Modularidade de Código X Propriedade Coletiva: Um Estudo Empírico sobre Projetos Populares de Código Aberto

Igor Muzetti Pereira, Carla Sanches Nere dos Santos, Vicente Jośe Peixoto de Amorim

[1]Departamento de Computação e Sistemas (DECSI) – Universidade Federal de Ouro Preto (UFOP)
Minas Gerais – MG – Brasil

igormuzetti@ufop.edu.br, carlasanches8@gmail.com, vjpamorim@gmail.com

Resumo. *A propriedade coletiva de código é uma prática de desenvolvimento de software comum em sistemas de código aberto. Ao longo da construção do código, cria-se uma rede de contribuição, uma associação entre desenvolvedores e os módulos do sistema. Este artigo apresenta as Matrizes Simples, uma abordagem para relacionar módulos de sistemas de software aos seus principais contribuidores, por meio de informações extraídas de seus códigos-fontes. Essas matrizes fornecem uma maneira simples e rápida de raciocinar sobre responsabilidades e módulos. Uma ferramenta para recuperar tal matriz é apresentada. Além disso, doze exemplos e aplicações das matrizes nos projetos de código aberto do GitHub são discutidos demostrando sua aplicabilidade.*

Abstract. *Collective code ownership is a standard software development practice in open source systems. Throughout the construction of the code, a contribution network is created, an association between developers and the system modules. This article presents the Simple Matrices, an approach to relate software system modules to their main contributors, through information extracted from their source codes. These matrices provide a simple and quick way to reason about responsibilities and modules. A tool to recover such an array is presented. Also, twelve examples and applications of the matrices in GitHub's Open-Source Projects are discussed demonstrating their applicability.*

1. Introdução

Sistemas de software são organizados em módulos, de forma a reduzir o tempo de desenvolvimento, facilitar o gerenciamento do projeto e sua evolução [Parnas 1972]. No entanto, sabe-se que a divisão em módulos de um sistema também pode ser afetada pela organização de sua equipe de desenvolvedores, conforme enunciado por Conway, no que ficou conhecido como Lei de Conway [Conway and Spandorfer 1968].

Representações de sistemas de software normalmente não consideram o fator humano. Por exemplo, representações como diagramas de classe e de componentes arquiteturais não incluem informação alguma sobre as equipes de desenvolvedores. Consequentemente, os modelos de software oferecem uma baixa rastreabilidade entre a estrutura modular do código e os desenvolvedores responsáveis pela sua implementação, manutenção e evolução. Essa rastreabilidade é fundamental em diversos cenários. Por exemplo, quando uma falha crítica é reportada para um determinado módulo, deve-se rapidamente identificar os responsáveis pela manutenção e evolução do mesmo. Um segundo cenário seria mitigar o *Truck Factor*, descobrindo módulos mantidos por apenas

um desenvolvedor e que, portanto, poderão ser seriamente afetados caso o mesmo abandone o projeto [Avelino et al. 2016]. O TF é uma analogia ao número de pessoas em uma equipe que devem ser atropeladas por um caminhão antes do sistema enfrentar problemas. Ou seja, o quanto um programa está preparado para a rotatividade de membros da equipe de desenvolvimento.

Sendo assim, o objetivo deste trabalho é analisar através do código-fonte e entender a propriedade coletiva do código, no que diz respeito à modularidade de software no contexto de sistemas de código aberto (open source systems - OSS) do ponto de vista de contribuidores. Este objetivo foi decomposto em duas Questões de Pesquisa (QPs): QP1. Os módulos de OSS costumam ter contribuidores que podem ser considerados responsáveis por eles? QP2. São encontrados contribuidores que são responsáveis por vários módulos no mesmo OSS e um mesmo módulo pode possuir vários responsáveis?

Este trabalho apresenta através de matrizes simples as responsabilidades dos contribuidores pelos módulos dos projetos. As linhas dessas matrizes representam os módulos de um sistema e as colunas os seus responsáveis. Na implementação atual, utiliza-se informações sobre o número de *commits* de um módulo para definir os seus responsáveis. Particularmente, considera-se que os responsáveis por um módulo são aqueles desenvolvedores que em conjunto contribuem com 80% dos *commits* realizados nos arquivos desse módulo. Além de definir e formalizar o conceito dessas matrizes, foi desenvolvida uma ferramenta em Java que extrai informações de repositórios de software Git e exporta os resultados através dessas matrizes. Finalmente, usamos essa ferramenta para extrair e analisar doze projetos populares de código aberto que possuem mais de cinquenta contribuidores e mais de mil estrelas no GitHub.

