Uma Abordagem para Verificação de Similaridade entre Sistemas Orientados a Objetos

Paloma Maira de Oliveira[1,2], Hudson Silva Borges[2]
Marco Tulio Valente[2], Heitor Augustus Xavier Costa[3]

[1] Instituto Federal de Minas Gerais (IFMG) - Formiga - MG - Brasil

[2]Departamento de Ciência da Computação da Universidade Federal de Minas Gerais (DCC/UFMG) - Belo Horizonte - MG - Brasil

[3]Departamento de Ciência da Computação da Universidade Federal de Lavras (DCC/UFLA) - Lavras - MG - Brasil

paloma.oliveira@ifmg.edu.br, hudsonsilbor@dcc.ufmg.br, mtov@dcc.ufmg.br, heitor@dcc.ufla.br

Abstract. *This paper presents a quantitative approach for identifying similarity among object-oriented systems. This approach shows three contributions: (a) a mechanism that provides reference values for a specific metric, considering different class profiles; (b) a mechanism to obtain a subset of similar systems from a portfolio of systems according to a specific software metric and using well-known clustering techniques; and (c) a mechanism to obtain subsets of similar systems according to a set of software metrics and using standard graph theory concepts. We also describe a supporting tool and the application of our approach in a corpus of 86 open-source systems, comprising around 12 million lines of code. As a result, we have been able to found a group of 15 systems with strong indications of similarity. Our claim is that such systems very probably present similar levels of internal quality.*

Resumo. *Neste artigo, apresenta-se uma abordagem quantitativa baseada em métricas de software para determinar similaridade entre sistemas orientados a objetos. Essa abordagem apresenta pelo menos as seguintes contribuições: (a) um mecanismo para obtenção de valores de referência para uma certa métrica de um sistema, em função dos diversos perfis de suas classes; (b) um mecanismo para obtenção de subconjuntos de sistemas similares dado um portfólio de sistemas, utilizando para isso de técnicas de clusterização; (c) um mecanismo para obtenção de subconjuntos de sistemas semelhantes, de acordo com um conjunto de métricas, utilizando para isso de conceitos de teoria de grafos. Adicionalmente, descreve-se no artigo uma aplicação real da abordagem proposta em um corpus de 86 sistemas de código aberto, totalizando cerca de 12 milhões de linhas de código. Os resultados obtidos mostram, por exemplo, que existe um grupo de 15 sistemas neste corpus com fortes indícios de similaridade. Nosso argumento é que tais sistemas apresentam provavelmente níveis bastante similares de qualidade interna.*

1. Introdução

A tecnologia de orientação a objetos é a mais encontrada em projeto (*design*) e implementação de sistemas legados e de sistemas em desenvolvimento. Por outro lado, passados mais de quatro décadas do lançamento dos princípios básicos dessa tecnologia -- incluindo o conceito de classes para modularização de dados e de operações --, pouco conhecimento quantitativo há sobre a estrutura interna de sistemas orientados a objetos [Concas 2007; Baxter 2006]. Mais precisamente, diversas métricas foram propostas para quantificar propriedades internas de tais sistemas, como tamanho, acoplamento, coesão e complexidade [Shyam *et al*., 1994]. No entanto, ainda existe certa carência de um corpo de conhecimento sólido que permita usar tais métricas para avaliar a qualidade e a estrutura interna desses sistemas.

Assim, o objetivo desse artigo é apresentar uma abordagem quantitativa baseada em métricas de software para determinar similaridade entre sistemas orientados a objetos. Essa abordagem possui três mecanismos como contribuições principais:

1. *Um mecanismo para obtenção do valor de referência para uma certa métrica de um sistema, em função dos diversos perfis de suas classes*. A definição do valor de referência para métricas de software em nível de classe não é trivial [Ferreira *et al*., 2011]. Por exemplo, a obtenção de um valor de referência (ou valor médio) para a quantidade de linhas de código para classes de um sistema é complexa. O principal motivo é a distribuição da quantidade de linhas de código das classes de um sistema seguir uma Lei de Potência (*power law*) [Newman, 2005; Louridas *et al*., 2008; Baxter *et al*., 2006]. Em geral, os sistemas possuem muitas classes com pequena quantidade de linhas de código (menos de uma centena de linhas) e poucas classes com grande quantidade de linhas de código (em alguns casos, até milhares de linhas) [Taube-Schock *et al*., 2011]. Assim, a média aritmética é pouco representativa, o que ocasiona certa dificuldade em analisar esses sistemas. Para contornar essa dificuldade, a abordagem proposta utiliza técnicas de clusterização [Tufféry, 2011] para obter valor de referência para perfis de classes. Assim, conjuntos (ou *clusters*) de classes homogêneas são identificados, o que favorece o cálculo de médias aritméticas um pouco mais representativas, pois obtém-se um valor de referência para diferentes perfis de classes em função de sua localização na curva característica de uma Lei de Potência. Esse valor difere de um valor de referência global para as classes de um sistema, pois elas estão organizadas em *clusters* representando classes com diferentes quantidades de linhas de código, por exemplo, classes pequenas, médias ou grandes;
2. *Um mecanismo para obtenção de subconjuntos de sistemas similares dado um portfólio de sistemas utilizando técnicas de clusterização, em função dos valores de referência de uma determinada métrica*. Sejam dois sistemas, S_1 e S_2, uma métrica, M, e um perfil, P. Esse mecanismo verifica a similaridade dos valores de referência de M para as classes de S_1 com perfil P e para as classes de S_2 com mesmo perfil. Por exemplo, verifica-se a similaridade entre as "classes grandes" de S_1 e as "classes grandes" de S_2;
3. *Um mecanismo para obtenção de subconjuntos de sistemas semelhantes de acordo com um conjunto de métricas*. Esse mecanismo estende o segundo mecanismo, pois utiliza simultaneamente valores de referência de duas ou mais

métricas como critério de similaridade, por exemplo métricas de tamanho, de complexidade e de acoplamento.

