Heurísticas para Identificação de Ambiguidade de Autores em Projetos *Open Source*

Talita S. Orfanó [1], Kecia A. M. Ferreira [2], Mariza A. S. Bigonha [1]

Departamento de Ciência da Computação - ICEx/UFMG [1]
Departamento de Computação - CEFET-MG [2]

{talita.orfano, mariza}@dcc.ufmg.br [1], kecia@decom.cefetmg.br [2]

Abstract. *Detection of ambiguity among project developers has been a constant challenge in the study of public repositories. In order to mitigate this problem, we implemented and analyzed five heuristics that aim to identify ambiguities in GitHub projects. These heuristics were originally created to solve the problem in the context of mailing lists. The heuristics that obtained the best results were the Bird, Robles and Canfora.*

Resumo. *A detecção de ambiguidade de desenvolvedores de um projeto tem sido um constante desafio enfrentado no estudo de repositórios públicos. Com a intenção de mitigar este problema, implementamos e analisamos cinco heurísticas que visam identificar ambiguidades em projetos do GitHub. Essas heurísticas foram criadas originalmente para solucionar o problema no contexto de listas de discussões de e-mail. As heurísticas que obtiveram os melhores resultados foram as de Bird, Robles e Canfora.*

1. Introdução

Diante do essencial papel que os repositórios públicos assumiram nos últimos anos, como o GitHub, Bitbucket e GitLab, as análises e estudos acerca de seus dados e características estão constantemente presentes na literatura. Dentre as principais informações coletadas, encontram-se os dados dos contribuidores responsáveis pelas alterações feitas em um ou mais projetos do repositório. Porém, nesse conjunto de dados frequentemente estão presentes contribuidores ambíguos, isto é, usuários que, por utilizarem nomes ou e-mails diferentes em um mesmo projeto são identificados erroneamente como sendo duas ou mais pessoas distintas, e com isso a análise dos estudos é impactada.

A identificação dessa ambiguidade de autores não é uma tarefa fácil, pois não há um padrão universal para a criação de prefixos de e-mail ou para nome definido pelo usuário no repositório. Segundo Wiese et al. [2016] , essa dificuldade na identificação dos contribuidores é uma ameaça importante à validade de estudos que se baseiam nessa informação. Destarte, existem na literatura significativos trabalhos no qual foram desenvolvidas heurísticas para solucionar esse recorrente problema, Bird et al. [2006], Canfora et al. [2011], Robles & Gonzales-Barahona [2005], Goeminne & Mens [2011], Kounters et al. [2012], dentre outros.

O objetivo deste trabalho é analisar de forma comparativa cinco heurísticas de detecção de ambiguidades, a partir de contribuidores de projetos do GitHub. Usamos esse repositório para análise, pois atualmente é o repositório onde estão hospedados o maior número de softwares públicos, dadas as facilidades oferecidas pela API GitHub

para extração dos dados, e consequentemente, por ser o mais usado como fonte de dados para estudos científicos.

Para a análise foram coletados dados dos colaboradores dos projetos do GitHub Joda-Time[1] , PowerShell[2] e ElasticSearch[3]. Foram implementadas as heurísticas propostas por Bird et al. [2006], Canfora et al. [2011], Robles & Gonzales-Barahona [2005], Goeminne & Mens [2013] e a Simples de Kounters et al. [2012]. Foi criada uma base de referência para avaliação comparativa acerca do resultado de cada uma das heurísticas.

Os resultados apontam que para projetos de software com até 100 contribuidores, as heurísticas retornam bons resultados, exceto a heurística Simples. Para aqueles de médio ou grande porte, recomenda-se usar as heurísticas de Bird, Robles ou Canfora. Nota-se que quanto maior o número de contribuidores, menor é a precisão das heurísticas.

As principais contribuições desse estudo compreendem: (1) a implementação das heurísticas propostas por Bird et al., Canfora et al., Robles & Gonzales-Barahona, Goeminne & Mens e a Simples de Kounters et al. adaptadas para o contexto do GitHub; (2) avaliação comparativa acerca do resultado de cada uma das heurísticas, tendo como base os dados coletados dos colaboradores de projetos do GitHub; (3) criação de três bases de referência para cada projeto analisado; (4) um algoritmo para validação dos resultados obtidos pelas heurísticas com os dados da base de referência. Acredita-se que essas implementações e o resultado deste estudo contribuem para a comunidade científica e auxiliarão na detecção de ambiguidade de futuros trabalhos que usarem dados dos contribuidores GitHub, garantindo resultados mais assertivos.

