Uma Abordagem de Clusterização Arquitetural baseada em Similaridades Estrutural e Semântica

Carlos Pereira, Ricardo Terra

Departamento de Ciência da Computação
Universidade Federal de Lavras (UFLA)

cpereira@computacao.ufla.br, terra@dcc.ufla.br

Abstract. *Software architecture is defined how systems are structured into components and how such components should interact. Although the architectural design of a system is crucial, it is not usually documented. This indicates that architectural knowledge is in the minds of developers and source code. Thereupon, this article proposes an architectural clustering approach based on the structural and semantic similarities of the elements of the source code. Given a set of classes of a system, the idea is to group these classes into n components where their internal classes are structurally and semantically similar. Using the Power Iteration Cluster clustering algorithm, the PTMC structural similarity coefficient, and the cosine similarity on a semantic vector model, the proposed approach—initially calibrated in JHotDraw—could correctly group 47.37% and 56.76% of the classes in two other systems.*

Resumo. *Arquitetura de software pode ser definida como sistemas são estruturados em componentes e como tais componentes devem interagir. Embora o projeto arquitetural de um sistema seja de extrema importância, é comum não ser documentada. Isso indica que o conhecimento arquitetural está na mente dos desenvolvedores e no código fonte. Diante disso, este artigo propõe uma abordagem de clusterização arquitetural baseada em similaridades estrutural e semântica dos elementos do código fonte. Dado um conjunto de classes que compõem um sistema, a ideia é agrupar essas classes em n componentes onde suas classes internas sejam similares estrutural e semanticamente. Usando o algoritmo de clusterização Power Iteration Cluster, o coeficiente de similaridade estrutural PTMC e similaridade de cosseno sobre um modelo vetorial semântico, a abordagem proposta – inicialmente calibrada no JHotDraw – conseguiu agrupar corretamente 47,37% e 56,76% das classes de dois outros sistemas.*

1. Introdução

Arquitetura de software pode ser definida como sistemas são estruturados em componentes e como tais componentes devem interagir [12]. Isso corresponde as decisões de projeto que têm impacto em cada aspecto da construção e evolução de sistemas de software [30]. O ***problema*** é que o projeto arquitetural de um sistema de software (i) ou não é propriamente documentado ou (ii) é documentado mas não atualizado no decorrer do projeto. Independente de (i) ou (ii), o conhecimento arquitetural está na mente dos desenvolvedores e no código fonte.

Dado o conhecimento arquitetural na mente de desenvolvedores, existem duas soluções: se o desenvolvedor ainda fizer parte da equipe, pede-se que explicite esse conhecimento em documentos arquiteturais (solução i); caso contrário, contrata-se o desenvolvedor para esse mesmo fim (solução ii). No entanto, ambas as soluções podem não funcionar como esperado. Por exemplo, caso o desenvolvedor não mais se recorde dos aspectos arquiteturais do sistema ou caso o desenvolvedor se torne inacessível[1], as soluções (i) e (ii) falham.

Diante disso, este artigo propõe uma ***solução*** de clusterização arquitetural baseada em similaridade estrutural e semântica[2] dos elementos do código fonte para auxiliar o arquiteto na identificação dos componentes do sistema. Basicamente, dado um conjunto de classes que compõem um sistema, a ideia é agrupar essas classes em n componentes onde suas classes internas sejam similares estrutural e semanticamente. Por um lado, para calcular a similaridade estrutural, classes são vistas como o conjunto de tipos que a classe estabelece dependência, e.g., $Dep(ClienteDAO) = \{Cliente, Connection, Statement, \dots\}$. Por outro lado, para calcular a similaridade semântica, classes são vistas como um vetor de pesos de termos, e.g., $Termos(ClienteDAO) = \{[cliente, 17], [connection, 11], [statement, 6], \dots\}$.

Como resultado, o algoritmo de clusterização *Power Iteration Cluster* (PIC) sobre valores de similaridade estrutural mensurados pelo coeficiente PTMC e valores de similaridade semântico mensurados pelo cosseno sobre um modelo vetorial semântico, a abordagem proposta – inicialmente calibrada no JHotDraw – conseguiu agrupar corretamente 47,37% e 56,76% das classes de dois outros sistemas.