Em última instância, a abordagem proposta neste artigo visa a contribuir na verificação da qualidade interna de sistemas legados ou em desenvolvimento. Mais especificamente, argumenta-se que os valores de referências de métricas (que podem cobrir diferentes propriedades) para os perfis de classes de um sistema concebido segundo boas práticas de desenvolvimento podem ser utilizados como indicadores de averiguação da qualidade interna de outros sistemas. Por exemplo, seja um sistema S_1 desenvolvido por meio de um projeto de qualidade elaborado de acordo com os melhores princípios/conceitos da Engenharia de Software e por uma equipe de desenvolvedores de inequívoca competência. Logo, a probabilidade de S_1 ter elevada qualidade interna é alta. Suponha ainda que, utilizando a abordagem proposta neste artigo, tenha sido identificado que um sistema S_2 é similar a S_1. Assim, essa similaridade é um indicativo forte de que S_2 deve também possuir bons níveis de qualidade interna.

O restante deste artigo está organizado da seguinte forma: Conceitos necessários para entendimento da abordagem proposta são apresentados na Seção 2; A abordagem para verificar similaridade entre sistemas orientados a objetos é apresentada na Seção 3; O emprego dos mecanismos previstos nessa abordagem utilizando os sistemas presentes no repositório *Qualitas Corpus* – uma extensa coleção de sistemas cuidadosamente criada para condução de experimentos em Engenharia de Software – é apresentado na Seção 4; Alguns trabalhos relacionados estão resumidos na Seção 5; Conclusões, contribuições e sugestões de trabalhos futuros são discutidos na Seção 6.

2. Conceitos Básicos

A abordagem para verificar similaridade entre sistemas proposta neste artigo utiliza conceitos concernentes a Leis de Potência, Clusterização e Cliques, os quais são descritos brevemente nessa seção.

2.1. Leis de Potência

A análise da interação entre módulos de um sistema tem sido alvo de vários pesquisadores, cujos estudos sinalizam que sistemas, em geral, estão em conformidade com a família de distribuições de Leis de Potência (*power law*) [Newman, 2005; Louridas *et al*., 2008; Baxter *et al*., 2006]. Matematicamente, uma Lei de Potência refere-se a uma família de distribuições de probabilidade com características específicas, em que a probabilidade de uma variável aleatória *X* assumir um valor *x* é proporcional a uma potência negativa de *x*. Essa família de distribuição possui uma função de densidade de probabilidade da seguinte forma [Barabási; Albert, 1999; Newman, 2005; Clauset *et al*., 2009]:

$$\mathrm{p(X = x)\ \alpha\ Kx^{-\alpha}}$$

sendo $p(X = x)$ a probabilidade de encontrar o valor x, K uma constante e α um parâmetro da distribuição chamado de parâmetro de escala. Há várias abordagens para estimar o parâmetro de escala de uma Lei de Potência. A abordagem utilizada neste artigo foi visual, consistindo na construção de um histograma dos dados.

2.2. Clusterização

Clusterização é uma técnica para agrupar elementos similares de uma amostra de modo a formar *clusters* mutuamente excludentes [Tufféry, 2011]. Existem diferentes métodos para realizar esse agrupamento, sendo um dos mais conhecidos o método de partição (algoritmo *Kmeans*).

O algoritmo *Kmeans* agrupa os elementos em análise movendo-os entre *clusters* em diferentes etapas do seu processamento, na tentativa de agrupá-los de maneira próxima a uma alocação ótima [Tufféry, 2011]. Esse algoritmo particiona iterativamente os elementos para que a distância total entre os elementos de um mesmo *cluster* e o seu centro seja mínima (o centro de um *cluster* é a média de seus elementos). Cabe ressaltar que esse algoritmo clusteriza dados unidimensionais e multidimensionais.

2.3 Clique de um grafo

Uma clique em um grafo G não dirigido é um subgrafo completo de G. Assim, um subconjunto C de vértices é uma clique se tiver a seguinte propriedade: para todo par ordenado (v, w) de vértices em C, existe uma aresta ligando v e w. Uma clique C é máxima se não existe clique C' que seja maior que C [Cormen *et al*., 2009].

Exemplo: Seja o grafo G = (V, E) apresentado na Figura 1. O grafo G possui duas cliques: {1, 2, 5};{2, 3, 6, 7}. É possível observar que ambos conjuntos são subgrafos completo de G.

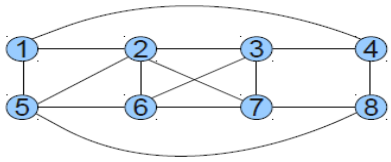


Figura 1 - Grafo com 8 Vértices (com duas cliques de tamanho 3 e 4)

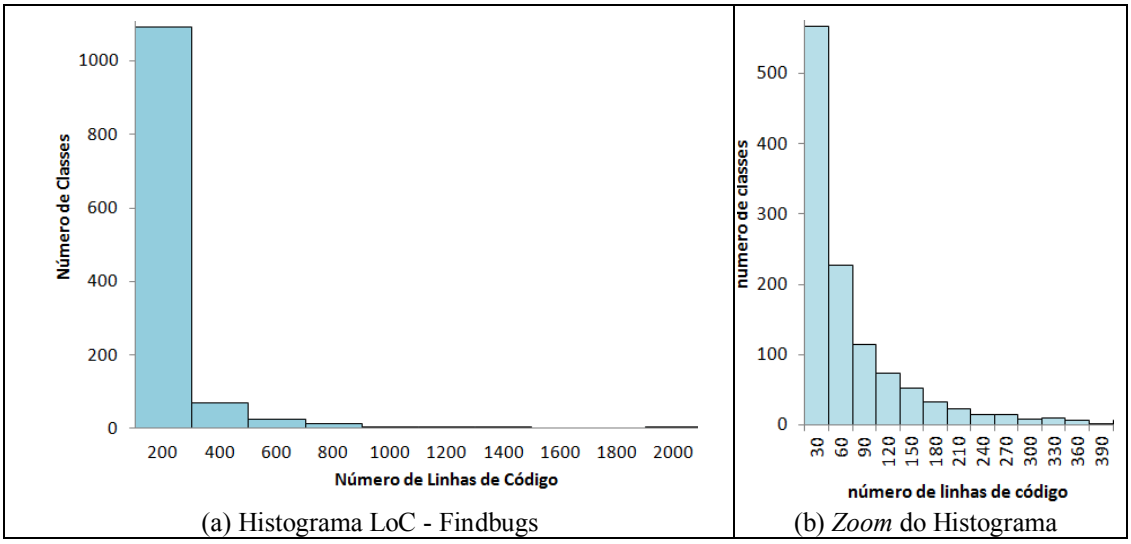
3. Verificação de Similaridade entre Sistemas