Este artigo está organizado da seguinte forma: Seção 2 apresenta um conjunto de trabalhos relacionados a temática abordada; as seções 3 e 4 mostram a metodologia usada na condução do trabalho realizado, apresentando, respectivamente, a coleta dos dados, a implementação das heurísticas e suas análises; Seção 5 expõe os resultados encontrados para cada heurística; Seção 6 discute e analisa de forma comparativa os resultados encontrados e os pontos que ameaçam a validade do trabalho e a Seção 7 sumariza os resultados encontrados e aponta os trabalhos futuros.

2. Trabalhos Relacionados

O processo para identificar ambiguidades de membros e desenvolvedores de um projeto, tem motivado diversos estudos nas últimas décadas. O trabalho de Bird et al. [2006] é um dos mais referenciados na literatura. Eles investigam como ocorre a comunicação e coordenação entre membros de um projeto *Open Source*. Para isso, realizam uma mineração da rede social do projeto Apache HTTP Server, a partir de todas as listas de discussão de e-mail de 1999 a 2006. Dessarte, eles propuseram uma heurística para identificar e tratar a ambiguidade dos participantes dessa lista. A heurística avalia por meio da distância de Levenshtein[4], a similaridade entre nomes e e-mails dos membros da lista de discussão.

Canfora et al. [2011] buscaram estudar as interações sociais entre os contribuidores responsáveis pelo *kernel* de dois sistemas operacionais *Open Source*: FreeBSD e

[1]Joda-Time: `https://github.com/JodaOrg/joda-time`
[2]PowerShell: `https://github.com/PowerShell/PowerShell`
[3]ElasticSearch: `https://github.com/elastic/elasticsearch`
[4]Levenshtein: `http://rosettacode.org/wiki/Levenshtein_distance`

OpenBSD. A coleta foi feita a partir de arquivos de lista de e-mail e repositórios CVS. Para criação da heurística, Canfora et al. basearam-se no estudo apresentado por Bird et al. [2006], porém não utilizaram da distância de Levenshtein. A heurística proposta para avaliação desses projetos compara o nome, o e-mail e suas diferentes derivações.

Robles & Gonzales-Barahona [2005] buscaram reconhecer a identidade dos desenvolvedores por meio da coleta de informações entre diversas fontes de dados em contextos distintos como: lista de *e-mail*, código fonte, repositório de controle de versão, *bug tracking*, dentre outras. A heurística proposta foi aplicada no projeto GNOME. Fontes como servidores GPG e o repositório CVS forneceram nome, *username* e *e-mails* dos desenvolvedores do projeto.

Goeminne & Mens [2011] e, posteriormente, Kounters et al. [2012] propuseram uma heurística simples para detecção de ambiguidade de autores. Essa heurística avalia a similaridade da identidade dos autores, considerando que os elementos em análise, isto é, nome ou *e-mails*, sejam iguais em pelo menos um comprimento mínimo *l*. Em seu trabalho, Goeminne & Mens [2011] avaliaram de forma comparativa as heurísticas de Bird et al. [2006], Robles & Gonzales-Barahona [2005], a simples e outra heurística proposta pelos autores. Foram avaliados três projetos *Open Source*: Brasero, Evince e Subversion. Foram coletados dados dos repositórios: (1) de código fonte, (2) de listas de discussão e (3) repositórios com o registro dos *bugs* reportados no projeto.

Wiese et al. [2016] comparam seis heurísticas que buscam identificar múltiplos endereços de *e-mail* pertencentes a um mesmo membro, em listas de discussão de *e-mail*. Os autores analisam 150 listas de discussão referentes ao projeto da Apache Software Foundation. Esse estudo também avalia se a janela de tempo e o tamanho da comunidade influenciam no desempenho das heurísticas.

Embora hajam diversos trabalhos que avaliam heurísticas no contexto de lista de discussão, ainda há pouco estudo relacionado com a assertividade dessas heurísticas no contexto exclusivo de repositórios GitHub. Como a demanda pela detecção de ambiguidade é alta nos trabalhos que mineram dados de repositórios públicos, e em geral, os autores não possuem a disponibilidade para avaliar outras fontes de dados, este trabalho de pesquisa busca suprir essa necessidade, via a avaliação das heurísticas propostas por Bird et al., Canfora et al., Robles & Gonzales-Barahona, Goeminne & Mens e Kounters et al., usando projetos presentes no GitHub. As seções 3 e 4 descrevem a coleta de dados a partir de projetos do GitHub, a implementação das heurísticas para a detecção de ambiguidade de contribuidores e o algoritmo usado para sua avaliação.