O restante do artigo está organizado como a seguir. A Seção 2 introduz os conceitos de similaridades estrutural e semântica e clusterização. A Seção 3 descreve a abordagem proposta e a Seção 4 avalia a abordagem proposta em dois sistemas de código aberto. Por fim, a Seção 5 relata trabalhos relacionados e a Seção 6 conclui.

2. Background

Esta seção apresenta conceitos fundamentais para a melhor compreensão deste trabalho. As Seções 2.1 e 2.2 descrevem respectivamente como são feitos os cálculos das similaridades estrutural e semântica entre classes. Por fim, a Seção 2.3 introduz o algoritmo de clusterização *Power Iteration Clustering*.

2.1. Similaridade Estrutural

Para agrupar as classes estruturalmente similares, cada classe é representada como o conjunto de tipos que ela estabelece dependência:[3]

$$Dep(C) = \text{Tipos com os quais a classe } C \text{ estabelece dependência} \quad (1)$$

Assumindo duas classes C_1 e C_2, os coeficientes para o cálculo de similaridade considera as seguintes variáveis [24, 10]:

[1] É irrelevante a discussão do motivo o qual pode ser desde desinteresse até mesmo falecimento.

[2] Neste artigo, o termo *semântica* refere-se a relações léxicas, o que difere da área de Compiladores.

[3] Por decisão de projeto, tipos comuns (e.g., primitivos e classes de propósito geral) são desconsiderados.

$$a = \text{quantidade de tipos que ambas as classes } C_1 \text{ e } C_2 \text{ estabelece dependência,} \quad (2)$$
$$b = \text{quantidade de tipos que apenas a classe } C_1 \text{ estabelece dependência,} \quad (3)$$
$$c = \text{quantidade de tipos que apenas a classe } C_2 \text{ estabelece dependência, e} \quad (4)$$
$$d = \text{quantidade de tipos do sistema que nem } C_1 \text{ nem } C_2 \text{ estabelece dependência.} \quad (5)$$

Para se obter o valor de similaridade estrutural com base nas variáveis acima, utilizou-se o coeficiente de similaridade denominado PTMC [22, 23]:

$$sim_{est}(Dep(C_1), Dep(C_2)) = \frac{2a^3 + 0.1\sqrt{d}}{1.71a^2 + 1.98b^2 + 1.78c^2 + 0.1d} \quad (6)$$

onde o valor 0 representa que as classes não possuem nenhum tipo em comum e não existe um valor máximo pré-determinado, i.e., quanto maior o número, maior a similaridade. Esse coeficiente é uma adaptação do coeficiente *Simple Matching*, resultado de um estudo empírico baseado em algoritmos genéricos. Em suma, esse coeficiente foi 5,23% a 6,81% estatisticamente melhor que 18 coeficientes do estado-da-arte em 101 sistemas de código aberto [22, 23].

As classes `Seriado` e `Filme` – ilustradas nos Códigos 1 e 2 – exemplificam o cálculo da similaridade estrutural. Primeiramente, desconsiderando os tipos comuns (`int`, `List` e `String`), são obtidas os tipos que cada classe estabelece dependência:

$$Dep(Seriado) = \{AudioVisual, Genero, ProdutorTelevisao, Temporada, Ator, Distribuidora\} \quad (7)$$
$$Dep(Filme) = \{AudioVisual, Genero, Diretor, Roteirista, Ator, Distribuidora\} \quad (8)$$

```
1  class Seriado implements AudioVisual {
2
3    String titulo;
4    int ano;
5    Genero genero;
6    ProdutorTelevisao criador;
7    List<Temporada> temporadas;
8    List<Ator> elenco;
9    Distribuidora distribuidora;
10
11   // adiciona uma nova temporada
12   adicionarTemporada(Temporada nova) {
13     elenco.addAll(nova.novosAtores());
14     temporadas.add(nova);
15   }
16 }
```

Código 1. Classe `Seriado`

```
1  class Filme implements AudioVisual {
2
3    String titulo;
4    int ano;
5    Genero genero;
6    Diretor diretor;
7    Roteirista roteirista;
8    List<Ator> elenco;
9    Distribuidora distribuidora;
10 }
```

