube	5	40.2	16459.4	47.75	207.75	62	452
enzyme	6	33.0	4397.16	83.25	429.0	98	924
hexo	5	261.8	21967.8	87.0	353.75	94	691
karma	5	139.6	15845.4	37.5	104.5	22	268
mocha	5	179.6	27568.6	143.0	449.0	206	890
nodemon	5	147.6	17510.8	29.0	55.25	17	130
react	5	399.0	85736.0	2146.25	14434.75	5603	30841
redux	5	145.6	6905.6	315.0	923.25	372	2015
vue	5	331.4	110451.2	238.5	689.0	506	1041

Tabela 3.4. Dados de Projetos `Python` Selecionados para o Estudo

PROJETO	NV	NARQ	NLC	CPV	ACPV	ACMIN	ACMAX
bokeh	5	547.6	55440.2	1678.75	17313.25	8746	24961
boto	5	461.4	115878.6	612.25	2779.5	1530	5213
bypy	3	22.0	5997.0	31.5	82.0	43	121
celery	5	252.6	50696.6	1667.75	7271.5	933	11139
certbot	5	226.2	53509.6	144.5	770.0	359	1440
chainer	5	531.6	93775.0	848.25	6978.25	2877	14221
chisel	5	19.2	3512.6	39.5	114.5	38	165
compose	5	67.6	19129.6	161.75	537.25	189	970
flask	5	56.6	11858.0	426.25	981.75	428	2229
ipython	5	433.6	90819.6	2864.25	8085.25	1922	17628
keras	5	122.2	37809.6	505.5	1247.25	547	1831
pandas	5	499.8	314335.6	770.5	3995.75	2615	7064
requests	5	58.4	19365.4	68.0	303.5	195	450
scrapy	5	312.8	33447.6	278.25	529.25	278	773
sentry	5	428.6	60967.8	2971.75	13272.0	2155	23763
sympy	5	641.6	274277.0	7078.0	43862.75	4389	122297

3.2.2. Modelos 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 3.2 apresenta o fluxo inicial da construção

Tabela 3.5. Dados de Projetos `Ruby` Selecionados para o Estudo

PROJETO	NV	NARQ	NLC	CPV	ACPV	ACMIN	ACMAX
`bourbon`	5	27.4	843.4	163.0	647.5	128	1495
`capistrano`	5	85.0	6492.4	54.0	214.25	83	523
`celluloid`	5	88.4	6606.2	282.75	896.75	332	2200
`CocoaPods`	5	183.8	26788.0	1465.75	6032.75	335	21699
`devise`	5	188.2	16025.2	68.5	206.75	130	286
`diaspora`	5	683.4	43484.6	1265.25	9958.75	3370	15518
`haml`	5	49.8	11152.6	1239.5	3513.25	1270	6890
`kaminari`	5	51.4	3571.2	233.0	475.5	178	770
`puma`	5	80.8	10475.8	37.5	93.25	54	131
`sass`	5	148.0	36673.0	591.75	2331.25	1198	3096
`sinatra`	5	62.8	11131.8	815.5	1639.0	512	2568
`thor`	5	64.8	10138.2	116.0	338.25	153	560

dos modelos de obtenção de valores de Acoplamento Conceitual. Nela é mostrado que após a coleta de projetos, são lidos os arquivos de código-fonte de cada versão de software. Em seguida, o código-fonte passa pelo processo de *tokenização*. O conjunto de *tokens* resultante passa por uma remoção de símbolos, números e palavras reservadas. Por último nesta etapa, 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 (LAWRIE *et al.*, 2010; LAWRIE; BINKLEY, 2011). Por exemplo, um identificador de variável com o nome representado pelo *token* 'a' não remete a nenhum conceito, portanto é eliminado nesta etapa.

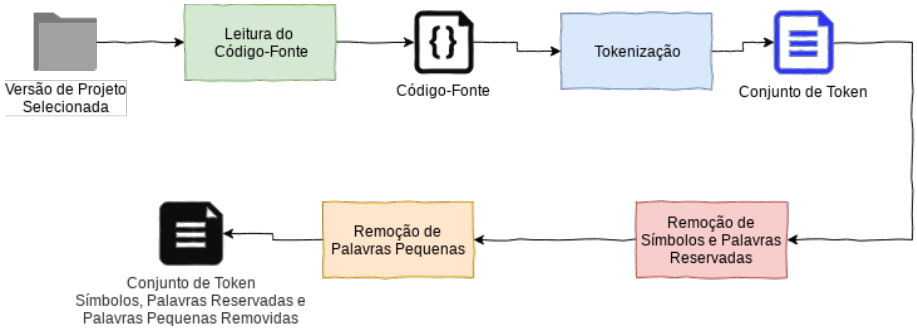


Figura 3.2. Método: Remoção de Símbolos, Caracteres Especiais, Números e Palavras Pequenas

Na Figura 3.3 é apresentado o fluxo de construção dos modelos que são comparados neste trabalho. De acordo com o que foi visto na seção de conceitos, especificamente a Seção 2.1 do Capítulo 2, foram comparados combinações de pré-processamento (usar ou descartar comentário de código-fonte, dividir ou não termos em virtude de *CamelCase*, aplicar ou não o *stemming*, remoção do primeiro ou terceiro quartil de termos menos frequentes em arquivos de código-fonte).

A primeira variação de pré-processamento é a consideração do uso de comentários de código-fonte. Em seguida, a divisão ou não dos termos em detrimento de *CamelCase*. A partir disso, considera-se a aplicação de *stemming* para reduzir termos às suas raízes. Neste

caso, a heurística utilizada para o *stemming* foi a de Porter (WILLETT, 2006). Após a aplicação desses pré-processamentos, os termos são quantificados de acordo com a frequência que ocorrem nos arquivos de código-fonte. Desse modo o resultado dessa etapa do método são listas de termos presentes no código-fonte quantificados.

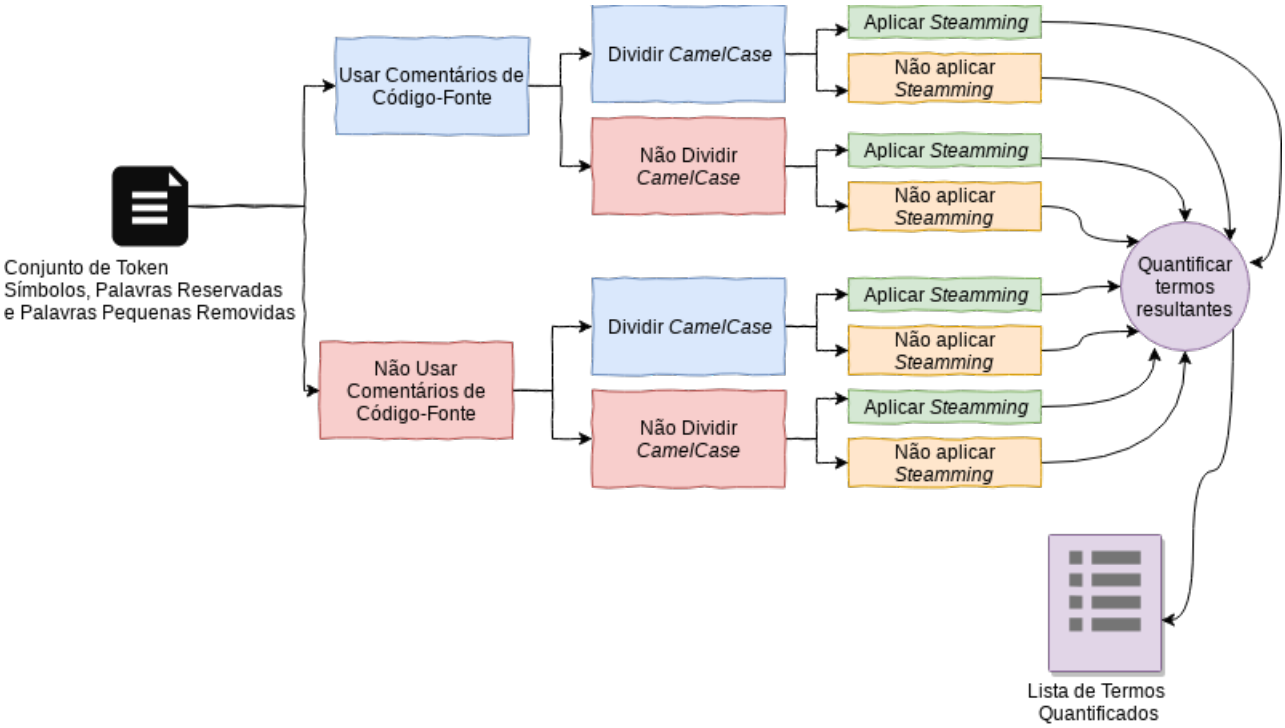


Figura 3.3. Método: Fluxo de Variações de Combinações de Pré-Processamentos

De acordo com a Figura 3.4, a lista de termos quantificados ainda sofrem a ação de pré-processamento. São feitas as considerações a respeito da remoção de termos menos frequentes, sendo removidos o primeiro ou o terceiro quartil de termos menos frequentes. A remoção do primeiro quartil significa que 75% dos termos quantificados são utilizados para o cálculo de valores de Acoplamento Conceitual, enquanto a remoção do terceiro quartil implica que apenas 25% dos termos quantificados são utilizados. Ao término desta etapa, foram aplicados todo o conjunto de pré-processamentos dos modelos de cálculo de valores de Acoplamento Conceitual.

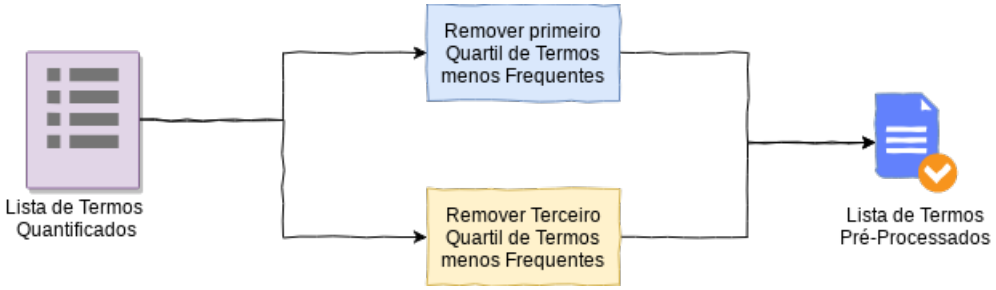


Figura 3.4. Método: Pré-processamento, Aplicação de remoção de termos menos frequentes

A última etapa do cálculo de valores de Acoplamento Conceitual consiste em escolher

um dos fatores de ponderação a serem aplicados nas listas de termos pré-processados. Cada lista é referente a um arquivo de código-fonte. O resultado da aplicação de um fator de ponderação é uma matriz onde as linhas e colunas são as combinações par-a-par entre todos os arquivos de código-fonte de uma versão de software com os seus respectivos valores de Acoplamento Conceitual.