Foi identificado que os projetos OSS costumam ter contribuidores que são responsáveis por alguns módulos dos sistemas. Também foi verificado que um mesmo módulo pode possuir diferentes responsáveis, bem como um contribuidor pode ser responsável por diferentes módulos. Em suma, os autores perceberam que um OSS não possui muitos responsáveis, apesar de muitos contribuidores.

O restante deste artigo está dividido como segue. A Seção 2 apresenta o conceito das matrizes simples. A Seção 3 explica a ferramenta desenvolvida e usada para mineração dos repositórios. Na Seção 4 são apresentados alguns exemplos e é discutido a aplicabilidade das matrizes. Os resultados são discutidos na Seção 5. Os trabalhos relacionados são discutidos na Seção 6. A Seção 8 possui as considerações finais e as limitações deste estudo constam na Seção 7.

2. Matrizes Simples

Uma Matriz Simples neste trabalho consiste em uma matriz onde as linhas representam os módulos de um sistema e as colunas representam os desenvolvedores responsáveis pelos mesmos. Cada célula indica a porcentagem de responsabilidade que um determinado desenvolvedor tem sobre um determinado módulo. Essa porcentagem de responsabilidade corresponde à quantidade de *commits* de autoria desse desenvolvedor em arquivos desse módulo, em relação ao total de *commits* em arquivos desse módulo, como demonstra a equação 1:

$$r(d,m) = \frac{\#\{x \in C(m) \mid a(x) = d\}}{\#C(m)} \times 100\% \quad (1)$$

Nesta equação, r(**d**,**m**) é a porcentagem de responsabilidade do desenvolvedor **d** sobre o módulo **m**, C(**m**) é o conjunto de todos os *commits* para arquivos do módulo **m** e a(**x**) é o autor do *commit* **x**.

É importante considerar que um mesmo *commit* pode afetar arquivos de vários módulos. Assim, é necessário analisar os arquivos que tiveram alterações em cada *commit*. Para definir os desenvolvedores responsáveis pela manutenção de um módulo, utilizamos a *Commit-Based Heuristic* [Mockus et al. 2002]. Essa ferramenta permite identificar os desenvolvedores principais de um projeto. De acordo com essa heurística, os desenvolvedores são ordenados de maneira decrescente pelo número de *commits*, sendo que os desenvolvedores principais são os primeiros desenvolvedores cuja soma dos *commits* atinge 80%. Entretanto, adaptamos essa heurística para evitar que desenvolvedores com pouca quantidade de *commits* façam parte do conjunto de desenvolvedores principais de um determinado módulo. Para tanto, desconsideramos aqueles desenvolvedores que possuem menos de 5% do total de *commits* de um módulo [Coelho et al. 2018].

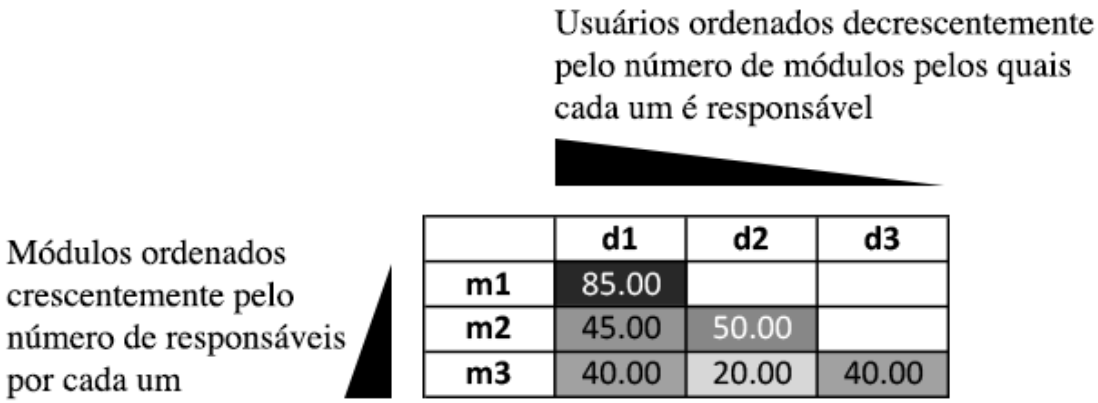

Figura 1. Exemplo de Matriz Simples

Como ilustrado pela Figura 1, as linhas da matriz são dispostas em ordem crescente, de acordo com o número de responsáveis pelo módulo. Já as colunas são dispostas em ordem decrescente, pelo número de módulos pelos quais o desenvolvedor é responsável. Adota-se uma escala de cinza para as células, variando do branco (5%) ao preto (100%).