3. Coleta de Dados

Nessa etapa foram coletados dados de três projetos do GitHub: Elasticsearch, PowerShell e Joda-Time. Esses dados correspondem ao conjunto <nome, *e-mail*> de cada contribuidor do projeto. Neste artigo designamos autor ou contribuidor de um projeto, o usuário responsável por efetuar cada *commit* analisado no repositório. Consideramos também projeto de pequeno porte aquele que contém até 100 contribuidores, médio os que possuem entre 100 e 500 autores e de grande porte, os projetos que possuem acima de 500.

Elasticsearch é um software que permite a busca e análise de um grande conjunto de dados em tempo real. O PowerShell é um *framework* da Microsoft multiplataforma

Tabela 1: Projetos do GitHub selecionados para análise.

Projeto	# de estrelas	# *releases*	# *commits*	# autores
Elasticsearch	26884	198	29220	908
PowerShell	8447	40	5563	162
Joda-Time	3252	54	2052	64

para automação e gerenciamento, incluindo um *shell* de linha de comando. O Joda-Time era a substituição mais utilizada para as classes de data e hora de Java anteriores a versão Java SE 8. A Tabela 1 sumariza algumas das principais informações sobre esses projetos.

Para validação dos resultados obtidos nas heurísticas fez-se necessário a criação de uma base de referência. A compilação dessa base foi feita de forma manual. Para essa construção foram considerados o nome, o *e-mail* e dados obtidos por meio de pesquisas externas ao GitHub. A base de referência foi registrada em um arquivo com extensão .csv onde, em uma mesma linha estão os *e-mails* identificados como sendo de um mesmo autor, de forma que cada linha dessa base corresponda a um autor distinto.

4. Implementação das Heurísticas

Esta seção descreve a implementação das heurísticas de Bird et al., Canfora et al., Robles & Gonzales-Barahona, Goeminne & Mens e a simples heurística proposta por Kounters et al. e Goeminne & Mens, para detecção de ambiguidade de autores de um repositório. Essas heurísticas possuem algumas ações em comum como a análise apenas do prefixo do *e-mail*; a normalização do nome e *e-mail*, retirando acento, letras maiúsculas e outras particularidades de cada heurística; a verificação da igualdade do nome e verificação da igualdade entre os prefixos de *e-mail*. No entanto, como essas heurísticas foram amplamente reconhecidas e estudadas no âmbito de listas de discussão de *e-mails*, especialmente no contexto de projetos *Open Source*, foi necessário uma adaptação de cada uma delas, uma vez que a fonte de dados considerada em nosso trabalho foi unicamente o GitHub, diferentemente do propósito original das heurísticas.

Heurística de Bird et al. [2006] . Esta heurística compara a similaridade entre o primeiro e último nome, o nome completo e o prefixo de *e-mail*. As duas identidades <*nome, e-mail*> são consideradas como pertencentes a um mesmo autor se a similaridade de Levenshtein for maior ou igual ao *threshold t*. Além disso, também é verificado se o prefixo de *e-mail* de um desenvolvedor contém o nome, sobrenome ou derivações desses, semelhantes ao outro desenvolvedor. Foram escolhidos arbitrariamente cinco valores para *t*: 0,84; 0,87; 0,90; 0,93; 0,96 e 0,99. Para cada um dos softwares analisados foram realizados seis experimentos com essa heurística, um para cada valor de *t*.

Heurística de Canfora et al. [2011] . Podemos seguimentar essa heurística em dois grandes módulos, o módulo que valida a ambiguidade entre os nomes e suas derivações, e o módulo que compara os *e-mails* e as suas derivações. O primero módulo compara: os nomes ignorando o nome do meio; o nome com as iniciais, comprovando que j k s é igual a *John Kennedy Simith*; apenas o último nome e outras derivações originárias do nome dos contribuidores. Enquanto o segundo módulo compara a igualdade entre os prefixos, o nome extraído a partir do prefixo, como *john.smith* é equivalente a *John*

Smith e outras informações oriundas do prefixo e comparadas com o nome dos autores.

Heurística de Robles & Gonzales-Barahona [2005] . A partir do nome do desenvolvedor, esta heurística cria um conjunto de possíveis prefixos de *e-mail* e a utiliza para comparar com o prefixo de *e-mail* de outro desenvolvedor. Alguns possíveis prefixos são, por exemplo, nome@dominio.com, nome.sobrenome@dominio.com, nome_sobrenome@dominio.com.