A contribuição central deste artigo consiste em uma abordagem quantitativa baseada em métricas de software para determinar a similaridade entre sistemas orientados a objetos. Essa abordagem inclui mecanismos para realizar as seguintes tarefas: (a) determinar valores de referência para perfis de classes; (b) Obter conjuntos de sistemas similares para uma métrica; e (c) Obter conjuntos de sistemas similares para um conjunto de métricas.

3.1. Determinar Valores de Referência para Perfis de Classes

Alguns trabalhos mostram que métricas para medir propriedades em nível de classe estão em conformidade com a família de distribuições que seguem uma Lei de Potência [Baxter *et al*., 2006; Ferreira *et al*., 2011]. Por exemplo, o histograma referente ao tamanho das classes do sistema `FindBugs`, em termos do número de linhas de código, é apresentado na Figura 2a. Pode-se observar na Figura 2b a presença de uma curva que caracteriza uma distribuição do tipo Lei de Potência (cauda longa). Além disso, observa-se que a maioria das classes possui menos do que 30 linhas de código; por

outro lado, há poucas classes com mais de 300 linhas de código. A quantidade média é de 82,6 linhas de código. No entanto, a média aritmética dessa métrica possui pouca representatividade, por causa do formato de cauda longa da distribuição da quantidade de linhas em nível de classe.



(a) Histograma LoC - Findbugs (b) *Zoom* do Histograma

Figura 2 - Tamanho das Classes do Sistema `Findbugs`

Assim, ao invés de utilizar um valor de referência (média aritmética global) das classes de um sistema, propõe-se a adoção de um mecanismo de clusterização – baseado no algoritmo *Kmeans* – para particionar automaticamente as classes em grupos, G_1, ..., G_n, conforme sua localização em diversas posições na curva, de acordo com uma determinada métrica. Sejam três grupos, G_1, G_2 e G_3, e uma métrica, M. O valor de referência em relação a M para esses grupos é obtido considerando o valor médio de M das classes presentes no grupo. Cabe ressaltar que a quantidade de classes a serem alocadas nos grupos deve ser homogênea (o que é garantido pelo algoritmo de clusterização). Por exemplo, G_1 possuirá classes com valores pequenos de M, G_2 possuirá classes com valores médios de M e G_3 possuirá classes com valores altos de M. Os valores pequeno, médio e grande são a distância euclidiana entre o valor de M da classe e o valor de referência dos grupos.

3.2. Obter Conjuntos de Sistemas Similares para uma Métrica

Esse mecanismo é utilizado para determinar subconjuntos (*clusters*) de sistemas com indícios de similaridade entre si para uma determinada métrica. Sejam um conjunto de sistemas, S, e uma métrica, M_1. Os valores de referências para perfis de classes de cada sistema em S são calculados em relação a M_1, gerando os grupos, G_1, ..., G_n (conforme descrito na seção anterior). A quantidade de grupos deve ser a mesma para os sistemas em S.

Como resultado, cada sistema em S tem o valor de referência dos seus grupos, os quais são fornecidos como entrada para o algoritmo *Kmeans*, a fim de obter grupos de sistemas similares, segundo o perfil de suas classes. Em outras palavras, em vez de clusterizar um valor unidimensional (como no mecanismo apresentado na seção

anterior), clusteriza-se um valor multidimensional correspondente aos valores de M para os perfis de classes G_1, ..., G_n de cada sistema em S. Quando o algoritmo *Kmeans* é utilizado, uma questão não trivial é determinar a quantidade de *clusters* (quantidade de subconjuntos de sistemas similares). Assim, recomenda-se executar o algoritmo diversas vezes para diferentes quantidades de *clusters* até obter coesão interna dos *clusters* superior a 90%.

Após a execução do algoritmo de clusterização, tem-se como saída os *clusters* de sistemas similares, sempre em relação aos valores de M. Isso significa que os sistemas alocados em um mesmo *cluster* possuem indícios de similaridade em relação a M.

3.3. Obter Conjuntos de Sistemas Similares para um Conjunto de Métricas

Esse mecanismo é utilizado para determinar subconjuntos (*clusters*) de sistemas com indícios de similaridade entre si para um (sub)conjunto de métricas, sendo uma extensão do mecanismo apresentando na seção anterior.

Assim, seja um conjunto de sistemas, S, e um conjunto de métricas, M. Inicialmente, o mecanismo descrito na seção anterior deve ser empregado para cada métrica em M. Além disso, propõe-se que a representação dos sistemas com indícios de similaridade para um conjunto de métricas seja realizada por meio de um grafo, cujos vértices são os sistemas e cujas arestas correspondem à quantidade mínima desejada de métricas em M que apontam indícios de similaridade entre dois sistemas. Por exemplo, sejam dois sistemas, S_i e S_j. Suponha que a quantidade de métricas para as quais foi detectada uma similaridade entre S_i e S_j seja dada por $Q_{m(i, j)}$. Suponha ainda que a quantidade mínima de métricas necessárias para decretar que dois sistemas são similares seja Q_{mmin}. Logo, S_i e S_j serão ligados por uma aresta caso $Q_{m(i, j)} >= Q_{mmin}$.