3. Ferramenta

A ferramenta desenvolvida para extração das matrizes foi implementada na linguagem Java. Tal ferramenta utilizou o arcabouço Repodriller [Aniche 2014]. Com este arcabouço, foi possível extrair a identificação e a quantidade de *commits* dos desenvolvedores sobre os módulos de cada sistema nos repositórios Git.

Para minerar os repositórios, foi realizado um clone do mesmo para o ambiente local. A ferramenta recebe como parâmetros de entrada um identificador do repositório, o

caminho para o repositório em ambiente local, e os caminhos dos módulos no repositório. A identificação dos caminhos para os módulos dos repositórios foi realizada de maneira manual, através de análises dos diretórios da estrutura dos projetos. Diretórios com código de terceiros e de teste não foram considerados. A saída do processamento é um arquivo da matriz em formato CSV. A ferramenta está publicamente disponível para uso e pode ser encontrada em: [https://github.com/carlasanches/ownership-matrice.git].

4. Metodologia

O desenvolvimento deste trabalho pode ser dividido em duas etapas principais: (1) seleção de softwares a serem analisados e (2) mineração dos softwares selecionados com o auxílio do arcabouço RepoDriller. Ao final do processo foram obtidas tabelas com as propriedades de código para cada projeto analisado.

4.1. Primeira Etapa: Seleção dos Projetos

Foram doze repositorios de alta popularidade na comunidade GitHub, sendo todos com mais de mil estrelas e cinquenta contribuidores. Foram escolhidos projetos em Java, por se tratar de uma das linguagens mais populares no GitHub. Para facilitar a pesquisa pelos repositórios, foi utilizada a ferramenta GitTrends[1]. A ferramenta desenvolvida por Avelino [Avelino et al. 2016] foi criada com o propósito de apresentar o *Truck Factor (TF)* de softwares do GitHub. Quanto menor o número, mais alta a dependência de contribuidores específicos [Avelino et al. 2016]. Porém, foi utilizada nesse projeto com o intuito de filtrar repositórios pelos critérios definidos. As ferramentas utilizadas são aquelas indicadas pelo *link* do GitHub no rodapé da página. Foi usado a última versão de cada no final do ano de 2018. Ao final do processo, os seguintes projetos foram escolhidos: Apache Cassandra[2], Apache Hadoop[3], Apache Maven[4], Eclipse Che[5], Eclipse Openj9[6], ElasticSearch[7], Google Guava[8], IntelliJ IDEA Community Edition[9], Junit 4[10], Mockito[11], Pentaho Kettle[12] e Spring Framework[13].

4.2. Segunda Etapa: Mineração de softwares selecionados com o RepoDriller

O RepoDriller[14] é um arcabouço Java *código aberto* para mineração de repositórios Git. Com ele é possível extrair informações de commits, desenvolvedores, modificações, diffs e código-fonte. O RepoDriller foi utilizado neste trabalho para gerar bases de dados que contêm informações necessárias para a criação das tabelas. Essas bases de dados tratam-se de arquivos em *Comma Separated Values (CSV)* exportados pelo

[1]http://gittrends.io
[2]https://github.com/apache/cassandra
[3]https://github.com/apache/hadoop
[4]https://github.com/apache/maven
[5]https://github.com/eclipse/che
[6]https://github.com/eclipse/openj9
[7]https://github.com/elastic/elasticsearch
[8]https://github.com/google/guava
[9]https://github.com/JetBrains/intellij-community
[10]https://github.com/junit-team/junit4
[11]https://github.com/mockito/mockito
[12]https://github.com/pentaho/pentaho-kettle
[13]https://github.com/spring-projects/spring-framework
[14]https://github.com/mauricioaniche/repodriller

arcabouço ao fim da mineração de cada repositório. A Tabela 1 exemplifica as bases de dados obtidas, apresentando parte da saída do RepoDriller para o projeto Eclipse Openj9, com as 15 primeiras modificações. Os nomes dos diretórios e nomes e e-mails de desenvolvedores foram ocultados para melhor visualização dos dados. Na coluna ”Diretório Antigo”, /dev/null é um nome padrão para arquivos que ainda não haviam sido criados, portanto não possuem um nome antigo.