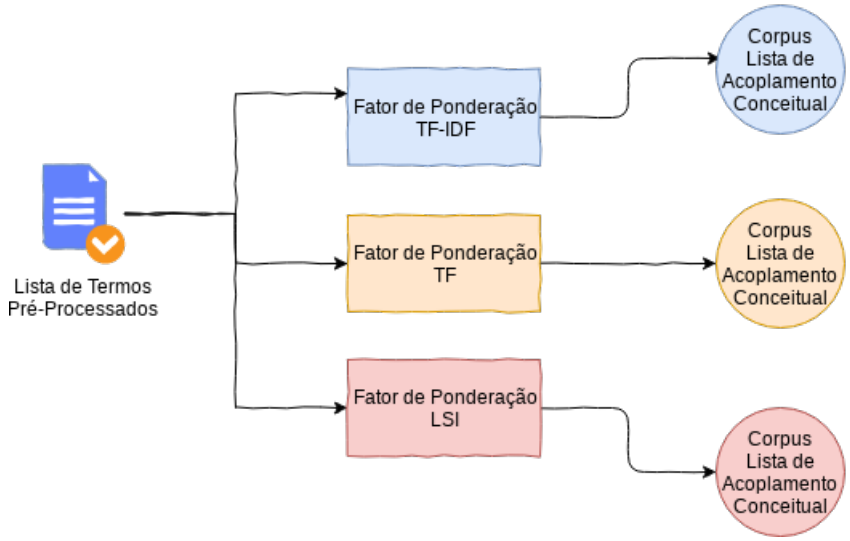


Figura 3.5. Método: Aplicação de Fator de Ponderação dos Termos

Apesar de haverem 16 variações de conjuntos de pré-processamento, para este estudo foram escolhidos 6 configurações apresentadas na Tabela 3.6. Essas configurações foram escolhidas a partir da literatura. Não foram utilizadas todas as combinações possíveis em virtude da grande quantidade de processamento necessário para a realização das operações sobre os arquivos de texto e das limitações de armazenamento dos resultados dos valores de acoplamento conceitual (o processo de obtenção dos valores gerou uma grande quantidade de dados, exigindo grande volume de armazenamento de dados).

De acordo com o observado a partir do conjunto de dados originados deste estudo, o volume de dados compactados é corresponde a aproximada 6 Gigabytes. Estes mesmos dados descompactados, totalizam um número próximo de 40 Gigabytes de dados. No que corresponde ao tempo utilizado para processamento dos dados por meio da aplicação dos modelos de cálculo de valores de Acoplamento Conceitual, foram cerca de 96 horas para a execução de todos os cálculos de valores de Acoplamento Conceitual para as 299 versões dos 61 projetos de software coletados, considerando apenas os modelos utilizados para este estudo.

Para que fosse possível realizar os cálculos de Acoplamento Conceitual, foi desenvolvido uma ferramenta em `Java` que a partir de uma entrada de parâmetros que corresponde às opções de pré-processamento e fator de ponderação, retorna as listas de valores de Acoplamento Conceitual entre arquivos de uma versão de software. Essa ferramenta recebeu o nome de

`simitrieve` (COSTA, 2018b)[2] (nome derivado da junção das palavras, em inglês, *similarity* e *retrieve*) e ela permite a execução de cálculos de valores de Acoplamento Conceitual para qualquer uma das variações de modelo apresentadas neste trabalho, gerando arquivos com extensão `.csv` como resultado.

Tabela 3.6. Combinações de Pré-processamento utilizados neste estudo

Utilizar Comentário	**Dividir por *CamelCase***	*stemming*	**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

3.3. Questões de Pesquisa

Este trabalho propõem uma análise para verificar se o modelo de cálculo do Acoplamento Conceitual impacta os valores de Acoplamento Conceitual. Para isso, duas questões de pesquisa foram avaliadas de forma empírica. A seguir, as questões de pesquisa são descritas e são apresentadas,resumidamente, as abordagens utilizadas para responder cada uma delas.

QP1: A variação dos parâmetros de pré-processamento e fator de ponderação dos modelos de cálculo resultam em diferentes valores de Acoplamento Conceitual?

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 $\alpha = 0.05$, 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, foi 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, foi usada 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. Dessa forma foi possível

[2] link para acessar a ferramenta `simitrieve`: `https://doi.org/10.5281/zenodo.1492898`

determinar a diferença entre os valores das amostras e assim definir os modelos de obtenção de Acoplamento Conceitual que resultam nos maiores valores por versão de projeto.

QP2: Os modelos de obtenção que resultam no cálculo dos maiores valores de Acoplamento Conceitual são os que proporcionam melhores predições de mudanças conjuntas?

Após verificar a diferença dos valores de Acoplamento Conceitual obtidos por meio de diferentes modelos, foi investigado se maiores valores implicam em melhores predições de mudanças conjuntas. Para isso foi necessário a realização de predições com base nas modificações entre as versões de software selecionadas para este estudo, em virtude dos valores de Acoplamento Conceitual obtidos na primeira questão de pesquisa. A escolha dos modelos de cálculo de Acoplamento Conceitual utilizados nesta segunda questão de pesquisa é referente aos modelos que para cada fator de ponderação apresentaram as distribuições com os maiores valores de Acoplamento Conceitual por linguagem, segundo o resultado apresentado para a primeira questão de pesquisa.

A primeira etapa da segunda questão de pesquisa consistiu em obter as modificações realizadas a cada *commit* entre duas versões *major* consecutivas para cada software selecionado para o estudo. O conjunto de *commits* utilizado compreende aqueles em que há pelo menos dois arquivos modificados. Além disso, foram descartados todo *commit* com número de arquivos modificados maior do que 10, pois um número elevado de arquivos modificados em apenas um *commit* pode indicar uma refatoração, assim sem alterações de Acoplamento Conceitual.

Em seguida, foram selecionados os modelos de cálculo de Acoplamento Conceitual que apresentaram maiores valores por linguagem, projeto e versão para que fossem utilizados na realização das predições de mudanças conjuntas. As predições foram realizadas para cada arquivo modificado em cada *commit*, sendo que o tamanho das listas de arquivos recomendados para a predição variou entre o número de arquivos que sofreram alterações em cada *commit* e o terceiro quartil do número de arquivos modificados por *commit* entre duas versões de software.

Para cada arquivo modificado em cada *commit* foi realizada uma predição de possíveis arquivos que mudam conjuntamente. Dessa forma para cada arquivo foi associado um valor AP (precisão média, do inglês *Average Precision*). Por meio do valor de AP de cada recomendação, foi calculado o MAP (Meio Termo da Precisão Média, do inglês *Mean Average Precision*) para cada *commit*, o que corresponde à eficiência da predição de mudanças conjuntas para cada *commit* entre duas versões de software.

Após verificar se existe diferença entre as distribuições dos valores de MAP, foi aplicado o teste *Cohen's D* (tamanho de efeito) para determinar o tamanho da diferença dos desempenhos

das predições (mesmo teste estatístico aplicado na primeira questão de pesquisa). Dessa forma foi possível determinar em quais casos um modelo de cálculo de Acoplamento Conceitual foi mais eficaz em suas recomendações de mudanças conjuntas e se há relação entre maiores valores de Acoplamento Conceitual e melhores predições.

Para a realização das predições de mudanças conjuntas é necessário saber quais os arquivos foram modificados nos *commits* entre as versões de software. Para isso, foi utilizado uma ferramenta desenvolvida em `Java` que se chama `Repodriller`[3]. Por meio dela foi possível obter as informações dos arquivos de código-fonte que foram modificados a cada *commit*. Desse modo, é possível saber quando dois ou mais arquivos mudam conjuntamente.

Para exemplificar o processo de predição de mudanças conjuntas realizado neste estudo, considere duas versões *major* do software `React`, 15 e 16. Por meio da ferramenta `Repodriller`, são obtidas as informações dos arquivos modificados entre as duas versões de software a cada *commit*.

Em seguida, são calculadas os valores de Acoplamento Conceitual entre os arquivos de código-fonte a partir da versão 15 do `React`. Suponha que o arquivo `A` foi modificado em um *commit*. A partir disso, é feito uma consulta à lista de valores de Acoplamento Conceitual para obter os arquivos que tem maiores valores de Acoplamento Conceitual em relação ao arquivo `A`. Caso a lista de arquivos com maiores valores de Acoplamento Conceitual indique um arquivo que foi modificado conjuntamente ao arquivo `A` no *commit* significa que ocorreu uma recomendação de mudança conjunta correta.

É preciso ressaltar que o tamanho das listas de predições é igual ao terceiro quartil do número de arquivos modificados entre *commits* entre duas versões *major* de software. Além disso, as predições de mudanças conjuntas foram feitas para todos os arquivos que apareceram nas listas de `commit` (arquivos de código-fonte, para cada linguagem).

3.4. Considerações Finais

Este capítulo apresentou os passos para a construção dos modelos de obtenção de valores de Acoplamento Conceitual utilizados no estudo realizado nesta monografia. Uma ferramenta que realiza aplicação dos parâmetros de pré-processamentos e fatores de ponderação foi implementada na linguagem de programação `Java` e está disponível no `GitHub`[4] e foi publicado no Zenodo[5]. A partir dessa ferramenta e dos modelos de cálculo de valores de Acoplamento Conceitual, foram realizadas as análises necessárias para determinar a influência das diferentes

[3] `https://github.com/mauricioaniche/repodriller`
[4] `simitrieve` no `GitHub`: `https://github.com/costabatista/simitrieve`
[5] `simitrieve` no Zenodo: `https://doi.org/10.5281/zenodo.1492898`

combinações de pré-processamentos e fatores de ponderação para a determinação dos valores de Acoplamento Conceitual entre arquivos de código-fonte, bem como, os impactos em predições de mudanças conjuntas entre as versões de software coletadas para este estudo.