Código 2. Classe `Filme`

Considerando o universo (U) de 15 tipos (i.e., o número de tipos que o sistema pode estabelecer dependência), as variáveis para cálculo de dependência ficam como a seguir:

$$a = 4 = |\{AudioVisual, Genero, Ator, Distribuidora\}| \quad (9)$$
$$b = 2 \quad |\{ProdutorTelevisao, Temporada\}| \quad (10)$$
$$c = 2 \quad |\{Diretor, Roteirista\}| \quad (11)$$
$$d = 7 \quad |U \setminus (Dep(Seriado) \cup Dep(Filme))| \quad (12)$$

Assim, a similaridade entre `Seriado` e `Filme` usando PTMC seria:

$$sim_{est}(Seriado, Filme) = \frac{2 * 4^3 + 0.1 * \sqrt{7}}{1.71 * 4^2 + 1.98 * 2^2 + 1.78 * 2^2 + 0.1 * 7} = 2.9760 \quad (13)$$

2.2. Similaridade Semântica

Para agrupar as classes semanticamente similares, cada classe é representada como um vetor, onde cada elemento desse vetor representa o peso de um termo de indexação. Os coeficientes para o cálculo de peso seguem o esquema TF-IDF (*Term Frequency-Inverse Document Frequency*) [3], conforme a Equação 14.

$$w_{ij} = \begin{cases} (1 + log\ f_{ij}) \times log\ \frac{N}{n_i} & \text{se } f_{ij} > 0 \\ 0 & \text{senão} \end{cases} \quad (14)$$

onde w_{ij} é o peso do termo k_i no documento d_j, f_{ij} é a frequência do termo k_i no documento d_j, N é a quantidade de documentos e n_i é a quantidade de documentos que contém o termo k_i. Para então calcular a similaridade semântica aplica-se a similaridade do cosseno, que é calculada através do cosseno do ângulo entre dois vetores [3] como descrita na Equação 15:

$$sim_{sem}(d_j, d_k) = cos(\Theta) = \frac{\vec{d_j} \cdot \vec{d_k}}{\|\vec{d_j}\| \times \|\vec{d_k}\|} = \frac{\sum_{i=1}^{t} w_{i,j} \times w_{i,k}}{\sqrt{\sum_{i=1}^{t} w_{i,j}^2} \times \sqrt{\sum_{i=1}^{t} w_{i,k}^2}} \quad (15)$$

Novamente, as classes `Seriado` e `Filme` – ilustradas nos Códigos 1 e 2 – exemplificam o cálculo da similaridade semântica. A Tabela 1 reporta as informações para o cálculo dos pesos TF-IDF dos termos das classes[4]. As colunas $f_{i,Seriado}$ e $f_{i,Filme}$ apresentam as frequências dos termos. A coluna n_i apresenta a quantidade de documentos em que cada termo aparece, o universo analisado possui 15 documentos, incluindo as classes sob análise. A coluna $IDF_i = log(N/n_i)$ apresenta o inverso da frequência dos termos nos documentos. Por fim, as colunas $w_{i,Seriado}$ e $w_{i,Filme}$ apresentam os pesos TF-IDF dos termos nas classes. Assim, aplicando os vetores encontrados com o esquema de peso TF-IDF na equação de similaridade do cosseno, têm-se que a similaridade semântica entre as classes `Seriado` e `Filme` é 0.7982.

[4]As palavras 'class', 'implements', 'int', 'ano', nova(os)', e 'all' não foram indexadas por serem palavras reservadas da linguagem Java ou *stop words*.