Tabela 1. Resultado da mineração do projeto Eclipse Openj9 utilizando o RepoDriller

Hash do commit	Tipo de Modificação	Diretório Atual	Diretório Antigo	Nome do Autor	E-mail do Autor	Nome do Committer	E-mail do Committer
1795753859	MODIFY	dir/arquivo1.java	dir/arquivo1.java	Dev 1	dev1@email.com	Dev 1	dev1@email.com
1795753859	ADD	dir/arquivo2.java	/dev/null	Dev 1	dev1@email.com	Dev 1	dev1@email.com
1795753859	ADD	dir/arquivo3.java	/dev/null	Dev 1	dev1@email.com	Dev 1	dev1@email.com
1795753859	ADD	dir/arquivo4.java	/dev/null	Dev 1	dev1@email.com	Dev 1	dev1@email.com
1795753859	MODIFY	dir/arquivo5.java	dir/arquivo5.java	Dev 1	dev1@email.com	Dev 1	dev1@email.com
1795753859	ADD	dir/arquivo6.java	/dev/null	Dev 1	dev1@email.com	Dev 1	dev1@email.com
1795753859	ADD	dir/arquivo7.java	/dev/null	Dev 1	dev1@email.com	Dev 1	dev1@email.com
1795753859	MODIFY	dir/arquivo8.java	dir/arquivo8.java	Dev 1	dev1@email.com	Dev 1	dev1@email.com
1471370558	MODIFY	dir/arquivo9.java	dir/arquivo9.java	Dev 2	dev2@email.com	Dev 2	dev2@email.com
1510214634	MODIFY	dir/arquivo9.java	dir/arquivo9.java	Dev 3	dev3@email.com	Dev 5	dev5@email.com
1760226975	MODIFY	dir/arquivo10.java	dir/arquivo10.java	Dev 4	dev4@email.com	Dev 4	dev4@email.com
1760226975	RENAME	dir/arquivo11.java	dir/arq11.java	Dev 4	dev4@email.com	Dev 4	dev4@email.com
1760226975	RENAME	dir/arquivo12.java	dir/arq12.java	Dev 4	dev4@email.com	Dev 4	dev4@email.com
1760226975	RENAME	dir/arquivo13.java	dir/arq13.java	Dev 4	dev4@email.com	Dev 4	dev4@email.com
1760226975	RENAME	dir/arquivo14.java	dir/arq14.java	Dev 4	dev4@email.com	Dev 4	dev4@email.com

5. Resultados

Cada projeto gerou um arquivo .CSV e então manualmente uma matriz foi construída para melhor visualização dos módulos e responsáveis. A Figura 2 é um exemplo de parte de uma matriz simples para o OpenJ9. Pela matriz da Figura 2 pode-se verificar que o desenvolvedor d8 possui 100% da responsabilidade do módulo m1. Isso mostra um caso extremo onde apenas um desenvolvedor possui commits para o módulo, o que indica um alto risco de manutenibilidade caso o desenvolvedor não contribua mais com o projeto. Outra observação pode ser feita em relação ao módulo m6. Nota-se que foram identificados três responsáveis (d1, d3 e d5), que juntos somam 48,42% da responsabilidade, ou seja, abaixo dos 80% requeridos para caracterizar responsáveis. Isso acontece porque a responsabilidade é distribuıda em uma maior quantidade de desenvolvedores, que foram desconsiderados por serem responsáveis por menos que 5% dos *commits* do módulo.

	d1	d2	d3	d4	d5	d6	d7	d8	d9	d10	d11	s12	d13	d14	d15
m1								100.00							
m2	30.89	63.35													
m3	67.94				10.38									6.23	
m4	42.03			37.68									7.61		
m5						32.26			19.35		32.26				
m6	7.69		31.12		9.61										
m7	16.74	16.74	15.81				16.28								15.81
m8	10.34	10.34		17.24						13.79		31.03			

Figura 2. Exemplo de Matriz Simples para o OpenJ9

Após gerar as tabelas e matrizes de cada sistema, foi realizada uma análise de seus resultados, visando responder às questões propostas. A Tabela 2 apresenta os resultados

para cada sistema analisado. A coluna Contrib refere-se ao número de contribuidores de cada projeto, Resp significa o número de responsáveis, RespMod (média) é a média de responsáveis que cada módulo possui, e ModResp (média) é, em média, o número de módulos pertencentes a cada responsável.