Heurística de Goeminne & Mens [2011] . Dois contribuidores são considerados ambíguos se a similaridade de Levenshtein entre os nomes ou prefixos de *e-mail* for maior ou igual a *t*. Também é verificado se o prefixo de *e-mail* contém parte do nome (de tamanho l), para isso o nome é separado por meio dos caracteres '_', '-', ' ', '+', ',' e '.'. Essa heurística se baseia em Robles & Gonzales-Barahona [2005] e cria uma lista de possíveis prefixos de *e-mail*, a partir do nome do contribuidor. Alguns possíveis prefixos são nomesobrenome, nome.sobrenome, nome_sobrenome, nsobrenome, nomeinicialsobrenome, nome-sobrenome, sobrenomenome, sobrenome_nome e assim por diante.

Heurística de Simples - Kounters et al. [2012] e Goeminne & Mens [2011] . Esta heurística verifica se o primeiro rótulo, o nome ou prefixo do *e-mail*, contém parte (de tamanho *l*) do segundo nome e o inverso também. A mesma análise é feita sobre o prefixo dos e-mails. Nessa heurística também foram escolhidos arbitrariamente cinco valores para *l*, variando de 4 a 8. Para cada software em análise obtivemos cinco resultados que serão expostos na Seção 6.

A avaliação das heurísticas foi feita por meio de um algoritmo que compara a base de referência construída com os autores identificados por cada heurística. Esse algoritmo classifica cada comparação entre os autores como sendo *falso-positivo, falso-negativo, verdadeiro-positivo* ou *verdadeiro-negativo*. Considera-se Falso-Positivo (FP) se a heurística afirma que ambos autores têm a mesma identidade, mas a base de referência garante que são pessoas diferentes. Falso-Negativo (FN) ocorre quando a heurística declara que ambos autores possuem identidades diferentes, no entanto a base referência comprova que as identidades pertencem a uma mesma pessoa. Verdadeiro-Positivo (TP) ocorre se a base de referência e a heurística constatam que ambas identidades pertencem ao mesmo contribuidor, enquanto o Verdadeiro-Negativo (TN) atesta o inverso, isto é, tanto a heurística quanto a base concordam que os autores possuem identidades distintas. Esses dados são usados no cálculo do valor de precisão e *recall* de cada uma das heurísticas [Goeminne & Mens 2011]. Desta forma, temos:

$$Precisao = \frac{tp}{tp + fp} (1) \qquad Recall = \frac{tp}{tp + fn} (2)$$

5. Resultados

Nesta seção são apresentados os resultados de cada heurística para os três projetos analisados: Joda-Time, PowerShell e ElasticSearch. A Tabela 2 sumariza os resultados de precisão e *recall* apresentados para cada uma das heurísticas.

Joda-Time. As heurísticas analisadas apresentaram um ótimo resultado de *recall*, conforme ilustra a Tabela 2. Isso ocorreu pois em todas as comparações realizadas pelas heurísticas não foi apontado nenhum resultado falso negativo. Exceto pela heurística simples,

Tabela 2: Precisão e *recall* das heurísticas no Joda-Time, PowerShell e ElasticSearch

	Precisão			*Recall*		
Heurísticas	Joda-Time	PoweShell	ElasticSearch	Joda-Time	PoweShell	ElasticSearch
Bird-84	1	0,32	0,32	1	0,93	0,96
Bird-87	1	0,33	0,33	1	0,93	0,96
Bird-90	1	0,33	0,33	1	0,93	0,95
Bird-93	1	0,33	0,33	1	0,93	0,95
Bird-96	1	0,33	0,33	1	0,93	0,95
Bird-99	1	0,33	0,33	1	0,93	0,95
Canfora	1	0,74	0,32	1	0,93	0,95
Simples-4	0,03	0,004	2,50E-04	1	0,96	1
Simples-5	0,17	0,1	633E-04	1	0,93	0,94
Simples-6	0,21	0,18	0,0038	1	0,93	0,94
Simples-7	0,43	0,29	0,07	1	0,93	0,93
Simples-8	0,43	0,56	0,14	1	0,93	0,92
Robles	1	0,65	0,28	1	0,96	0,98
Goeminne	1	0,58	0,07	1	0,96	0,98

as demais apresentaram 100% de precisão. O gráfico da Figura 1 mostra os resultados de *recall* e precisão, no eixo x e y respectivamente, para cada uma das heurísticas analisadas.