Capítulo 4

Resultados e Discussão

Neste capítulo são apresentados os resultados e discussões dos experimentos realizados para responder as questões de pesquisa. A análise dos resultados está divida em duas partes, sendo que a primeira é dedicada a verificar se diferentes modelos de cálculo de Acoplamento Conceitual resultam em valores distintos de Acoplamento Conceitual. A segunda parte da análise investiga se os modelos de cálculo que resultam em valores de Acoplamento Conceitual maiores geram as melhores recomendações de mudanças conjuntas. Os resultados e discussões das questões de pesquisa investigadas neste trabalho são apresentados a seguir.

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

Para responder a primeira questão de pesquisa, foram realizadas 45747 comparações entre diferentes modelos de cálculo de valores de Acoplamento Conceitual para responder as questões de pesquisa. Esse número de comparações resulta da comparação par a par de 18 modelos de cálculo de acoplamento conceitual para as 299 versões de software. Para cada versão foram realizadas 153 comparações.

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, de acordo com o capítulo 3. A partir disso, as fatores de ponderação foram fixadas, variando somente os pré-processamentos listados na Tabela 3.6. Não foram realizadas variações quanto a ordem da aplicação dos pré-processamentos do texto do código-fonte.

De acordo com a Tabela 4.1, identificou-se para as linguagens `Javascript`, `Python` e `Ruby` que o conjunto de pré-processamentos que resulta em maiores valores de Acoplamento Conceitual é: usar comentário de código-fonte, dividir termos por *CamelCase*, não aplicar *stemming*, 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 fatores de ponderaçãoTF-IDF e TF.

No entanto, o mesmo não ocorreu para as linguagens `C++` e `Java` com o uso do LSI. Neste caso, o conjunto de pré-processamento que resulta em maiores valores de Acoplamento Conceitual é formado pelo uso de comentário de código, não divisão de termos por *CamelCase*, não aplicar *stemming* nos termos 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 4.1. Pré-processamentos que resulta em maiores valores de Acoplamento Conceitual para cada fator de ponderação por linguagem

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

É importante ressaltar o que há de comum entre os conjuntos de pré-processamentos que resultam nos maiores valores de Acoplamento Conceitual. Para todas as linguagens de programação e fatores de ponderação o uso de comentário de código-fonte eleva os valores das distribuições de Acoplamento Conceitual. O mesmo ocorre com a não aplicação de *stemming*. Dessa forma. e possível afirmar que esses pré-processamentos tendem a aumentar os valores de Acoplamento Conceitual.

Para os fatores de ponderação TF-IDF e TF, constata-se que remover o primeiro quartil de termos menos frequentes resulta em maiores valores de Acoplamento Conceitual. O mesmo não ocorre para o fator de ponderação LSI para as linguagens `C++` e `Java`, pois os maiores valores são obtidos por meio da remoção do terceiro quartil de termos menos frequentes.

Ao comparar os modelos com fator de ponderaçãoLSI 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 fator de ponderaçãoLSI retorna valores muito semelhantes de acoplamento conceitual, independentemente da linguagem de

programação analisada. O mesmo não aconteceu com os modelos TF-IDF e TF. No caso do TF-IDF, 60,28% das comparações retornaram um tamanho de efeito muito pequeno. No modelo TF, 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 fator de ponderação usada é o TF-IDF e o modelo TF. Nota-se que para as linguagens `C++` e `Java` os conjuntos dos pré-processamentos que resultam em maiores valores de Acoplamento Conceitual em virtude das fatores de ponderação 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`.

Além disso, observou-se que apesar da diferença entre os valores de Acoplamento Conceitual ser, na maioria dos casos, uma diferença com tamanho pequeno e muito pequeno, esse fato não pode ser ignorado. Isso pois, não se sabe nada a respeito do impacto dessas diferenças em processos que dependem do acoplamento conceitual como parâmetro de entrada, como por exemplo, a predição de mudanças conjuntas em projetos de software.

Desse modo, mesmo que a diferença entre as distribuições de valores de Acoplamento Conceitual sejam pequenas, há modelos que promovem valores maiores. Afim de se estudar os pré-processamentos e fatores de ponderação que causam o aumento desses valores, a primeira questão de pesquisa sofreu um desdobramento. O direcionamento desta questão de pesquisa é conferido a seguir.

4.2. É possível determinar o modelo (pré-processamento + técnica de fator de ponderação) que resulta os maiores valores de Acoplamento Conceitual?

Após a realização das comparações entre modelos de cálculo de Acoplamento Conceitual que usam a mesma fator de ponderação, foram comparados todos os modelos construídos entre si em cada versão de cada linguagem. De acordo com a Tabela 4.2, pode-se observar que as linguagens `C++` e `Java` possuem combinação de pré-processamento e fator de ponderação que resultam maiores valores de Acoplamento Conceitual em comum. O mesmo pode ser dito entre as linguagens `Javascript`, `Python` e `Ruby`. Em todas as linguagens os modelos baseados em LSI obtiveram os maiores valores de Acoplamento Conceitual. Do total de comparações, os modelos que resultaram nos maiores valores de Acoplamento Conceitual 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.

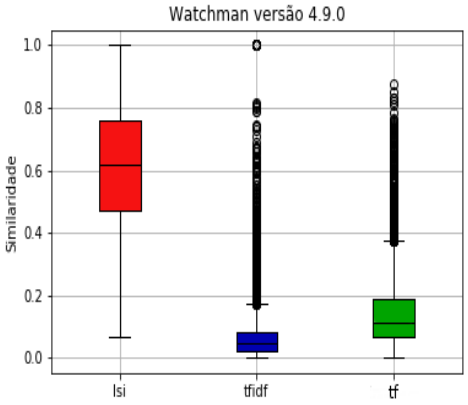
Tabela 4.2. Combinações de pré-processamento e fator de ponderação que resultam em maiores valores de Acoplamento Conceitual por linguagem

Linguagem	**Pré-processamento**	**fator de ponderação**
`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

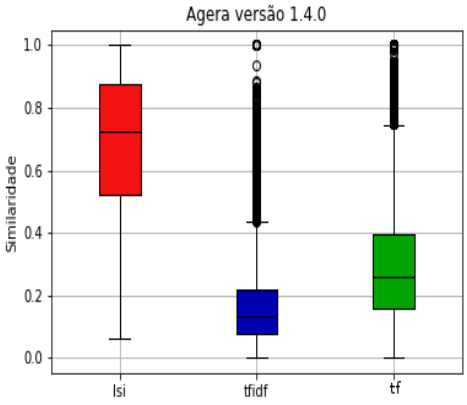
Em termos práticos, em relação a linguagem `C++`, o modelo de Acoplamento Conceitual que apresentou maiores similaridades entre os arquivos de código fonte foi superior em 6 projetos (27 versões). No caso da linguagem `Java`, o modelo com maiores valores de Acoplamento Conceitual foi superior em 10 projetos (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 modelo com maiores valores de acoplamento em 8 projetos (33 versões). Os modelos que apresentaram maiores valores de Acoplamento Conceitual estão disponíveis detalhadamente nos anexos (Capítulo **??**).

A Figura 4.1, apresenta gráficos que mostram as distribuições de valores de Acoplamento Conceitual. Em 4.1(a) está o exemplo da versão `4.9` do projeto `watchman` (`Facebook`), em 4.1(b) a versão `1.4.0` do projeto `Agera` (`Google`), em 4.1(c) a versão `16` do projeto `React` (`Facebook`), em 4.1(d) a versão `0.12` do projeto `Flask` e em 4.1(e) a versão `3.10` do projeto `Capistrano`. Os gráficos revelam os valores de Acoplamento Conceitual de modelos com fator de ponderação LSI superiores aos valores obtidos por modelos com as demais fatores de ponderação. Além disso, observa-se que modelos com fatores de ponderação LSI não apresentam valores com grande afastamento das séries (*outliers*), diferindo dos valores obtidos por modelos com fatores de ponderação TF-IDF e TF.

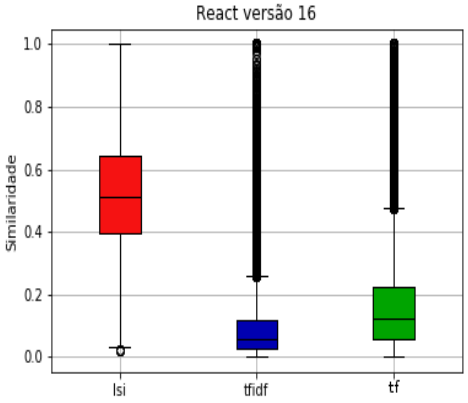
Conclui-se que de uma forma geral, usar a fator de ponderação baseada no LSI fornece valores maiores da similaridade de Acoplamento Conceitual entre os arquivos de código-fonte. No entanto, o pré-processamento depende da linguagem de desenvolvimento do projeto que se deseja analisar.



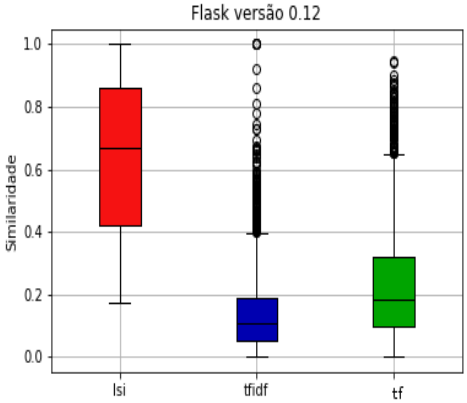
(a) Exemplo de Projeto `C++`



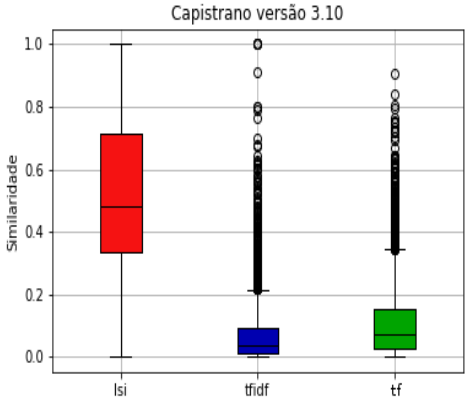
(b) Exemplo de Projeto `Java`



(c) Exemplo de Projeto `Javascript`



(d) Exemplo de Projeto `Python`



(e) Exemplo de Projeto `Ruby`

Figura 4.1. Gráficos de Distribuição de Valores de Acoplamento Conceitual.