Tabela 2. Resultados da análise da propriedade de código dos sistemas selecionados

Nome do Projeto	Contrib	Resp	RespMod (média)	ModResp (média)
Apache Cassandra	274	13	1	3
Apache Hadoop	148	21	2	1
Apache Maven	68	4	1	3
Eclipse Che	106	14	1	3
Eclipse Openj9	103	6	1	2
ElasticSearch	1107	22	1	5
Google Guava	167	1	1	19
IntelliJ IDEA[5]	358	9	1	2
Junit 4	139	3	1	3
Mockito	131	1	1	3
Pentaho Kettle	162	24	2	2
Spring Framework	317	2	1	5

A partir da análise dos dados da Tabela 2 é possível responder as questões de pesquisa.

QP1. Os módulos de OSS costumam ter contribuidores que podem ser considerados responsáveis por eles? A determinação dos contribuidores que se tornam responsáveis pelos módulos pode ser realizada considerando o resultado do cálculo da propriedade de código. Foi utilizada uma convenção que determina que um desenvolvedor é responsável por um módulo quando responsável por 80% dos arquivos desse módulo e se tiver um percentual mínimo de 5% de arquivos sob sua responsabilidade. Esses cálculos foram feitos pela ferramenta. Sendo assim, foram selecionados os contribuidores que apresentavam valores iguais ou acima de 80% sobre os módulos. Para os repositórios em que não haviam contribuidores que apresentavam esse valor, e para os contribuidores que possuíam uma porcentagem muito próxima de 80%, foram incluídos aqueles cuja propriedade era a maior de um módulo, desde que acima dos 5%. Essa última etapa foi realizada a fim de abranger o máximo de contribuidores que demonstravam certa representatividade nos *commits* do repositório. Então, sim, os módulos de um OSS costumam ter responsáveis por eles. Dentre os projetos analisados, o Pentaho Kettle, Apache Hadoop e o Eclipse Che apresentaram os maiores índices 14,8%, 14,1% e 13.2%, respectivamente. Contudo, outros projetos dessa amostra apresentaram módulos com menos de 1% de propriedade de código, são eles: Google Guava, Spring Framework e Mockito com 0,05%, 0,06% e 0,07%, respectivamente. Ou seja, em geral existem muito

mais contribuidores que não são responsáveís pelos módulos do que responsáveis em um OSS. Neste caso de OSS, alguns contribuidores podem ajudar no reparo de partes particulares dos sistemas, muitas vezes por conta própria ou indicados por algum membro ativo da comunidade. Nesta amostra de doze projetos populares, um pequeno número de contribuidores é responsável pela grande maioria do desenvolvimento significativo do OSS. Por isso, acredita-se que as chances disso acontecer em um universo maior de OSS é alta, uma vez que isso já faz parte de uma convenção entre as comunidades de OSS. Também é conhecido dentro do desenvolvimento de OSS que o estereótipo popular de uma comunidade ampla se unindo para criar software pode depender inclusive de alguns poucos contribuidores pagos pelos fornecedores e empresas que prestam consultoria na implantação desses OSS. Sumarizando, embora a ampla comunidade em torno dos projetos possa ser grande, veja a quantidade de responsáveis em relação a quantidade de contribuidores de acordo com a Tabela 2, o número de contribuidores principais significativos é relativamente pequeno. Esse padrão parece comum em muitos projetos de código aberto.

QP2. É comum em um projeto OSS um módulo possuir vários responsáveis e um contribuidor ser responsável por vários módulos? Em média, a maioria dos projetos possui apenas um responsável por módulo. Considerando que existem outros contribuidores que não são responsáveis trabalhando nesses módulos, todos teriam ao menos um contribuidor com maior conhecimento do código, supervisionando as mudanças e liderando a equipe. Possuir mais contribuidores responsáveis por módulos pode ser útil no caso de algum deles desistir do projeto. Porém, conforme Greiller et al. (2015), a falta de um proprietário bem definido pode fazer com que as responsabilidades de manutenção do código sejam deixadas de lado, tornando-o mais sujeito a falhas [Greiler et al. 2015]. E um contribuidor pode ser responsável por vários módulos, analisando a Tabela 2, as médias variaram entre 1 e 19 módulos por responsável. Destacou-se o Google Guava, que possui apenas um responsável, e 19 módulos sob sua responsabilidade. Esses 19 módulos foram modificados por 10 ou mais desenvolvedores. O projeto possui, então, uma responsabilidade bem definida, porém uma grande quantidade de módulos sob responsabilidade de apenas um contribuidor. Esse pode ser considerado um caso de TF baixo, em que o projeto não está preparado para a rotatividade de membros da equipe. No restante dos projetos, as médias mais altas são do ElasticSearch e do Spring Framework. O Spring Framework apresenta uma situação semelhante à do Google Guava, com a responsabilidade distribuída entre apenas dois desenvolvedores. Já o ElasticSearch pode ter desenvolvedores sobrecarregados, porém com mais flexibilidade para a distribuição do trabalho, visto que o projeto possui mais de um responsável.