PowerShell. As heurísticas que obtiveram os melhores resultados foram de Canfora e Robles com 74% e 65% de precisão respectivamente. O resultado do *recall* de todas heurísticas é considerado ótimo, variando de 93 a 96%. Figura 2 ilustra esses resultados.

ElasticSearch. Para esse projeto, as heurísticas de Bird, Canfora e Robles foram aquelas com melhor precisão se comparado com as demais, no entanto a precisão foi consideravelmente baixa, atingindo apenas 33% no melhor resultado. A precisão das heurísticas Simples com comprimento l de 4, 5 e 6 tendeu a zero, de acordo com a Tabela 2. O valor de *recall* foi ótimo em todas as heurísticas com valores variando de 92 a 100%. Figura 3 apresenta de forma expressiva esses resultados.

6. Discussão

Os resultados deste trabalho de pesquisa mostraram que a precisão tende a reduzir proporcionalmente com o aumento do número de contribuidores no projeto. Enquanto o software Joda-Time considerado como de pequeno porte atingiu uma precisão de 100% em grande parte das heurísticas, Elasticsearch considerado de grande porte conquistou aproximadamente 33% de precisão nos melhores resultados das heurísticas. Os resultados obtidos para o *recall* foram ótimos em todas as heurísticas nos três projetos analisados. Isso mostra que as heurísticas raramente deixam de detectar uma ambiguidade.

Para compreender melhor a razão pela qual a precisão foi tão baixa para o projeto ElasticSearch, examinamos manualmente os resultados gerados por cada heurística e observamos que muitas identidades foram unidas de forma equivocada devido ao prefixo idêntico das mesmas, como é o caso dos prefixos “github”, “contact” e “mail”, como por exemplo “github@smithsrock.com” e “github@tobigue.de”.

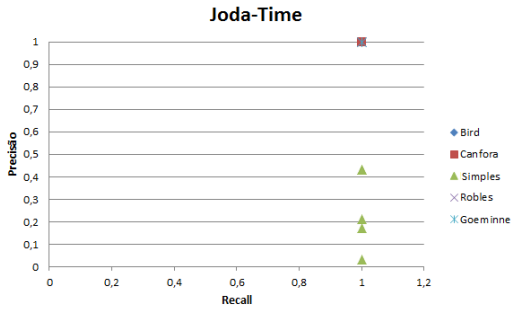

Figura 1: Representação da interação entre desenvolvedores e suas contribuições no Joda-Time

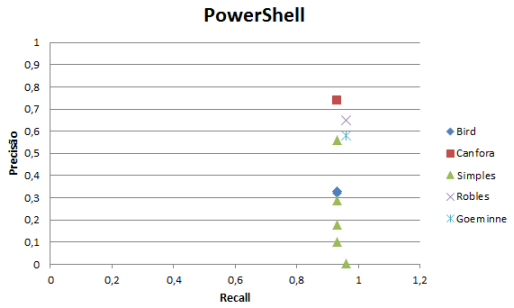

Figura 2: Representação da interação entre desenvolvedores e suas contribuições no PowerShell