Após a criação desse grafo, os sistemas com indício de similaridade para um (sub)conjunto de métricas são indicados por cliques nesse grafo. Ressalte-se que esse grafo pode ter mais de uma clique, o que caracteriza a existência de diversos subconjuntos de sistemas com indícios de similaridade.

4. Estudo de Caso: Qualitas Corpus

O *Qualitas Corpus* [Tempero *et al*., 2010] é um repositório com 106 sistemas com código fonte aberto desenvolvidos na linguagem Java. Este repositório foi criado para ser utilizado em estudos empíricos em Engenharia de Software. O repositório inclui sistemas dos seguintes domínios: *3D/Graphics/Media*, *IDE*, *SDK*, *Database*, *Diagram/Visualisation*, *Tools*, *Games*, *Middleware*, *Parsers/Generators/Make*, *Programming Language* e *Testing*.

Cabe ressaltar que os sistemas presentes nesse repositório possuem várias versões, contudo a última versão disponível de cada sistema foi utilizada no escopo deste trabalho (versão 20101126r). Por causa da limitação de memória dos apoios computacionais utilizados, foram analisados 86 sistemas para ilustrar a abordagem proposta (Tabela 1). Ao todo, esses sistemas possuem em torno de 111 mil classes que totalizam mais de 12 milhões de linhas de código.

Tabela 1 - Sistemas Analisados

Ant	Exoportal	James	Jmeter	Mvnforum	Sablecc
Antlr	Findbugs	Jasml	Jmoney	Myfaces_core	Sandmark
Aoi	Freecol	Jasperreports	Joggplayer	Nakedobjects	Squirrel_sql
Argouml	Freecs	Javacc	Jparse	Nekohtml	Struts
Aspectj	Galleon	Jboss	Jpf	Openjms	Sunflow
Axion	Ganttproject	Jchempaint	Jrat	Oscache	Tapestry
C_jdbc	Gt2-2	Jedit	Jrefactory	Picocontainer	Tomcat
Castor	Heritrix	Jena	Jspwiki	Pmd	Velocity
Cayenne	Hibernate	Jext	Jsxe	Poi	Webmail
Cobertura	Htmlunit	Jfin_datemath	Jtopen	Pooka	Weka
Columba	Informa	Jgraph	Jung	Proguard	Xmojo
Displaytag	Ireport	Jgraphpad	Junit	Quartz	
Drawswf	Itext	Jgrapht	Log4j	Quickserver	
Drjava	Ivatagroupware	Jgroups	Marauroa	Quilt	
Emma	Jag	Jhotdraw	Maven	Rssowl	

4.2. Apoio Computacional e Ferramenta para Automação da Abordagem

Três apoios computacionais foram utilizados no emprego da abordagem proposta e são brevemente mencionados. A ferramenta VerveineJ[1] e a plataforma Moose[2] foram utilizadas para obter as métricas e o sistema R [R Development Core Team 2011] foi utilizado executar as rotinas de clusterização.

A ferramenta VerveineJ permite extrair informações a partir do código fonte de sistemas Java e exportá-las para um arquivo com extensão `.mse`, cujo formato padrão é aceito pela plataforma Moose. Esse formato é semelhante ao formato de arquivos com extensão `.xml` e a principal diferença é a utilização de parênteses no arquivo `.mse` para identificar início e fim de um elemento ao invés de usar *tags*.

A plataforma Moose é independente de linguagem de programação, tendo sido projetada para realizar engenharia reversa e reengenharia de sistemas orientados a objetos. Essa plataforma funciona como um repositório para sistemas e permite visualização e análises sob diferentes perspectivas da estrutura, da qualidade, da interação entre componentes, dentre outras características de um sistema [Nierstrasz *et al*., 2005]. Particularmente, ela é capaz de gerar informações sobre diversas métricas de um sistema.

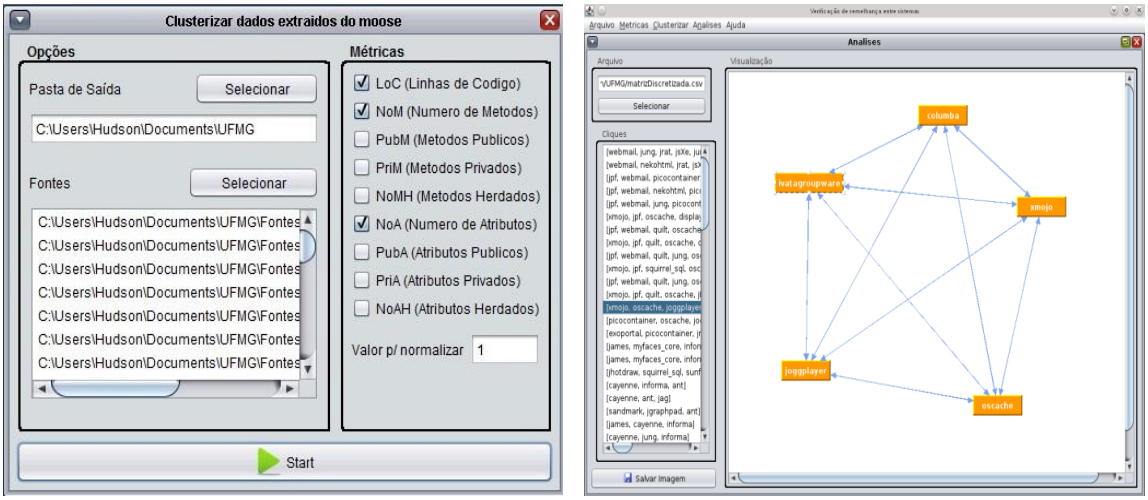
O R é uma plataforma especializada em cálculos estatísticos e geração de gráficos. Esse sistema possui um função que implementa o algoritmo *Kmeans*, cuja sintaxe é `kmeans(X, k)`, sendo `X` os dados a serem clusterizados e `k` a quantidade de grupos para particionamento dos dados.