6. Trabalhos Relacionados

Modularidade é um mecanismo usado para melhorar a flexibilidade e a compreensibilidade de um sistema, ao mesmo tempo que permite a redução do tempo de desenvolvimento [Parnas 1972]. Através dos módulos, a indústria consegue construir subsistemas menores que podem ser projetados independentemente, mas que funcionam juntos como um todo. Para otimizar processos de desenvolvimento de produtos, o conceito de *DSM - Dependency Structure Matrix* foi criado. DSM apresenta uma matriz simples que permite uma melhor visualização das dependências entre tarefas diante dos módulos de um sistema [Baldwin and Clark 1999]. Sangal et al. apresentam uma abordagem para gerenciar

a arquitetura de grandes sistemas de software. No trabalho destes autores, dependências são extraıdas do código por uma análise estática convencional e mostradas através de DSMs [Sangal et al. 2005].

No desenvolvimento de software existem desenvolvedores líderes que assumem uma maior responsabilidade nos projetos. Isso acontece inclusive nas comunidades de código aberto. Yamashita et al. (2015) sugere que o princípio de Pareto não é compatível com as equipes de muitos projetos do GitHub. Ou seja, não necessariamente 80% do software é desenvolvido por 20% dos desenvolvedores da equipe [Yamashita et al. 2015].

Nakakoji et al. (2002) em seu estudo, diferentemente da maioria dos estudos anteriores sobre evolução de software, examinam não apenas a evolução dos sistemas de código aberto, mas também a evolução das comunidades associadas. Os autores concluem que existem diferentes modelos de colaboração e, que essas diferenças resultam em diferentes padrões de evolução dos sistemas de código aberto e suas comunidades. Eles ainda afirmam que os principais desenvolvedores são responsáveis por gerenciar o desenvolvimento de um sistema de software [Nakakoji et al. 2002].

Um trabalho semelhante foi realizado por Avelino et al. (2016) para estimar o Truck Factor (TF) de 133 repositórios do GitHub. O cálculo foi realizado com base no histórico de commits, utilizando um modelo denominado *Degree-of-Authorship (DOA)*. O DOA foi proposto por Fritz et al. (2010) como um método para identificar o conhecimento de desenvolvedores acerca de certas partes do código, e, consequentemente, a propriedade de código [Fritz et al. 2010].

Também em uma abordagem baseada em *commits*, o estudo de Greiler et al. (2015) tem o objetivo de confirmar se a propriedade de código influencia diretamente em sua qualidade. Foi realizada uma investigação nos produtos da Microsoft: Office, Office365, Windows e Exchange. A análise de dados foi realizada a nível de diretórios de código (agrupamento de arquivos que contêm propriedades em comum). Eles oferecem um nível de granularidade intermediário, proporcionando uma melhor análise de dados em relação a erros. Uma granularidade muito alta traz um número de erros muito grande. Da mesma forma, uma granularidade pequena faria com que o número de erros detectados fosse muito baixo, distorcendo os dados. Os contribuidores foram classificados em contribuidores maiores e menores. Foi considerado um contribuidor maior, aquele que realizou no mínimo 50% das mudanças em um código. Aqueles que realizaram menos de 50% das mudanças foram considerados contribuidores menores [Greiler et al. 2015].

Em comparação com os trabalhos correlatos, neste trabalho foi utilizado o modelo DOA para cálculo de propriedade de código. O modelo proporciona uma boa estimativa da propriedade de código para diferentes sistemas, além de respeitar o fluxo de experiência do desenvolvedor. Isso acontece porque o grau de conhecimento de um desenvolvedor sobre um determinado arquivo aumenta quando este confirma as alterações no repositório de origem e diminui quando outros desenvolvedores fazem alterações.