QP1: A variação dos parâmetros de pré-processamento e fator de ponderação dos modelos de cálculo resultam em diferentes valores de Acoplamento Conceitual?

Os resultados indicam que os pré-processamentos influenciam o cálculo do Acoplamento Conceitual, e que o LSI é a fator de ponderação que compõem os modelos de cálculo com maiores valores de Acoplamento Conceitual- independente da linguagem - para se calcular o Acoplamento Conceitual. Verificou-se também que as linguagens `C++` e `Java` possuem o conjunto de pré-processamentos que resultam em maiores valores de Acoplamento Conceitual em comum. Esses pré-processamentos compreendem o uso de comentário de código-fonte, não dividir em virtude de *CamelCase*, não lematizar termos e a remoção do terceiro quartil de termos menos frequentes. Por sua vez, as linguagens `Javascript`, `Python` e `Ruby` também apresentam o mesmo conjunto de pré-processamentos para cálculo dos maiores valores de Acoplamento Conceitual, sendo composto pelo uso de comentário de código-fonte, dividir em virtude de *CamelCase*, não lematizar termos e remoção do primeiro quartil de termos menos frequentes. Dessa forma, conclui-se que os pré-processamentos influenciam na obtenção do acoplamento conceitual independentemente da fator de ponderação utilizada.

4.3. Modelos que resultam no cálculo dos maiores valores de Acoplamento Conceitual produzem melhores predições de mudanças conjuntas?

Para responder esta questão de pesquisa, foram realizadas predições de mudanças conjuntas dos 233 intervalos entre as versões de software selecionadas para este estudo por meio dos modelos de cálculo de valores de Acoplamento Conceitual que resultaram em maiores valores obtidos a partir das análises feitas para a primeira questão de pesquisa. A Tabela 4.3 contém os modelos usados para as predições de mudanças conjuntas, da esquerda para a direita se tem a linguagem dos projetos, seguidos pelos correspondentes modelos com fatores de ponderação LSI, TF-IDF e TF. O significado dos rótulos que referenciam os modelos se encontram nos apêndices.

Tabela 4.3. Modelos de cálculo de valores de Acoplamento Conceitual utilizados em análises da Segunda Questão de Pesquisa

LINGUAGEM	LSI	TF-IDF	TF
`C++`, `Java`	`UcoNcaNsR3qLsi`	`UcoUcaNsR1qTfidf`	`UcoUcaNsR1qTf`
`Javascript`, `Python`, `Ruby`	`UcoUcaNsR1qLsi`	`UcoUcaNsR1qTfidf`	`UcoUcaNsR1qTf`

A partir das predições de mudanças conjuntas foram obtidos as distribuições de valores de MAP que indicam a qualidade das recomendações corretas em virtude do modelo de cálculo de valores de Acoplamento Conceitual em cada linguagem. A primeira comparação estabelecida foi entre modelos com fatores de ponderação LSI e TF, o que totaliza 233 comparações entre distribuições de valores de MAP. Os resultados dessas comparações se encontram na Tabela 4.4.

Tabela 4.4. Diferença de tamanho de efeito entre distribuições de valores de MAP obtidas a partir de modelos com fatores de ponderação TF e LSI

Linguagem	**Total**	**Sem diferenças**	**Muito Pequeno**	**Pequeno**	**Médio**	**Grande**
`C++`	38	9 (23.68%)	8 (21.05%)	16 (42.11%)	4 (10.53%)	1 (2.63%)
`Java`	39	7 (17.95%)	6 (15.38%)	18 (46.15%)	8 (20.51%)	0
`Javascript`	47	11 (23.40%)	13 (27.66%)	18 (38.3%)	4 (8.51%)	1 (2.13%)
`Python`	62	27 (43.55%)	18 (29.03%)	14 (22.58%)	3 (4.84%)	0
`Ruby`	47	17 (36.17%)	8 (17.02%)	18 (38.3%)	4 (8.51%)	0

A Tabela 4.4, da esquerda para a direita, estão dispostas as informações sobre a linguagem dos projetos, o total de comparações realizadas por linguagem, números de comparações que resultaram em tamanho de diferença muito pequeno, pequeno, médio e grande. Entre parênteses estão os percentuais representativos das quantidades de comparações, por exemplo, na linguagem `C++`, há 23,68% (9 comparações) entre modelos com fatores de ponderação LSI e TF sem diferenças entre si.

Analisando os resultados das predições de mudanças conjuntas, ao somar os números de comparações que apresentaram diferenças, 76,31% das comparações em projetos `C++` (29 comparações) mostram diferenças entre distribuições de valores de MAP obtidas por modelos com fatores de ponderação LSI e TF, em projetos `Java` foram 82,05% (32 comparações), 76,59% em projetos `Javascript` (36 comparações), 56,45% em projetos `Python` (35 comparações) e 63,82% em projetos `Ruby` (30 comparações). Dessa forma, em 69,52% das comparações ocorreram maiores valores de MAP em predições a partir de modelo com fator de ponderação TF (162 comparações), enquanto em 30,48% (71 comparações) não foram apresentadas diferenças entre as distribuições de valores de MAP.

Em todos os casos em que ocorreram diferenças entre as distribuições de valores de MAP, os valores obtidos por meio de predições realizadas a partir de valores de Acoplamento Conceitual de modelos com fator de ponderação TF foram sempre maiores. Dessa forma, em

66,95% das comparações ocorreram maiores valores de MAP em predições a partir de modelo com fator de ponderação TF (156 comparações), enquanto em 33,05% (77 comparações) não foram apresentadas diferenças entre as distribuições de valores de MAP.

De acordo com os percentuais presentes na tabela, dentre as comparações que apresentaram diferenças para as linguagens `C++`, `Java`, `Javascript` e `Ruby` há um maior número de casos com tamanho de diferença pequeno. As comparações com diferenças em `Python` apresentam maior número com tamanho de diferença pequeno no caso de predições de tamanho fixo. Os dados desses comparações revelam que apesar de modelos com fator de ponderação TF apresentarem maiores valores de MAP, as diferenças entre esses modelos e os com fator de ponderação LSI são pequenas.

Foram feitas 233 comparações entre distribuições de valores de MAP obtidos por meio de predições de mudanças conjuntas realizadas a partir de valores de Acoplamento Conceitual calculados por modelos com fatores de ponderação TF-IDF e LSI. A Tabela 4.5 contém dados referentes a essas comparações.

Tabela 4.5. Diferença de tamanho de efeito entre distribuições de valores de MAP obtidas a partir de modelos com fatores de ponderação TF-IDF e LSI

Linguagem	**Total**	**Sem diferenças**	**Muito Pequeno**	**Pequeno**	**Médio**	**Grande**
`C++`	38	7 (18.42%)	5 (13.16%)	19 (50.0%)	6 (15.79%)	1 (2.63%)
`Java`	39	6 (15.79%)	7 (17.95%)	18 (46.15%)	8 (20.51%)	0
`Javascript`	47	21 (44.68%)	9 (19.15%)	14 (29.79%)	2 (4.26%)	1 (2.13%)
`Python`	62	23 (37.1%)	22 (35.48%)	13 (20.97%)	4 (6.45%)	0
`Ruby`	47	12 (25.53%)	15 (31.91%)	15 (31.91%)	5 (10.63%)	0

Para essas predições de mudanças conjuntas, a soma de comparações que apresentam diferenças entre valores de distribuições de valores de MAP a partir de modelos com fatores de ponderação TF-IDF e LSI representam 81,57% das comparações em projetos `C++` (31 comparações), em projetos `Java` foram 84,61% (33 comparações), 55,31% em projetos `Javascript` (26 comparações), 62,90% em projetos `Python` (39 comparações) e 74.46% em projetos `Ruby` (35 comparações). De acordo com a Tabela 4.5, dentre as comparações que apresentaram diferenças, há percentualmente mais comparações com tamanho de pequeno para as linguagens `C++`, `Java` e `Javascript`. Na linguagem `Python`, entre as diferenças há mais comparações com tamanho de efeito muito pequeno e em `Ruby` ocorreram os mesmo números de comparações com tamanho de efeito muito pequeno e pequeno. Dessa forma, em

70,38% das comparações ocorreram maiores valores de MAP em predições a partir de modelo com fator de ponderação TF-IDF (164 comparações), enquanto em 29,62% (69 comparações) não foram apresentadas diferenças entre as distribuições de valores de MAP.

Apesar de houverem mais casos que apresentam diferenças entre as distribuições de MAP em virtude aos modelos com fatores de ponderação TF-IDF e LSI para todas as linguagens, essas diferenças são pequenas. Além disso, em todos os casos de diferenças, as distribuições de valores de MAP obtidos por meio de predições de mudanças conjunta a partir de modelos com fator de ponderação TF-IDF foram sempre maiores.

Foram feitas 233 comparações entre distribuições de valores de MAP obtidos por meio de predições de mudanças conjuntas realizadas a partir de valores de Acoplamento Conceitual calculados por modelos com fatores de ponderação TF-IDF e TF. Os resultados das comparações que se encontram na Tabela 4.6 são referentes a aquelas em que não correram diferenças entre as distribuições ou nas quais as diferenças apontam que o modelo com fator de ponderação TF-IDF apresentou maiores valores de MAP. Por sua vez, na Tabela 4.7 estão presentes as informações sobre as comparações em que não ocorreram diferenças entre as distribuições de valores de MAP ou os valores de MAP obtidos por meio de modelos com fator de ponderação TF são maiores.