Com o intuito de integrar os sistemas R e VerveineJ/Moose, foi desenvolvida uma ferramenta automatizada que prove ao usuário interfaces gráficas para melhorar a usabilidade da abordagem proposta. Por exemplo, para extração das métricas a partir do código fonte do sistema, a interface apresentada permite ao usuário (i) selecionar os arquivos fonte (`.java`) dos sistemas a serem analisados e (ii) definir um diretório de saída para armazenar os resultados (planilhas eletrônicas) gerados pelas ferramentas VerveineJ e Moose.

[1] https://gforge.inria.fr/projects/verveinej/
[2] http://www.moosetechnology.org/

Outro exemplo de interface provida pela ferramenta automatizada ao usuário é apresentado na Figura 3a. Essa interface permite ao usuário realizar a clusterização. Para isso, ele deve informar (i) o conjunto de sistemas a serem analisados (arquivos `.csv` resultantes da análise pelo Moose), (ii) as métricas a serem utilizadas (até o momento apenas métricas de tamanho) e (iii) o diretório de saída para armazenamento das planilhas com o resultado dos *clusters*.

O resultado da abordagem pode ser visto usando duas visões (Figura 3b). A interface apresentada nessa figura mostra duas áreas que apresentam duas visões das cliques resultantes. No lado esquerdo, as cliques encontradas são listadas em uma visão textual e, no lado direito, as cliques podem ser apresentadas por meio de representação de um grafo.



(a) Interface para Clusterização (b) Interface para Visualizar os Resultados (Cliques)

Figura 3 - Ferramenta para Automatizar a Abordagem Proposta

4.3. Definição das Métricas

Para a aplicação da abordagem proposta nos sistemas do *Qualitas Corpus*, foram utilizadas nove métricas de tamanho de classes:

- Número de Linhas de Código (LoC - *Line of Code*);
- Número de Atributos (NoA - *Number of Attributes*);
- Número de Atributos Públicos (NoPubA - *Number of Public Attributes*);
- Número de Atributos Privados (NoPriA - *Number of Private Attributes*);
- Número de Atributos Herdados (NoIA - *Number of Inherited Attributes*);
- Número de Métodos (NoM - *Number of Methods*);
- Número de Métodos Públicos (NoPubM - *Number of Public Methods*);
- Número de Métodos Privados (NoPriM - *Number of Private Methods*);
- Número de Métodos Herdados (NoIH - *Number of Inherited Methods*).

4.4. Passo #1: Determinar Valores de Referência para Perfis de Classes

Foram definidos cinco grupos para determinar o valor de referência para perfis de classes dos sistemas:

- G_1 - Classes com valores pequenos da métrica em questão;
- G_2 - Classes com valores da métrica em questão na faixa de pequeno a médio;
- G_3 - Classes com valores médios da métrica em questão;
- G_4 - Classes com valores da métrica em questão na faixa de médio a alto;
- G_5 - Classes com valores altos da métrica em questão.

O valor de referência dos cinco grupos de classes do sistema `FindBugs` (um dos sistemas do *Qualitas Corpus* analisados no trabalho) com relação à métrica LoC é apresentado na Tabela 2. O valor de referência é apresentado na segunda linha, a quantidade de classes é apresentada na terceira linha e o percentual das classes é apresentado na quarta linha. As informações presentes nessa tabela propiciam um melhor entendimento de como as classes do sistema `FindBugs` estão distribuídas com relação à quantidade de linhas de código. Por exemplo, cerca de 73,84% das classes podem ser consideradas pequenas (média de 26,2 linhas de código); por outro lado, 0,25% das classes podem ser consideradas grandes (média de 2331,7 linhas de código).

Tabela 2 - Valores de Referência para Perfis de Classes do Sistema `FindBugs`

	G_1	G_2	G_3	G_4	G_5
# LOC	26,2	135,2	359,3	787,5	2331,7
# Classes	878	226	63	19	3
% Classes	73,84	19,00	5,30	1,60	0,25

A Tabela 3 mostra valores de referências dos cinco grupos para outros dez sistemas presentes no *Qualitas Corpus*.

Tabela 3 - Valores de Referência de 10 Sistemas com Relação à Métrica LoC

Sistemas	G_1	G_2	G_3	G_4	G_5
Ant	19,64	92,18	217,49	448,91	916,92
Antlr	31,27	221,43	427,30	982,75	2627,50
Aoi	48,50	213,37	523,74	1130,36	2635,20
argouml	26,04	163,69	478,88	1813,00	9122,00
aspectj	56,38	382,73	1078,24	2719,89	6332,00
axion	27,87	100,02	238,50	787,71	2736,00
c_jdbc	26,17	121,08	300,00	830,62	1771,86
castor	25,69	145,70	385,61	899,85	2343,40
cayenne	17,11	82,16	197,59	400,77	973,33
cobertura	23,88	130,73	359,78	1806,00	7855,75

4.5. Passo #2: Obter Conjuntos de Sistemas Similares para uma Métrica

O algoritmo *Kmeans* deve ser executado para descobrir em quantos grupos (*clusters*) de sistemas similares os 86 sistemas (conjunto `S`) devem ser particionados. Para isso, a

função `kmeans(X, k)` do sistema R foi utilizada sendo `X` as informações dos cinco grupos dos 86 sistemas em análise e `k` a quantidade de *clusters* (a ser descoberto). Essa função foi executada diversas vezes para diferentes quantidades de *clusters* até obter uma coesão interna dos *clusters* superior a 90%, conforme sugerido na Seção 3.2. O valor alcançado foi k = 5 que apresentou coesão interna de aproximadamente 95%.