Diferente do estudo de Avelino (2016), este estudo abrange um número maior de desenvolvedores como parte dos principais contribuidores ao analisar módulos mais específicos dos sistemas. Além disso, também investiga quais são os módulos em que os desenvolvedores contribuem e as suas porcentagens de contribuição. De forma semelhante ao estudo de Greiler et al. (2015), este trabalho realiza a análise de diretórios,

porém o nível de granularidade é definido pelo número de desenvolvedores que os alteram. A classificação dos contribuidores foi realizada de maneira diferente, utilizando como base o modelo de DOA e 80% das mudanças do código para identificar um contribuidor principal. O DOA foi elaborado para selecionar os proprietários de módulos de um sistema. Sendo assim, esta se mostrou uma estratégia mais adequada para selecionar os contribuidores responsáveis. Apesar do modelo DOK (*Degree-of-Knowledge*) ser mais completo. Diferente do estudo de Fritz et al. (2010), no contexto deste trabalho não é possível identificar quando um desenvolvedor acessou um arquivo. Portanto, foi utilizado somente o método DOA. Com o intuito de encontrar desenvolvedores responsáveis para todos os principais módulos, também foi analisado todo o histórico de *commits*.

7. Limitações

Estão documentadas nessa seção possíveis ameaças à validade do estudo e alguns vieses que podem ter afetado os resultados. Também são explicadas ações para atenuá-las [Wohlin et al. 2012].

Validade Externa. Os resultados obtidos estão restritos a doze projetos Java. Porém esse risco é atenuado pelo fato de os projetos serem armazenados no GitHub, o repositório de código mais popular atualmente, e de fácil acesso.

Validade Interna. Na ferramenta de geração das matrizes é necessário que o usuário passe como parâmetro de entrada a lista de módulos que foram identificados. Isso exige que o usuário da ferramenta deduza uma arquitetura a partir de diretórios do repositório, o que nem sempre é trivial, visto que a modularização de um projeto pode ser radicalmente diferente da de outro. Outra limitação é que as matrizes tiveram que ser formatadas manualmente para melhorar sua visualização, uma vez que a ferramenta apenas gera um arquivo CSV com os valores das porcentagens de responsabilidade em cada célula. É necessário um aprimoramento para a geração de matrizes já inteiramente prontas, que não necessitem de nenhuma formatação para visualização.

Validade de Construção. Considerando que todas as informações obtidas foram através da mineração e análise do histórico de *commits*, a maioria dos softwares sofreu muitas mudanças em sua estrutura ao longo dos anos de desenvolvimento, e uma quantidade elevada de commits antigos pode ocultar o trabalho atual do resultado final. Não existe uma data correta para iniciar a análise, mas o ideal seria que o algoritmo considerasse o trabalho de contribuidores mais antigos sem diminuir a propriedade de contribuidores mais recentes. Também devido aos *commits* mais antigos, desenvolvedores que já deixaram a equipe podem ser classificados pelo algoritmo como parte dos desenvolvedores principais.

Validade de Conclusão. As conclusões do estudo estão sujetas à confiabilidade das métricas escolhidas para responder as questões de pesquisa apresentadas. Para mitigar essa limitação, as métricas de cálculo da propriedade de código, TF e DOA são comumente usadas na literatura.

8. Considerações Finais

Neste trabalho foi apresentado o conceito de Matrizes Simples, que oferece uma maneira simples e rápida de visualizar a relação da responsabilidade entre os módulos

de um sistema, extraıdos do código-fonte, e os principais contribuidores de cada um. A visualização apresentada é uma proposta para melhor explicitar o fator humano em representações de software, com relação a sua estrutura arquitetural.

Os contribuidores são os responsáveis pela implementação, manutenção e evolução do software. Ter uma visualização rápida que mostre os principais responsáveis por cada área do código, pode ter grande valia para o gerenciamento e acompanhamento da evolução de projetos de software.

Foi desenvolvida uma ferramenta para o cálculo dessas matrizes a partir de repositórios reais de software de código aberto. Foram escolhidos doze repositórios de alta popularidade, com mais de mil estrelas e cinquenta contribuidores no GitHub.