Tabela 4.6. Diferença de tamanho de efeito entre distribuições de valores de MAP obtidas a partir de modelos com fatores de ponderação TF-IDF e TF

Linguagem	**Total**	**Sem diferenças**	**Muito Pequeno**	**Pequeno**	**Médio**	**Grande**
`C++`	38	27 (71.05%)	7 (18.42%)	2 (5.26%)	0	0
`Java`	39	24 (61.53%)	5 (12.82%)	3 (7.68%)	1 (2.56%)	0
`Javascript`	47	20 (42.55%)	6 (12.77%)	11 (23.40%)	2 (4.26%)	0
`Python`	62	32 (51.61%)	12 (19.35%)	9 (14.52%)	0	0
`Ruby`	47	28 (59.57%)	11 (23.40%)	5 (10.63%)	1 (2.13%)	0

Tabela 4.7. Diferença de tamanho de efeito entre distribuições de valores de MAP obtidas a partir de modelos com fatores de ponderação TF e TF-IDF

Linguagem	Total	Sem diferenças	Muito Pequeno	Pequeno	Médio	Grande
C++	38	27 (71.05%)	1 (5.26%)	1 (5.26%)	0	0
Java	39	24 (61.53%)	3 (7.68%)	3 (7.68%)	0	0
Javascript	47	20 (42.55%)	3 (6.38%)	3 (6.38%)	2 (4.26%)	0
Python	62	32 (51.61%)	4 (6.45%)	5 (8.06%)	0	0
Ruby	47	28 (59.57%)	1 (2.13%)	1 (2.13%)	0	0

De acordo com os dados presentes nessas tabelas, a soma de comparações que apresentam diferenças entre valores de distribuições de valores de MAP a partir de modelos com fatores de ponderação TF-IDF e TF representam 28,94% das comparações realizadas em projetos C++ (11 comparações), 38,46% em projetos Java (15 comparações), 42,55% em projetos Javascript (20 comparações), 48,38% em projetos Python (30 comparações) e 40,42% em projetos Ruby (15 Comparações).

Dessas 233 comparações entre valores de MAP com predições de tamanho fixo, 32,18% (75 comparações) apresentam valores maiores obtidos por meio de modelos com fator de ponderação TF-IDF, 11,58% (27 comparações) apresentam valores maiores obtidos a partir de modelos com fator de ponderação TF e 54,50% (127 comparações) não apresentam diferenças entre as distribuições de valores de MAP.

Analisando as comparações das distribuições de valores de MAP obtidos por meio de predições de mudanças feitas a partir de valores de Acoplamento Conceitual obtidos por meio de modelos com fatores de ponderação TF-IDF e TF, é possível afirmar que as distribuições são compostas por valores próximos, pois há a predominância de comparações que não apresentam diferenças. Além disso, as distribuições que apresentam diferenças mostram que os números de comparações que resultaram em tamanho de efeito pequeno e muito pequeno estão em maiores quantidades do que médio e grande. Ainda é possível afirmar que os modelos com fator de ponderação TF-IDF apresentam maior número de comparações que mostram as distribuições com maiores valores de MAP.

Nota-se por meio da Figura 4.2 que as distribuições de valores de MAP relacionadas às predições de mudanças conjuntas se configuram de forma parecida para todas as linguagens. Dessa forma, ao visualizar os gráficos é possível notar que para todas as linguagens os valores de MAP são maiores a partir de modelos com fator de ponderação TF-IDF.

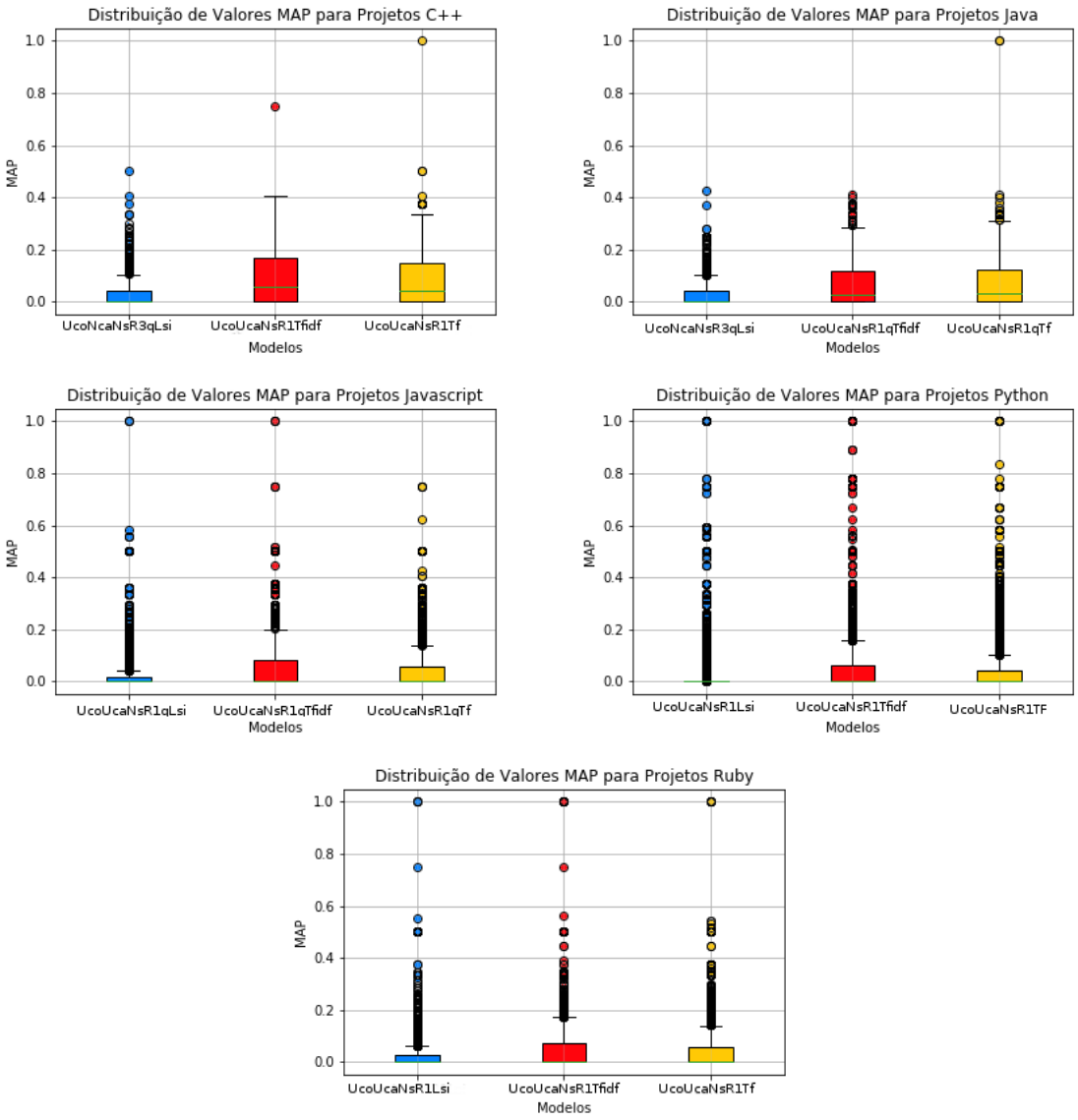


Figura 4.2. Distribuições de Valores de MAP Obtidos para Projetos por Linguagem

A partir dessas constatações, foi verificado de forma empírica qual modelo de cálculo de valores de Acoplamento Conceitual resultou no maior número de arquivos recomendados corretamente. De acordo com os valores de MAP e com os gráficos, é correto afirmar que são os modelos com fator de ponderação TF-IDF.

Tabela 4.8. Números de Arquivos Recomendados por Modelo de Cálculo de Valores de Acoplamento Conceitual

Linguagem	Modelo	A.R	A.R.C
C++	UcoNcaNsR3qLsi	27385	1385 (5.06%)
	UcoUcaNsR1qTfidf		3092 (11.29%)
	UcoUcaNsR1qTf		2872 (10.48%)
Java	UcoNcaNsR3qLsi	14284	1246 (4.97%)
	UcoUcaNsR1qTfidf		2094 (8.36%)
	UcoUcaNsR1qTf		2020 (8.06%)
Javascript	UcoUcaNsR1qLsi	48294	2489 (5.14%)
	UcoUcaNsR1qTfidf		3472 (7.19%)
	UcoUcaNsR1qTf		2874 (5.94%)
Python	UcoUcaNsR1qLsi	202066	10386 (5.14%)
	UcoUcaNsR1qTfidf		10796 (10.68%)
	UcoUcaNsR1qTf		15310 (7.58%)
Ruby	UcoUcaNsR1qLsi	77348	3615 (4.67%)
	UcoUcaNsR1qTfidf		5695 (7.35%)
	UcoUcaNsR1qTf		4694 (6.06%)

Dessa forma, foi confirmado qual modelo de cálculo de valores de Acoplamento Conceitual resultam em melhores predições de mudanças conjuntas por meio da contagem do número de de recomendações corretas por linguagem. A Tabela 4.8 apresenta dados referentes ao número de arquivos recomendados por modelo em virtude da linguagem e dos modelos de cálculo de Acoplamento Conceitual. Da esquerda para a direita estão dados referentes à linguagem dos projetos, os modelos de cálculo de valores de Acoplamento Conceitual, o número de arquivos recomendados (**A.R**) e arquivos recomendados corretamente (**A.R.C**). Entre parenteses estão os valores relacionados aos percentuais do número de arquivos recomendados corretamente em virtude do total de arquivos recomendados. Por exemplo, em `C++`, 1385 arquivos foram recomendados corretamente. Essa quantidade representa 5,06% do total de 27385 arquivos recomendados.

A tabela revela que para as predições de mudanças conjuntas, o modelo `UcoUcaNsR1qTfidf`[1] é aquele que proporciona o maior número de arquivos recomendados corretamente. Esse fato foi observado em todas as linguagens dos projetos coletados para este estudo.

A partir disso, afirma-se que o modelo `UcoUcaNsR1qTfidf` não apresentou os maiores valores de Acoplamento Conceitual, mas foi o que proporcionou os maiores valores de MAP. Além disso, é possível afirmar que maiores valores não são indicativos de qualidade para o

[1] Uso de comentários de código-fonte, dividir *CamelCase*, não aplicar *stemming*, remover o primeiro quartil de termos menos frequentes, fator de ponderação TF-IDF

Acoplamento Conceitual. Apesar das diferenças entre as distribuições de valores de MAP serem pequenas, é importante o fato de que o processo de recomendações de mudanças conjuntas pode ser melhorado.

Q2: Modelos que resultam no cálculo dos maiores valores de Acoplamento Conceitual produzem melhores predições de mudanças conjuntas?

> Não. De acordo com os estudos realizados, os modelos que resultam em cálculo dos maiores valores de Acoplamento Conceitual são os que têm a fator de ponderação LSI em sua composição, especificamente os modelos rotulados por `UcoNcaNsR3qLsi` e `UcoUcaNsR1qLsi`. No entanto, o modelo com fator de ponderação TF-IDF rotulado por `UcoUcaNsR1qTfidf` foi o que proporcionou predições de mudanças conjuntas com distribuições com maiores valores de MAP, sendo assim o modelo que possibilitou o maior número de recomendações corretas. Dessa forma, conclui-se que maiores valores não são medidas de boa qualidade para o Acoplamento Conceitual.