Isso significa que os 86 sistemas em análise podem ser classificados em cinco grupos, considerando indícios de similaridade com relação à métrica Número de Linhas de Código (LoC). Os sistemas classificados em cada um dos *clusters* são apresentados na Tabela 4, indicando que os sistemas de cada *cluster* possuem, em média, classes com a mesma quantidade de linhas de código. Conforme apresenta a Tabela 4, a maioria dos sistemas (33 sistemas) foi classificada no *cluster* 3. Esses sistemas apresentam em média a seguinte quantidade de LoC: $G_1 = 23,77$; $G_2 = 116,87$; $G_3 = 286,43$; $G_4 = 619,17$ e $G_5 = 1525,36$. Por outro lado, três sistemas foram classificados no *clusters* 2 e possuem em média a seguinte quantidade de LoC: $G_1 = 39,17$; $G_2 = 226,43$; $G_3 = 625,47$; $G_4 = 1887,83$ e $G_5 = 6147,33$.

Tabela 4 - Sistemas com Indícios de Similaridade com Relação à Métrica LoC

Número	*Cluster*$_1$	*Cluster*$_2$	*Cluster*$_3$	*Cluster*$_4$	*Cluster*$_5$
1	argouml	aspectj	c_jdbc	antlr	Ant
2	cobertura	drjava	drawswf	aoi	Cayenne
3	gt2-2	jparse	emma	axion	Columba
4	jrefactory		freecs	castor	Displaytag
5	pmd		galleon	findbugs	Exoportal
6	sablecc		ganttproject	freecol	Informa
7			heritrix	hibernate	Jag
8			htmlunit	ireport	James
9			itext	jasperreports	Jasml
10			ivatagroupware	jedit	Jext
11			javacc	jgroups	jFin_DateMath
12			jboss	jtopen	Jgrapht
13			jchempaint	quartz	Jmeter
14			jena	tomcat	Jmoney
15			jgraph	velocity	Jpf
16			jgraphpad		Jrat
17			jhotdraw		jsXe
18			joggplayer		jung
19			jspwiki		junit
20			maven		log4j
21			mvnforum		marauroa
22			myfaces_core		nakedobjects
23			poi		nekohtml
24			pooka		openjms
25			quickserver		oscache
26			rssowl		picocontainer
27			sandmark		proguard
28			squirrel_sql		quilt
29			struts		webmail
30			sunflow		
31			tapestry		
32			weka		
33			xmojo		

4.6. Passo #3: Obter Conjuntos de Sistemas Similares para um Conjunto de Métricas

O mesmo trabalho realizado para a métrica LoC deve ser feito para as demais métricas (NoA, NoPubA, NoPriA, NoIA, NoM, NoPubM, NoPriM e NoIH) definidas na Seção 4.3. Assim sendo, o mecanismo anterior foi empregado nove vezes, uma vez para cada métrica (conjunto M).

Uma matriz com dimensões 86 x 86 (86 sistemas em análise) foi construída. Essa matriz contém em cada célula a quantidade de vezes em que dois sistemas (S_i e S_j) foram classificados em um mesmo *cluster* ($Q_{m(i, j)}$). Um subconjunto dessa matriz é apresentado na Tabela 5, na qual indica-se a quantidade de métricas similares para cada par de sistemas.

Tabela 5 - Quantidade de Métricas entre Pares de Sistemas (Recorte da Matriz)

Sistemas	ant	antlr	aoi	argouml	aspectj	axion	c_jdbc	castor	cayenne
ant	9	4	0	3	4	2	2	4	4
antlr	4	9	0	3	5	3	2	6	4
aoi	0	0	9	0	0	2	4	0	0
argouml	3	3	0	9	3	2	3	5	3
aspectj	4	5	0	3	9	4	4	6	5
axion	2	3	2	2	4	9	6	3	3
c_jdbc	2	2	4	3	4	6	9	3	3
castor	4	6	0	5	6	3	3	9	7
cayenne	4	4	0	3	5	3	3	7	9

Por exemplo, a célula de interseção entre a linha com o sistema `axion` e a coluna com o sistema `ant` tem o valor 2, o que indica que esses sistemas estão no mesmo grupo em duas métricas. Outro exemplo, a célula de interseção entre a linha com o sistema `castor` e a coluna com o sistema `cayenne` tem o valor 7, o que indica que esses sistemas estão no mesmo grupo para sete métricas. Pode-se observar que as células da diagonal principal possuem o valor 9, pois o sistema é comparado com ele próprio.

A quantidade mínima de métricas desejadas entre dois sistemas (Q_{mmin}) foi definida com o valor 5 (metade da quantidade de métricas analisadas arredondada para cima). A representação computacional utilizada foi uma matriz de adjacência, sendo o valor 1 utilizado quando há presença de aresta entre dois sistemas e o valor 0 na ausência de aresta. A existência de uma aresta entre dois sistema, S_i e S_j, foi definida por $Q_{m(i, j)} >= Q_{mmin}$, ou seja, há aresta entre S_i e S_j se $Q_{m(i, j)} >= 5$.

Assim, a matriz de adjacência referente a Tabela 5 é apresentada na Tabela 6. Por exemplo, os sistemas `axion` e `ant` estão em grupos distintos e sozinhos (pois eles são similares em relação a apenas duas métricas). Por outro lado, os sistemas `castor` e `cayenne` estão no mesmo grupo, junto com os sistemas `ant` e `c_jdbc`, pois eles são similares em relação a pelo menos cinco métricas.

O grafo correspondente a essa matriz de adjacência é apresentado na Figura 4. Pode-se observar que há quatro cliques, sendo duas com três vértices (`antlr`, `aspectj` e `castor`; `castor`, `aspectj` e `cayenne`) e duas com dois vértices (`argouml` e `castor`; `axion` e `c_jdbc`).