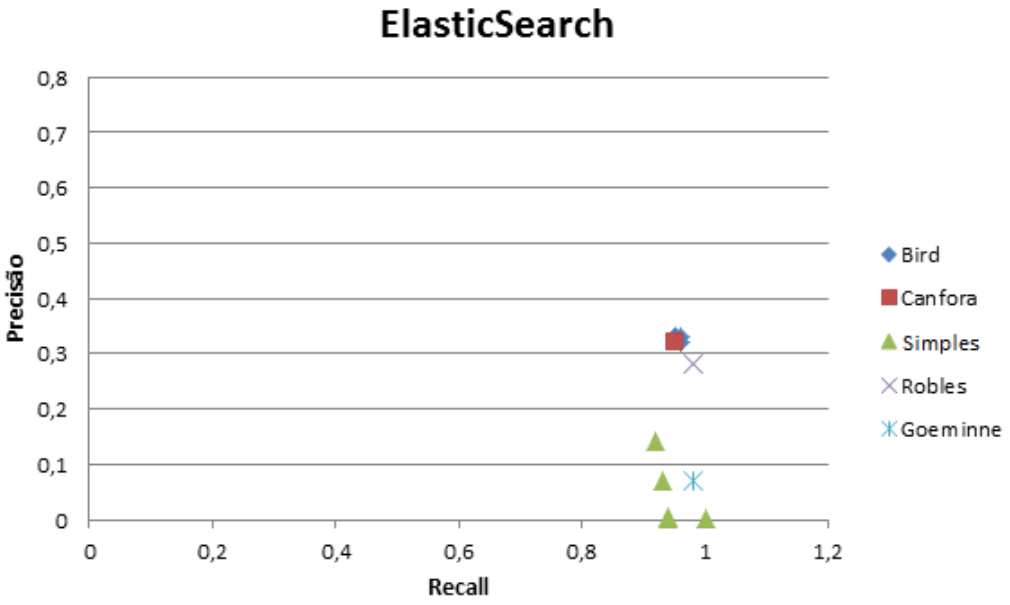

Figura 3: Representação da interação entre desenvolvedores e suas contribuições no Elasticsearch

Resultados apontam que as melhores heurísticas para se utilizar em um projeto de nível médio ou grande são a Bird, Canfora e Robles, enquanto em um projeto pequeno a heurística de Goeminne também apresenta bons resultados. No entanto, devido a baixa precisão faz-se necessário verificar dentre as ambiguidades detectadas pela heurística selecionada, se existem falso-positivo entre as identidades.

Ameaças à Validade. Apesar do minucioso cuidado na criação da base de referência de cada projeto, ela pode conter erros e influenciar no resultado comparativo de cada heurística. Além disso, as heurísticas foram implementadas baseada nas idéias dos artigos onde foram originalmente publicadas e adaptadas para o contexto do GitHub, o que não garante a total integridade das ideias propostas pelos autores. O número de projetos selecionados, bem como as características próprias desses projetos podem, porventura, ter influenciado os resultados obtidos e a análise comparativa realizada posteriormente.

7. Conclusões

A detecção de ambiguidade de autores em um projeto é um importante passo no tratamento dos dados coletados em repositórios como o GitHub. O objetivo do trabalho apresentado neste artigo foi implementar e analisar de forma comparativa cinco heurísticas que detectam ambiguidade de contribuidores. Para isso foram selecionados três projetos do GitHub: Joda-Time, PowerShell e ElasticSearch. Criamos três bases de referência, uma para cada projeto analisado, bem como a implementação de cada uma das heurísticas e de um algoritmo para validação dos resultados obtidos pelas heurísticas com os dados da base de referência.

Os três projetos apresentaram para todas as heurísticas ótimos valores de *recall*, o que nos permite concluir que as heurísticas tendem a detectar todas as ambiguidades do conjunto de dados em análise. No entanto, a precisão variou de acordo com o tamanho do projeto, o Joda-Time que possui poucos autores, apresentou uma precisão de 100%, enquanto os projetos de médio e grande porte PowerShell e ElasticSearch, resultaram em precisões de 74% e 33%, respectivamente.

Como trabalhos futuros, pretendemos: (1) analisar um conjunto maior de projetos do GitHub para obter uma conclusão madura e precisa da melhor heurística a ser usada para cada caso; e (2) implementar uma heurística que diminua o número de falsos-positivos para softwares com um grande número de autores e ambiguidades, para com isso obter uma precisão que transpareça maior confiança para os pesquisadores que necessitam dessas heurísticas.

Referências

Bird, C., Gourley, A., Devanbu, P., Gertz, M., and Swaminathan, A. (2006). Mining email social networks. In *Proceedings of the 2006 international workshop on Mining software repositories*, pages 137–143. ACM.

Canfora, G., Cerulo, L., Cimitile, M., and Di Penta, M. (2011). Social interactions around cross-system bug fixings: the case of freebsd and openbsd. In *Proceedings of the 8th working conference on mining software repositories*, pages 143–152. ACM.

Goeminne, M. and Mens, T. (2011). A comparison of identity merge algorithms for software repositories. *Science of Computer Programming*, 78(8):971–986.

Kouters, E., Vasilescu, B., Serebrenik, A., and van den Brand, M. G. (2012). Who's who in gnome: Using lsa to merge software repository identities. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*, pages 592–595. IEEE.

Robles, G. and Gonzalez-Barahona, J. M. (2005). Developer identification methods for integrated data from various sources. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5.

Wiese, I. S., da Silva, J. T., Steinmacher, I., Treude, C., and Gerosa, M. A. (2016). Who is who in the mailing list? comparing six disambiguation heuristics to identify multiple addresses of a participant. In *Software Maintenance and Evolution (ICSME), 2016 IEEE International Conference on*, pages 345–355. IEEE.