4.4. Ameaças à Validade

Nesta seção são apresentadas e discutidas as limitações e ameaças à validade do estudo realizado. Tais se concentram no método empregado par calcular os valores de Acoplamento Conceitual, bem como avaliar a eficiência dos modelos de cálculo de Acoplamento Conceitual em servir de fonte de recomendações de mudanças conjuntas.

Generalização. Apesar do estudo realizado abranger 5 linguagens de programação, apresentar variação dos domínios de aplicação dos projetos selecionados e o número de versões de projeto de software coletados ser maior do que o encontrado em estudos relacionados ao cálculo do Acoplamento Conceitual, não é possível generalizar os resultados deste estudo. Isto se dá em virtude dos valores de Acoplamento Conceitual ser uma métrica relacionada com a linguagem de programação usada no desenvolvimento do software e a forma como os desenvolvedores escrevem os arquivos de código-fonte, como pode ser observado na resposta da primeira questão de pesquisa. No entanto, a aplicação do método para obter o modelo de cálculo de Acoplamento Conceitual que resulta nos maiores valores pode ser estendido, independentemente da linguagem de programação e do domínio de aplicação do software.

Mudanças de nome do Acoplamento Conceitual. Entre as versões de software é possível identificar as mudanças dos valores de Acoplamento Conceitual entre arquivos de código-fonte. Quando ocorre a modificação dos arquivos, há a mudança do valor de Acoplamento Conceitual. No entanto, às vezes a mudança de um nome não implica a mudança do conceito do conteúdo do arquivo. Por exemplo: Considere dois arquivos de

código-fonte `X` e `Y` de um software. Em `X` e `Y` ocorrem o termo `carro`. A partir de uma determinada versão do software, o nome do termo muda para `automóvel` no arquivo `Y`, permanecendo o mesmo no arquivos `X`. Nesse caso, pode ser percebido que há uma diferença de Acoplamento Conceitual entre esses dois arquivos, pois houve a mudança do conteúdo. No entanto, não houve a mudança do conceito, uma vez que o conceito da palavra `carro` e `automóvel` é o mesmo.

Ordem dos parâmetros de pré-processamento. Neste estudo foram estudadas combinações de diferentes parâmetros de pré-processamento na composição de modelos de cálculo de valores de Acoplamento Conceitual. No entanto, nada pode ser afirmado a respeito da ordem em que esses pré-processamento são aplicados. Assim, não é possível afirmar se a ordem influencia nos valores de Acoplamento Conceitual obtidos. A ordem dos pré-processamentos podem afetar a sequência de passos de construção do modelo de cálculo dos valores de Acoplamento Conceitual, por exemplo, aplicando depois de remover termos menos frequentes poderá fazer com que mais ou menos termos sejam desconsiderado no cálculo.

Seleção de combinações de parâmetros de pré-processamento. No estudo presente nesta monografia foram selecionadas algumas variações de combinações de parâmetros de pré-processamento de texto de arquivos de código-fonte, portanto não foram utilizados todas as combinações possíveis. Elas foram escolhidas com base nas combinações presentes na literatura. Assim, das 16 combinações possíveis foram utilizadas apenas 6. Isso foi necessário em virtude da quantidade de processamento e armazenamento de dados exigidos para suportar o volume de dados gerados pelo cálculo de valores de Acoplamento Conceitual (tabelas no formato `csv` que ocupam grande espaço em disco). Essas limitações reduzem as possibilidades de generalizar os resultados dos estudos, sendo assim que os modelos de cálculo são avaliados em virtude das combinações escolhidas.

Descarte de *commits*. Para avaliar a qualidade da recomendação de mudança conjunta, o método sugere a eliminação de *commit* inválido. Por *commit* inválido, entende-se aquele que se refere à refatorações (KAGDI *et al.*, 2013). Desse modo, neste estudo se assume que todo *commit* cujo número de arquivos de código-fonte modificados seja superior a 10 é um `commit` de refatorações. No entanto, isso não pode ser considerado uma verdade absoluta, sendo assim existe a possibilidade de que *commits* válidos tenham sido removidos da análise realizada.

Uso de Modelos em predições de mudanças conjuntas. A realização das predições de mudanças conjuntas foi pautada nas distribuições com maiores valores de Acoplamento Conceitual. Assim, não foram utilizados todos os modelos estudados neste trabalho. Isso não permite fazer, neste estudo, uma análise direta do impacto de cada parâmetro dos modelos de cálculo de valores de Acoplamento Conceitual nas predições de mudanças conjuntas.

4.5. Considerações Finais

Este capítulo se dedicou à apresentação dos resultados obtidos por meio da análise dos experimentos. Foi possível identificar que a variação modelo de cálculo implica em diferenças de valores de Acoplamento Conceitual, mesmo que pequenas. Além disso, ficou provado que não existe uma relação direta entre maiores valores de Acoplamento Conceitual e melhores predições de mudanças conjuntas. Assim, pode ser observado que dentre as formas de cálculo que resultam os maiores valores de Acoplamento Conceitual, o modelo `UcoUcaNsR1qTfidf` (com uso de comentário de código-fonte, divisão por *CamelCase*, não aplicar *stemming* nos termos, remoção do primeiro quartil de termos menos frequentes e fator de ponderação TF-IDF) foi aquele que proporcionou os valores de Acoplamento Conceitual que melhor refletiram as mudanças conjuntas.

Capítulo 5

Conclusões

O Acoplamento Conceitual é uma métrica que determina a interdependência entre arquivos de código-fonte em virtude de conceitos empregados no desenvolvimento de um projeto de software. Ele é utilizado em tarefas como recomendações de refatoração, melhorar a modularidade, recomendações de mudanças conjuntas entre artefatos e em técnicas para predizer a ocorrência de defeitos.

Este trabalho mostrou que é possível variar o modelo de cálculo dos valores de Acoplamento Conceitual e que isso causa impacto nas distribuições desses valores. A partir das análises, verificou-se que para as linguagens `C++` e `Java` são comuns os pré-processamentos que resultaram nas distribuições dos maiores valores de Acoplamento Conceitual em virtude do fator de ponderação. O mesmo pode ser afirmado para as linguagens `Javascript`, `Python` e `Ruby`. Esse fato está possivelmente relacionado às semelhanças das estruturas do código-fonte e de como os desenvolvedores escrevem esses códigos para essas diferentes linguagens de programação. É possível perceber as semelhanças nos estilos de como estão escritos e organizados os arquivos de código-fonte entre `C++` e `Java`. O mesmo pode ser afirmado entre `Javascript`, `Python` e `Ruby`.

Além disso, pode-se afirmar que as comparações entre as distribuições de valores de Acoplamento Conceitual obtidos por diferentes modelos apresentaram valores muito próximos, tendo a predominância dos tamanhos de diferenças pequeno e muito pequeno. Tal fato foi constatado em todas as linguagens relacionadas aos projetos coletados para este estudo.

Para verificar o impacto da variação dos modelos de cálculo de Acoplamento Conceitual, foram realizadas análises de predições de mudanças conjuntas. Essas predições foram feitas com base nos modelos de cálculo de Acoplamento Conceitual que resultaram nas distribuições com maiores valores para cada um dos fatores de ponderação utilizados neste estudo, sendo

estes o LSI, TF-IDF e o TF. As predições foram feitas com base em 61 projetos de software livre desenvolvidos nas linguagens `C++`, `Java`, `Javascript`, `Python` e `Ruby`. A realização de predições de mudanças conjuntas foi um instrumento muito importante para verificar se maiores valores de Acoplamento Conceitual são medidas mais adequadas de qualidade do Acoplamento Conceitual.

Apesar dos modelos com fator de ponderação LSI apresentarem distribuições de valores de Acoplamento Conceitual maiores, o modelo que se mostrou mais eficiente em predições de mudanças conjuntas foi o com fator de ponderação TF-IDF (`UcoUcaNsR1qTfidf`[1]). Dessa forma, não foi possível estabelecer a relação entre os modelos que resultam em maiores valores de Acoplamento Conceitual e predições mais eficientes de mudanças conjuntas.

Desse modo, é possível afirmar que maiores valores de Acoplamento Conceitual não implicam em valores de Acoplamento Conceitual de maior qualidade. Foi observado que o modelo com normalização composto pelo fator de ponderação TF-IDF (`UcoUcaNsR1qTfidf`) é aquele que além de promover maiores indicativos de predições de mudanças conjuntas, é aquele que apresenta distribuições de valores de Acoplamento Conceitual menores, porém com maior variação entre seus valores.

Considerando tais constatações, surgem questões importantes a respeito dos pré-processamentos. Apesar do *stemming* reduzir os valores das distribuições de Acoplamento Conceitual, ele não melhora a qualidade dos valores? A não remoção de termos de menor frequência de aparições em arquivos de código-fonte não trariam aumento de variância dos valores de Acoplamento Conceitual? Seria possível encontrar um conjunto de pré-processamentos que proporcionem maior variância dos valores de Acoplamento Conceitual e consequentemente melhores medidas para tarefas de engenharia de software?

A partir dessas indagações, como trabalhos futuros há a proposta de encontrar a relação entre a variância das distribuições de valores de Acoplamento Conceitual e melhores medidas para tarefas de engenharia de software, tais como a predição de mudanças. Além disso, este estudo evoluirá para as análises minuciosas do efeito de cada pré-processamento diretamente nas predições de mudanças conjuntas com o objetivo de encontrar o melhor conjunto de pré-processamentos para tarefas de recomendações de mudanças conjuntas em virtude das linguagens em que os projetos são desenvolvidos.

Espera-se que este estudo contribua para o entendimento das características que determinam a qualidade dos valores de Acoplamento Conceitual. Também espera-se que o material desenvolvido em decorrência deste trabalho possa ser aproveitado para pesquisas futuras ou relacionadas com a área. Os *scripts* e todo conjunto de dados gerados e utilizados

[1] Uso de comentário de código-fonte, dividir *CamelCase*, remoção do 1º quartil de termos menos frequentes, uso de fator de ponderação TF-IDF

nesta pesquisa está publicado em (COSTA, 2018a)[2]. Além dessas contribuições, há a ferramenta desenvolvida para o cálculo de valores de Acoplamento Conceitual, o `simitrieve`. Essa ferramenta foi publicada tanto por meio de repositório no `GitHub`[3] como por meio do zenodo[4].