Tabela 6 - Matriz de Adjacências (Recorte da Matriz)

Sistemas	ant	antlr	aoi	argouml	aspectj	axion	c_jdbc	castor	cayenne
ant	1	0	0	0	0	0	0	0	0
antlr	0	1	0	0	1	0	0	1	0
aoi	0	0	1	0	0	0	0	0	0
argouml	0	0	0	1	0	0	0	1	0
aspectj	0	1	0	0	1	0	0	1	1
axion	0	0	0	0	0	1	1	0	0
c_jdbc	0	0	0	0	0	1	1	0	0
castor	0	1	0	1	1	0	0	1	1
cayenne	0	0	0	0	1	0	0	1	1

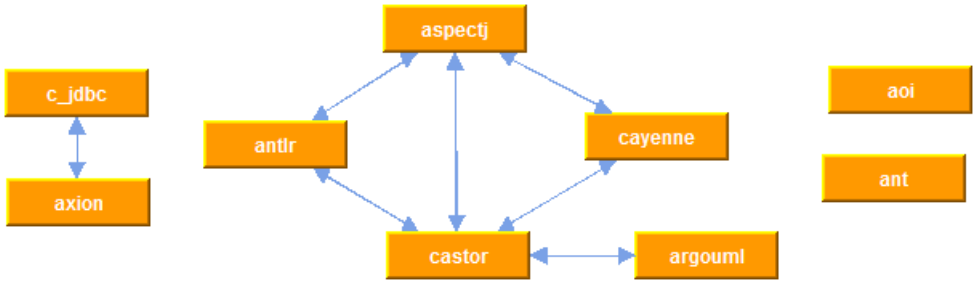


Figura 4 - Grafo Resultante da Matriz de Adjacência

A matriz de adjacência referente aos 86 sistemas em análise foi utilizada para encontrar cliques no grafo. Três cliques são apresentadas na Tabela 7. A clique máxima identificada possui 15 vértices (sistemas), caracterizando que esses sistemas possuem indícios de similaridade em pelo menos cinco métricas. Observa-se que a clique de tamanho 14 é bem semelhante à clique de tamanho 15, com a diferença que o sistema `drjava` participa da clique e os sistemas `jung` e `quilt`, que participam da clique de tamanho 15, não participam da clique de tamanho 14. Na Tabela 7, também apresenta-se uma clique de tamanho 10. Em comparação com a clique máxima, essa clique possui o sistema `squirrel_sql` e não possui os sistemas `findbugs`, `ivatagroupware`, `jFin_DateMath`, `jmoney`, `jung` e `junit`.

Tabela 7 - Resultado de 3 Cliques de Tamanho 15, 14 e 10.

Tamanho	Sistemas Presentes na Clique
15 Sistemas	`xmojo; findbugs; ivatagroupware; jFin_DateMath; jgrapht; jmoney; jpf; jung; junit; nekohtml; openjms; oscache; picocontainer; quilt; webmail`
14 Sistemas	`xmojo; drjava; findbugs; ivatagroupware; jFin_DateMath; jgrapht; jmoney; jpf; junit; nekohtml; openjms; oscache; picocontainer; webmail`
10 Sistemas	`xmojo; jgrapht; jpf; nekohtml; openjms; oscache; picocontainer; quilt; webmail, squirrel_sql`

Discussão dos Resultados: A qualidade de um produto de software pode ser medida tanto por fatores internos, como externos [Meyer, 2000]. A abordagem proposta no artigo destina-se a avaliar exclusivamente a qualidade interna de um software, tal como capturada por um conjunto de métricas. Particularmente, em nosso estudo de caso

escolhemos um conjunto de métricas de tamanho. Assim, os sistemas da Tabela 7 do ponto de vista dessa medida de qualidade interna (tamanho) possuem indícios fortes de similaridade. No entanto, três questões devem ser ressaltadas: (a) como esperado, a abordagem proposta não é capaz de explicar tal similaridade, a qual pode ser devida, por exemplo, a processos de desenvolvimento similares, padrões de programação similares, compartilhamento da mesma equipe de programadores etc ou mesmo ser mera coincidência; (b) apesar disso, por construção da abordagem descrita no trabalho, pode-se afirmar que os sistemas listados na tabela são bastante semelhantes do ponto de vista de tamanho; (c) por fim, nada impede que a abordagem seja usada com outras medidas de qualidade interna (por exemplo, destinadas a aferir acoplamento, coesão, complexidade, dentre outras). Acredita-se que quanto mais métricas forem usadas, maiores os indícios de similaridade apontados pela abordagem proposta.

6. Trabalhos Relacionados

Pesquisadores têm utilizado repositórios de projetos de software para servirem como *benchmark* em suas pesquisas. Em uma dessas pesquisas [Simon *et al*., 2006], foi utilizada a comparação entre projetos para verificar as melhores práticas na indústria de software, sendo focado especificamente em sistemas orientados a objetos.

Em outra pesquisa [Baggen *et al*., 2011], há afirmação da existência da relação entre a qualidade do código fonte e a manutenibilidade de um sistema. Neste sentido, foi proposta uma abordagem baseada na norma ISO/IEC 9126, mais especificamente na característica de qualidade manutenibilidade. Essa abordagem (i) analisou o código fonte de um sistema (extração de métricas), (ii) comparou com o resultado de outros sistemas e (iii) informou a classificação geral do sistema (corresponde a cinco níveis representados por estrelas) em termos de qualidade interna quando comparado aos demais. Para isso, o grupo *SIG.edu* mantém um repositório de *benchmark* que contém o resultado de centenas de avaliações padrões realizadas. O repositório é atualizado com o resultado de cada avaliação. Atualmente, o repositório deste grupo possui 45 tipos diferentes de linguagem de programação, tal como, Java, C, COBOL, C#, C++ e ABAP (maior contribuinte em termos de linhas de código). A validação do estudo em sistemas *open source* mostra que a classificação obtida pelo modelo de qualidade correlaciona positivamente com a velocidade na qual defeitos são resolvidos por mantenedores de sistema [Luijten; Visser, 2010].

O trabalho de Ferreira e outros (2011) identifica valores de referência para 6 métricas, a partir de 40 sistemas Java de código-fonte aberto. Para obter os valores de referência para cada uma das métricas é realizado um ajuste da melhor distribuição estatística nos dados coletados. Como resultado, os autores classificam o valor de referência em: Bom, Regular e Ruim.