Além destas contribuições, neste trabalho foi identificado que os projetos OSS costumam ter contribuidores que são responsáveis por alguns módulos dos sistemas. Isto foi possível devido ao grande número de modificações em diferentes arquivos que eles fazem através dos *commits* dos projetos. Como foram considerados os *masters* de cada projeto, é possível que um contribuidor possa ter feito alterações de outros pequenos contribuidores. Contudo, foi considerado que se no universo OSS, este contribuidor de alguma maneira é revisor do *master*, ou seja, exerce uma certa autoridade, ele pode ser considerado de certa maneira como responsável pelas alterações.

Outra contribuição foi a identificação que um mesmo módulo pode possuir diferentes responsáveis, bem como um contribuidor pode ser responsável por diferentes módulos. Em geral, foi percebido que um OSS não possui muitos responsáveis, apesar de muitos contribuidores. Isso pode se justificar pela contratação destes poucos por empresas fornecedoras e consultoras dos softwares. Ou ainda a estrutura da equipe que conta com alguns revisores que se tornam responsáveis por alguns módulos e são os donos de alguns *commits* grandes, contendo muitas alterações feitas até por pequenos contribuidores. Devido a grande concentração da responsabilidade dos módulos com poucos contribuidores, o que pode ser um indicativo de um *Truck Factor* baixo. Isso significa que, se essas pessoas abandonarem a equipe de desenvolvimento, o sistema sofre risco de descontinuidade. Ainda assim, pode ser discutido que nos casos de módulos com uma quantidade baixa de desenvolvedores principais, um deles detêm maior responsabilidade, enquanto os outros dão apenas algum suporte pontual.

Como trabalhos futuros, a ferramenta implementada pode evoluir com a automação da seleção de diretórios que serão analisados por projeto. Outros projetos do GitHub também poderiam ser adicionados e novos cenários discutidos. Um *survey* pode ser realizado com os contribuidores e responsáveis dos projetos OSS para uma triangulação dos dados e verificação de hipóteses. A técnica usada pode futuramente propor um tratamento que levasse em conta a data dos *commits* e a maturidade do projeto. Desta nova maneira, os resultados da aplicação da técnica podem ser mais robustos e não serem impactados negativamente por mudanças estruturais que ocorrem nos projetos analisados ao longo dos anos.

Referências

Aniche, M. (2014). Repodriller. `https://github.com/mauricioaniche/repodriller`.

Avelino, G., Passos, L., Hora, A., and Valente, M. T. (2016). A novel approach for estimating truck factors. In *24th International Conference on Program Comprehension (ICPC)*, pages 1–10.

Baldwin, C. Y. and Clark, K. B. (1999). *Design Rules: The Power of Modularity Volume 1*. MIT Press, Cambridge, MA, USA.

Coelho, J., Valente, M. T., Silva, L. L., and Hora, A. (2018). Why we engage in FLOSS: Answers from core developers. In *11th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, pages 1–8.

Conway, M. E. and Spandorfer, L. M. (1968). A computer system designer's view of large scale integration. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), pages 835–845, New York, NY, USA. ACM.

Fritz, T., Ou, J., Murphy, G. C., and Murphy-Hill, E. (2010). A degree-of-knowledge model to capture source code familiarity. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, page 385–394, New York, NY, USA. Association for Computing Machinery.

Greiler, M., Herzig, K., and Czerwonka, J. (2015). Code ownership and software quality: A replication study. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, page 2–12. IEEE Press.

Mockus, A., Fielding, R. T., and Herbsleb, J. D. (2002). Two case studies of open source software development: Apache and mozilla. *ACM Trans. Softw. Eng. Methodol.*, 11(3):309–346.

Nakakoji, K., Yamamoto, Y., Nishinaka, Y., Kishida, K., and Ye, Y. (2002). Evolution patterns of open-source software systems and communities. In *Proceedings of the International Workshop on Principles of Software Evolution*, IWPSE '02, pages 76–85, New York, NY, USA. ACM.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12):1053–1058.

Sangal, N., Jordan, E., Sinha, V., and Jackson, D. (2005). Using dependency models to manage complex software architecture. *SIGPLAN Not.*, 40(10):167–176.

Wohlin, C., Runeson, P., Hst, M., Ohlsson, M. C., Regnell, B., and Wessln, A. (2012). *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated.

Yamashita, K., McIntosh, S., Kamei, Y., Hassan, A. E., and Ubayashi, N. (2015). Revisiting the applicability of the pareto principle to core development teams in open source software projects. In *Proceedings of the 14th International Workshop on Principles of Software Evolution*, IWPSE 2015, pages 46–55, New York, NY, USA. ACM.