[2] conjunto de dados e *scripts* desenvolvidos neste estudo: `https://doi.org/10.5281/zenodo.1495566`
[3] acesso ao `simitrieve` no `GitHub`: `https://github.com/costabatista/simitrieve`
[4] acesso ao `simitrieve` no zenodo: `https://doi.org/10.5281/zenodo.1492898`

Apêndices

Tabela 1. Versões de Projetos em `C++` Coletadas

Projeto	Organização	Domínio	Versões
electron	electron	Bibliotecas e *frameworks*	1.4.0, 1.5.0, 1.6.0, 1.7.0, 1.8.0
flatbuffers	google	Bibliotecas e *frameworks*	1.4.0, 1.5.0, 1.6.0, 1.7.0, 1.8.0
Gource	acaudwell	Ferramentas de Software	0.35, 0.37, 0.40, 0.45, 0.47
leveldb	google	Ferramentas de Sistemas	1.16, 1.17, 1.18, 1.19, 1.20
liteide	visualfc	Ferramentas de Software	x29, x30, x31, x32, x33
mosh	mobile-shell	Ferramentas de Software	1.0, 1.1, 1.2, 1.3
muduo	chenshuo	Bibliotecas e *frameworks*	0.6.0, 0.7.0, 0.8.0, 0.9.0, 1.0.0
osquery	facebook	Ferramentas de Software	2.7.0, 2.8.0, 2.9.0, 2.10.0, 2.11.0
watchman	facebook	Ferramentas de Software	v4.4.0, v4.5.0, v4.6.0, v4.7.0, v4.9.0
zeal	zealdocs	Aplicações de Software	v0.1.0, v0.2.0, v0.3.0, v0.4.0, v0.5.0

Tabela 2. Versões de Projetos em `Java` Coletadas

Projeto	Organização	Domínio	Versões
agera	google	Bibliotecas e *frameworks*	1.0.0, 1.1.0, 1.2.0, 1.3.0, 1.4.0
Dexter	Karumi	Bibliotecas e *frameworks*	1.0.0, 2.0.0, 3.0.0, 4.0.0
epoxy	airbnb	Bibliotecas e *frameworks*	2.4.0, 2.5.0, 2.6.0, 2.7.0, 2.8.0
fresco	facebook	Bibliotecas e *frameworks*	1.0.0, 1.2.0, 1.4.0, 1.5.0, 1.7.0
guice	google	Bibliotecas e *frameworks*	2.0, 3.0, 4.0
moshi	square	Bibliotecas e *frameworks*	1.1.0, 1.2.0, 1.3.0, 1.4.0, 1.5.0
okhttp	square	Bibliotecas *Web* e *frameworks*	3.5.0, 3.6.0, 3.7.0, 3.8.0, 3.9.0
okio	square	Bibliotecas e *frameworks*	1.5.0, 1.7.0, 1.8.0, 1.9.0, 1.10.0
picasso	square	Bibliotecas e *frameworks*	2.1.0, 2.2.0, 2.3.0, 2.4.0, 2.5.0
stetho	facebook	Bibliotecas e *frameworks*	1.1.0, 1.2.0, 1.3.0, 1.4.0, 1.5.0
tinker	tencent	Bibliotecas e *frameworks*	1.6.0, 1.7.0, 1.8.0, 1.9.0

Tabela 3. Versões de Projetos em `Javascript` Coletadas

Projeto	Organização	Domínio	Versões
angularJS	angular	Bibliotecas *Web* e *frameworks*	4.0.0, 4.3.0, 4.4.0, 5.0.0, 5.1.0
async	caolan	Bibliotecas e *frameworks*	1.2.0, 1.3.0, 1.4.0, 1.5.0, 2.0.0
bluebird	petkaantonov	Bibliotecas e *frameworks*	3.1.0, 3.2.0, 3.3.0, 3.4.0, 3.5.0
cube	square	Bibliotecas e *frameworks*	0.0.10, 0.1.0, 0.2.0, 0.2.5, 0.2.10
enzyme	airbnb	Bibliotecas *Web* e *frameworks*	2.8.0, 2.9.0, 3.0.0, 3.1.0, 3.2.0, 3.3.0
hexo	hexojs	Aplicações de Software	3.0.0, 3.1.0, 3.2.0, 3.3.0, 3.4.0
karma	karma-runner	Ferramentas de Software	1.4.0, 1.5.0, 1.6.0, 1.7.0, 2.0.0
mocha	mochajs	Ferramentas de Software	2.5.0, 3.0.0, 3.4.0, 3.5.0, 4.0.0
nodemon	remy	Ferramentas de Software	1.9.0, 1.10.0, 1.11.0, 1.13.0, 1.14.0
react	facebook	Bibliotecas *Web* e *frameworks*	0.12.0, 0.13.0, 0.14.0, 15.0.0, 16.0.0
redux	reactjs	Bibliotecas *Web* e *frameworks*	3.0.0, 3.3.0, 3.5.0, 3.6.0, 3.7.0
vue	vuejs	Bibliotecas *Web* e *frameworks*	2.1.0, 2.2.0, 2.3.0, 2.4.0, 2.5.0

Tabela 4. Versões de Projetos em `Python` Coletadas

Projeto	Organização	Domínio	Versões
bokeh	bokeh	Bibliotecas *Web* e *frameworks*	0.8.0, 0.9.0, 0.10.0, 0.11.0, 0.12.0
boto	boto	Bibliotecas e *frameworks*	2.9.0, 2.10.0, 2.20.0, 2.30.0, 2.40.0
bypy	houtianze	Bibliotecas *Web* e *frameworks*	1.3.0, 1.4.0, 1.5.0
celery	celery	Bibliotecas e *frameworks*	2.5.0, 3.0.0, 3.1.0, 4.0.0, 4.1.0
certbot	certbot	Ferramentas de Software	0.16.0, 0.17.0, 0.18.0, 0.19.0, 0.20.0
chainer	chainer	Bibliotecas e *frameworks*	2.0.0, 2.1.0, 3.0.0, 3.1.0, 3.2.0
chisel	chisel	Ferramentas de Software	1.3.0, 1.4.0, 1.5.0, 1.6.0, 1.7.0
compose	docker	Ferramentas de Software	1.7.0, 1.8.0, 1.9.0, 1.10.0, 1.11.0
flask	pallets	Bibliotecas *Web* e *frameworks*	0.8, 0.9, 0.10, 0.11, 0.12
ipython	ipython	Bibliotecas e *frameworks*	1.0.0, 2.0.0, 4.0.0, 5.0.0, 6.0.0
keras	fchollet	Bibliotecas e *frameworks*	1.0.0, 1.1.0, 1.2.0, 2.0.0, 2.1.0
pandas	pandas-dev	Bibliotecas e *frameworks*	0.18.0, 0.19.0, 0.20.0, 0.21.0, 0.22.0
requests	requests	Ferramentas de Software	2.14.0, 2.15.0, 2.16.0, 2.17.0, 2.18.0
scrapy	scrapy	Bibliotecas e *frameworks*	1.1.0, 1.2.0, 1.3.0, 1.4.0, 1.5.0
sentry	getsentry	Ferramentas de Software	4.0.0, 5.0.0, 6.0.0, 7.0.0, 8.0.0
sympy	sympy	Bibliotecas e *frameworks*	0.5.0, 0.6.0, 0.7.0, 1.0, 1.1

Tabela 5. Versões de Projetos em `Ruby` Coletadas

Projeto	Organização	Domínio	Versões
bourbon	thoughtbot	Ferramentas de Software	4.0.0, 4.1.0, 4.2.0, 4.2.0, 4.3.0, 5.0.0
capistrano	capistrano	Ferramentas de Software	3.6.0, 3.7.0, 3.8.0, 3.9.0, 3.10.0
celluloid	celluloid	Bibliotecas e *frameworks*	0.13.0, 0.14.0, 0.15.0, 0.16.0, 0.17.0
CocoaPods	CocoaPods	Ferramentas de Software	0.9.0, 1.0.0, 1.1.0, 1.2.0, 1.3.0
devise	plataformatec	Bibliotecas e *frameworks*	4.0.0, 4.1.0, 4.2.0, 4.3.0, 4.4.0
diaspora	diaspora	Aplicações de Software	0.3, 0.4, 0.5, 0.6, 0.7
haml	haml	Bibliotecas *Web* e *frameworks*	1.0.0, 2.0.0, 3.0.0, 4.0.0, 5.0.0
kaminari	kaminari	Bibliotecas *Web* e *frameworks*	0.15.0, 0.16.0,0.17.0, 1.0.0, 1.1.0
puma	puma	Ferramentas de Software	3.7.0, 3.8.0, 3.9.0, 3.10.0, 3.11.0
sass	sass	Ferramentas de Software	3.1.0, 3.2.0, 3.3.0, 3.4.0, 3.5.0
sinatra	sinatra	Bibliotecas *Web* e *framworks*	1.1.0, 1.2.0, 1.3.0, 1.4.0, 2.0.0
thor	erikhuda	Bibliotecas e *frameworks*	0.16.0, 0.17.0, 0.18.0, 0.19.0, 0.20.0

Tabela 6. Significado dos acrônimos dos modelos de cálculo de Acoplamento Conceitual

Acrônimo	Significado
Uco	Utiliza comentário de código-fonte
Nco	Não utiliza Comentário de código-fonte
Uca	Utiliza divisão por *CamelCase*
Nca	Não utiliza divisão por *CamelCase*
Us	Utiliza *stemming*
Ns	Não utiliza *stemming*
R3q	Remoção do terceiro quartil de termos menos frequentes
R1	Remoção do primeiro quartil de termos menos frequentes
Lsi	Uso de fator de ponderação LSI
Tfidf	Uso de fator de ponderação TFIDF
Tf	Uso de fator de ponderação TF

Tabela 7. Modelos que resultaram em maiores Valores de Acoplamento Conceitual, linguagem C++

Projeto	Modelo
electron	UcoUcaNsR1qLsi
flatbuffers	UcoUcaUsR3qLsi
gource	UcoNcaNsR3qLsi
leveldb	UcoUcaNsR1qLsi
liteide	UcoNcaNsR3qLsi
mosh	UcoNcaNsR3qLsi
muduo	UcoNcaNsR3qLsi
osquery	UcoNcaNsR3qLsi
watchman	UcoUcaNsR1qLsi
zeal	UcoNcaNsR3qLsi