Um estudo para verificar a estrutura de sistemas desenvolvidos em Java foi realizado por Baxter e outros (2006). Os pesquisadores trabalharam com 56 sistemas de código aberto desenvolvidos em Java e 17 métricas no escopo de classes, interfaces, métodos e pacotes. Os resultados desta análise mostra que algumas métricas estão em conformidade com lei de potência enquanto outras não.

7. Considerações Finais

Neste artigo, uma abordagem para verificar a similaridade entre sistemas orientados a objetos foi apresentada, bem como foi avaliada a sua aplicabilidade em um repositório de sistemas orientados a objetos clássicos/tradicionais (*Qualitas Corpus*). Essa abordagem possui três mecanismos para nortear a usa utilização: i) determinar valores de referência para perfis de classes; ii) obter conjuntos de sistemas similares para uma métrica e iii) obter conjuntos de sistemas similares para um conjunto de métricas.

A utilização dessa abordagem permitiu confirmar a existência de perfis de classes distintos em cada um dos sistemas analisados em relação ao seu tamanho (linhas de código), ratificando o resultado de alguns autores que caracterizam a conformidade da distribuição da quantidade de linhas de código das classes em uma Lei de Potência (por exemplo, [Taube-Schock *et al*., 2011]). Pode-se perceber que os sistemas, em geral, possuem muitas classes com pequena quantidade de linhas de código e poucas classes com grande quantidade de linhas de código o que caracteriza um valor de referência (ou valor médio) pouco representativo para as classes de um mesmo sistema. Assim, clusterizar classes em relação a quantidade de linhas de código é uma alternativa interessante, por favorecer o cálculo de médias aritméticas um pouco mais representativas, pois obtém-se um valor de referência para diferentes perfis de classes em um mesmo sistema.

A utilização do repositório *Qualitas Corpus* foi motivada por possui sistemas orientados a objetos funcionais e reais. Esses sistemas possuem tamanho considerável (milhares de linhas de código), o que proporcionou alcançar resultados mais robustos com a aplicação da abordagem. Além disso, propiciou a agregação de mais informações sobre esses sistemas. A utilização do conceito de clusterização para caracterizar perfis das classes em relação à quantidade de linhas de código permitiu descobrir valores de referência mais representativos.

Como sugestões de trabalhos futuros, tem-se a utilização da abordagem proposta em outros repositórios de sistemas orientados a objetos de código aberto e a utilização de outras métricas, tendo em vista que há diversas métricas para quantificar propriedades internas de sistemas, por exemplo, métricas de acoplamento, de coesão e de complexidade. Além disso, pretende-se identificar sistemas desenvolvidos por uma mesma empresa com relação às características internas das classes desses sistemas.

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

Referências

Baggen, R., Correia, J. P., Schill, K., & Visser, J. (2011). Standardized code quality benchmarking for improving software maintainability. Software Quality Journal, 1-21.

Barabási, A. L.; Albert, R. (1999) “Emergence of Scaling in Random Networks”. Science 286(5439), 509-520.

Baxter, G.; Frean, M.; Noble, J.; Rickerby, M.; Smith, H.; Visser, M.; Melton, H.; Tempero, E. (2006) “Undertanding the Shape of Java Software”. In: OOPSLA’06, 397-412.

Clauset, A.; Shalizi, C. R.; Newman, M. E. J. (2009) “Power-Law Distributions in Empirical Data”. SIAM Rev. 51(4), 661-703.

Concas, G.; Marchesi, M.; Pinna, S.; Serra, N. (2007) “Power-Laws in a Large Object-Oriented Software System”. IEEE Transaction Software Engineering, 33(10), 687-708.

Cormen, T.; Leiserson, C.; Rivest, R.; Cliford C. (2009) “Introduction to Algorithms”. 3rd edition, MIT Press.

Csardi G.; Nepusz T. (2006) “The IGraph Software Package for Complex Network Research”. InterJournal, Complex Systems 1695. Disponível em: <http://igraph.sf.net>.

Ferreira, K. A. M.; Bigonha, M. A. S.; Bigonha, R. S.; Mendes, L. F. O.; Almeida, H. C., (2011) “Identifying Thresholds for Object-Oriented Software Metrics”. The Journal of Systems and Software, 85(2), 244-257.

Louridas, P.; Spinellis, D.; Vlachos, V. (2008) “Power Laws in Software”. ACM Transactions on Software Engineering and Methodology, 18(1), 1-26.

Luijten, B.; Visser, J. (2010) “Faster Defect Resolution with Higher Technical Quality of Software”. In: 4th International Workshop on Software Quality and Maintainability (SQM 2010), Madrid, Spain, 10p.

Meyer, B. (2000) “Object-oriented Software Construction”, 2nd edition, Prentice-Hall.

Newman, M. E. J. (2005) “Power Laws, Pareto Distributions and Zipf’s Law”. Contemporary Physics, 46(5):323-351

R Development Core Team (2011) “R: A Language and Environment for Statistical Computing”, R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0. Disponível em: <http://www.R-project.org/>.

Shyam R. Chidamber and Chris F. Kemerer. (1994) “A metrics suite for object oriented design”. IEEE Transactions on Software Engineering, 20(6):476–493.

Simon, F.; Seng, O.; Mohaupt, T. (2006) “Code Quality Management: Technische Qualita¨t Industrieller Softwaresysteme Transparent und Vergleichbar Gemacht”. Heidelberg, Germany: Dpunkt-Verlag.

Tufféry, S. (2011) “Association Analysis, in Data Mining and Statistics for Decision Making”, John Wiley & Sons.

Taube-Schock, C.; Walker, R. J.; Witten I. H. (2011). “Can We Avoid High Coupling?”. 25th European Conference on Object-Oriented Programming (ECOOP), 204-228.

Tempero, E.; Anslow, C.; Dietrich, J.; Han, T.; Li, J.; Lumpe, M.; Melton, H.; Noble, J. (2010) “The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies”. Asia-Pacific Software Engineering Conference (APSEC), 336-345.