Tabela 8. Modelos que resultaram em maiores Valores de Acoplamento Conceitual, linguagem Java

Projeto	Modelo
agera	UcoNcaNsR3qLsi
dexter	UcoNcaNsR3qLsi
epoxy	UcoUcaNsR1qLsi
fresco	UcoNcaNsR3qLsi
guice	UcoNcaNsR3qLsi
moshi	UcoNcaNsR3qLsi
okhttp	UcoNcaNsR3qLsi
okio	UcoNcaNsR3qLsi
picasso	UcoNcaNsR3qLsi
stetho	UcoNcaNsR3qLsi
tinker	UcoNcaNsR3qLsi

Tabela 9. Modelos que resultaram em maiores Valores de Acoplamento Conceitual, linguagem Javascript

Projeto	Modelo
angular	UcoUcaNsR1qLsi
async	UcoUcaUsR3qLsi
bluebird	UcoNcaNsR3qLsi
cube	UcoUcaNsR1qLsi
enzyme	NcoNcaUsR1qLsi
hexo	NcoNcaNsR1qLsi
karma	UcoUcaNsR1qLsi
mocha	NcoNcaUsR1qLsi
nodemon	NcoNcaUsR1qLsi
react	UcoUcaNsR1qLsi
redux	UcoUcaNsR1qLsi
vue	UcoUcaNsR1qLsi

Tabela 10. Modelos que resultaram em maiores Valores de Acoplamento Conceitual, linguagem Ruby

Projeto	Modelo
bourbon	NcoNcaUsR1qLsi
capistrano	UcoUcaNsR1qLsi
celluloid	UcoUcaNsR1qLsi
cocoapods	UcoUcaNsR1qLsi
devise	NcoNcaNsR1qLsi
diaspora	UcoUcaNsR1qLsi
haml	UcoUcaNsR1qLsi
kaminari	UcoUcaNsR1qLsi
puma	NcoNcaNsR1qLsi
sass	UcoUcaNsR1qLsi
sinatra	UcoUcaNsR1qLsi
thor	NcoNcaNsR1qLsi

Tabela 11. Modelos que resultaram em maiores Valores de Acoplamento Conceitual, linguagem `Python`

Projeto	Modelo
bokeh	UcoUcaNsR1qLsi
boto	UcoNcaNsR3qLsi
bypy	UcoUcaNsR1qLsi
celery	NcoNcaNsR1qLsi
certbot	UcoUcaNsR1qLsi
chainer	NcoNcaNsR1qLsi
chisel	UcoNcaNsR3qLsi
compose	UcoUcaNsR1qLsi
flask	NcoNcaUsR1qLsi
ipython	UcoUcaNsR1qLsi
keras	UcoUcaNsR1qLsi
pandas	UcoUcaNsR1qLsi
requests	UcoUcaNsR1qLsi
scrapy	NcoNcaNsR1qLsi
sentry	UcoUcaNsR1qLsi
sympy	UcoUcaNsR1qLsi

Referências

ANTONIOL, G.; CANFORA, G.; CASAZZA, G.; LUCIA, A. De. Information retrieval models for recovering traceability links between code and documentation. In: *Proceedings 2000 International Conference on Software Maintenance*. [S.l.: s.n.], 2000. p. 40–49. ISSN 1063-6773.

BAVOTA, Gabriele; LUCIA, Andrea De; MARCUS, Andrian; OLIVETO, Rocco. Software re-modularization based on structural and semantic metrics. In: IEEE. *Reverse Engineering (WCRE), 2010 17th Working Conference on*. [S.l.], 2010. p. 195–204.

BAVOTA, Gabriele; LUCIA, Andrea De; OLIVETO, Rocco. Identifying extract class refactoring opportunities using structural and semantic cohesion measures. *Journal of Systems and Software*, Elsevier, v. 84, n. 3, p. 397–414, 2011.

COSTA, P. Batista da. *Conceptual Coupling Values - Github Projects Dataset*. Zenodo, 2018. Disponível em: `https://doi.org/10.5281/zenodo.1495566`.

COSTA, P. Batista da. *Simitrieve*. Zenodo, 2018. Disponível em: `https://doi.org/10.5281/zenodo.1492898`.

GARNIER, Marcelo; GARCIA, Alessandro. On the evaluation of structured information retrieval-based bug localization on 20 c# projects. In: ACM. *Proceedings of the 30th Brazilian Symposium on Software Engineering*. [S.l.], 2016. p. 123–132.

HYNDMAN, Rob J; FAN, Yanan. Sample quantiles in statistical packages. *The American Statistician*, Taylor & Francis, v. 50, n. 4, p. 361–365, 1996.

ISO, IEC. Ieee, systems and software engineering–vocabulary. *IEEE computer society, Piscataway, NJ*, 2010.

ISO, ISO; IEC, TR. 19759: Guide to the software engineering body of knowledge (swebok). *International Organization for Standardization, Geneva, Switzerland*, 2005.

KAGDI, Huzefa; GETHERS, Malcom; POSHYVANYK, Denys. Integrating conceptual and logical couplings for change impact analysis in software. *Empirical Software Engineering*, Springer, v. 18, n. 5, p. 933–969, 2013.

KAGDI, Huzefa; GETHERS, Malcom; POSHYVANYK, Denys; COLLARD, Michael L. Blending conceptual and evolutionary couplings to support change impact analysis in source code. In: IEEE. *Reverse Engineering (WCRE), 2010 17th Working Conference on*. [S.l.], 2010. p. 119–128.

LAWRIE, Dawn; BINKLEY, Dave. Expanding identifiers to normalize source code vocabulary. IEEE, 2011.

LAWRIE, Dawn; BINKLEY, Dave; MORRELL, Christopher. Normalizing source code vocabulary. In: IEEE. *Reverse Engineering (WCRE), 2010 17th Working Conference on*. [S.l.], 2010. p. 3–12.

LINSTEAD, Erik; RIGOR, Paul; BAJRACHARYA, Sushil; LOPES, Cristina; BALDI, Pierre. Mining concepts from code with probabilistic topic models. In: *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*. New York, NY, USA: ACM, 2007. (ASE '07), p. 461–464. ISBN 978-1-59593-882-4. Disponível em: `http://doi.acm.org/10.1145/1321631.1321709`.

LOVINS, Julie Beth. Development of a stemming algorithm. *Mech. Translat. & Comp. Linguistics*, v. 11, n. 1-2, p. 22–31, 1968.

LUCIA, Andrea De; FASANO, Fausto; OLIVETO, Rocco; TORTORA, Genoveffa. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Trans. Softw. Eng. Methodol.*, ACM, New York, NY, USA, v. 16, n. 4, set. 2007. ISSN 1049-331X. Disponível em: `http://doi.acm.org/10.1145/1276933.1276934`.

LUCIA, Andrea De; FASANO, Fausto; OLIVETO, Rocco; TORTORA, Genoveffa. Recovering traceability links in software artifact management systems using information retrieval methods. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, ACM, v. 16, n. 4, p. 13, 2007.

LUKINS, Stacy K; KRAFT, Nicholas A; ETZKORN, Letha H. Source code retrieval for bug localization using latent dirichlet allocation. In: IEEE. *Reverse Engineering, 2008. WCRE'08. 15th Working Conference on*. [S.l.], 2008. p. 155–164.

MARCUS, Andrian. Semantic driven program analysis. In: IEEE. *Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on*. [S.l.], 2004. p. 469–473.

MARCUS, Andrian; MALETIC, Jonathan I. Recovering documentation-to-source-code traceability links using latent semantic indexing. In: IEEE. *Software Engineering, 2003. Proceedings. 25th International Conference on*. [S.l.], 2003. p. 125–135.

MARCUS, Andrian; POSHYVANYK, Denys; FERENC, Rudolf. Using the conceptual cohesion of classes for fault prediction in object-oriented systems. *IEEE Transactions on Software Engineering*, IEEE, v. 34, n. 2, p. 287–300, 2008.

POSHYVANYK, Denys. Using information retrieval to support software maintenance tasks. In: IEEE. *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*. [S.l.], 2009. p. 453–456.

POSHYVANYK, Denys; MARCUS, Andrian. The conceptual coupling metrics for object-oriented systems. In: IEEE. *Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on*. [S.l.], 2006. p. 469–478.

POSHYVANYK, D.; MARCUS, A. Combining formal concept analysis with information retrieval for concept location in source code. In: *15th IEEE International Conference on Program Comprehension (ICPC '07)*. [S.l.: s.n.], 2007. p. 37–48. ISSN 1092-8138.

RAEMAEKERS, Steven; DEURSEN, Arie Van; VISSER, Joost. Semantic versioning versus breaking changes: A study of the maven repository. In: IEEE. *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on*. [S.l.], 2014. p. 215–224.

SAFEER, Yasir; MUSTAFA, Atika; ALI, Anis Noor. Clustering unstructured data (flat files)-an implementation in text mining tool. *arXiv preprint arXiv:1007.4324*, 2010.

SUN, Chengnian; LO, David; WANG, Xiaoyin; JIANG, Jing; KHOO, Siau-Cheng. A discriminative model approach for accurate duplicate bug report retrieval. In: *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*. New York, NY, USA: ACM, 2010. (ICSE '10), p. 45–54. ISBN 978-1-60558-719-6. Disponível em: `http://doi.acm.org/10.1145/1806799.1806811`.

UJHAZI, B.; FERENC, R.; POSHYVANYK, D.; GYIMOTHY, T. New conceptual coupling and cohesion metrics for object-oriented systems. In: *2010 10th IEEE Working Conference on Source Code Analysis and Manipulation*. [S.l.: s.n.], 2010. p. 33–42.

WALL, Michael E; RECHTSTEINER, Andreas; ROCHA, Luis M. Singular value decomposition and principal component analysis. In: *A practical approach to microarray data analysis*. [S.l.]: Springer, 2003. p. 91–109.

WILLETT, Peter. The porter stemming algorithm: then and now. *Program*, Emerald Group Publishing Limited, v. 40, n. 3, p. 219–223, 2006.

ZIMMERMANN, T.; ZELLER, A.; WEISSGERBER, P.; DIEHL, S. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, v. 31, n. 6, p. 429–445, June 2005. ISSN 0098-5589.