Tabela 1. Cálculo dos pesos TF-IDF dos termos nas classes $Seriado$ e `Filme`

termo	$f_{i,Seriado}$	$f_{i,Filme}$	n_i	$IDF_i = log(N/n_i)$	$w_{i,Seriado}$	$w_{i,Filme}$
seriado	1	0	6	1.3219	2.6438	0.0000
audio	1	1	13	0.2065	0.2065	0.2065
visual	1	1	4	1.9069	1.9069	1.9069
string	1	1	14	0.0995	0.0995	0.0995
titulo	1	1	7	1.0995	1.0995	1.0995
genero	2	2	10	0.5850	1.1700	1.1700
produtor	1	0	6	1.3219	1.3219	0.0000
televisao	1	0	8	0.9069	0.9069	0.0000
criador	1	0	5	1.5850	1.5850	0.0000
list	2	1	8	0.9069	1.8138	0.9069
temporada(s)	6	0	13	0.2065	0.7401	0.0000
ator(es)	2	1	3	2.3219	4.6439	2.3219
elenco	2	1	7	1.0995	2.1990	1.0995
distribuidora	2	2	3	2.3219	4.6438	4.6438
adiciona(r)	2	0	13	0.2065	0.4129	0.0000
add	2	0	13	0.2065	0.4129	0.0000
filme	0	1	11	0.4475	0.0000	0.4475
diretor	0	2	10	0.5850	0.0000	1.1700
roteirista	0	2	6	1.3219	0.0000	2.6438

2.3. *Power Iteration Clustering*

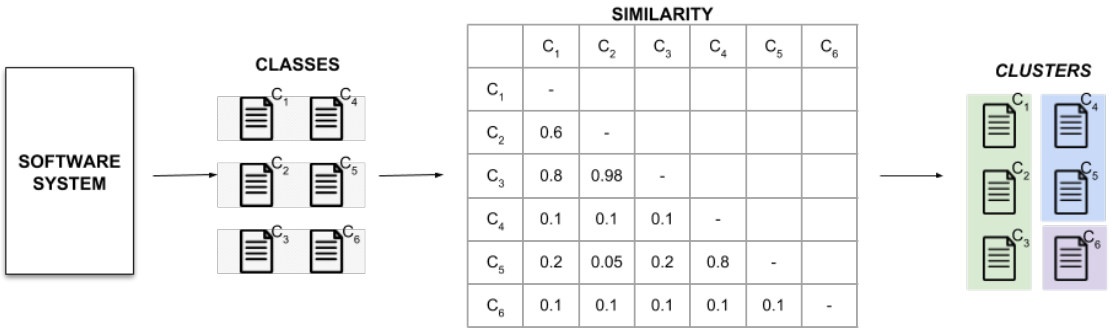
A clusterização é uma técnica de aprendizado de máquina não supervisionado que agrupa elementos que são semelhantes [29]. Particularmente neste artigo, é utilizada uma técnica de clusterização denominada *Power Iteration Clustering* (PIC) [15] que obteve resultados equiparáveis a técnica do estado-da-arte (k-means), porém com desempenho superior.

O PIC é uma técnica de clusterização semelhante as técnicas espectrais, pois ambos criam pontos em um espaço derivado de baixa dimensão a partir da matriz de similaridade dos dados, os quais são definidos os *clusters*. No entanto, na clusterização espectral, os pontos são definidos pelos menores autovetores da matriz Laplaciana de similaridade, já no PIC os pontos são definidos por uma aproximação dos autovalores com uma combinação linear de todos autovetores da matriz de similaridade normalizada. Essa característica que torna o PIC mais eficiente em relação aos métodos de clusterização espectrais, pois o cálculo dos autovetores são substituídos por uma pequena quantidade de multiplicações de matriz com vetor. O PIC mostrou ser um método simples e escalável com resultados melhores que os métodos de clusterização espectral [15].

Para computar os maiores autovetores da matriz de similaridade normalizada, o PIC utiliza a técnica *Power Iteration* (PI), um método iterativo onde começa com um vetor arbitrário e através de consecutivas multiplicações da matriz de similaridade com o vetor converge para os maiores autovetores da matriz. O PIC utiliza uma heurística de parada para o método PI, de forma a parar mais rápido. Após calcular os k maiores autovetores, o PIC determina os k *clusters* usando o algoritmo de clusterização k-means [15].

3. Abordagem Proposta

Este artigo tem como objetivo propor e avaliar uma abordagem de clusterização arquitetural baseada tanto em similaridade estrutural quanto semântica. Em outras palavras, dado um conjunto de classes que compõem um sistema, a ideia é agrupar essas classes em n componentes onde suas classes internas sejam similares estrutural e semanticamente, conforme ilustrado na Figura 1. O objetivo é apoiar arquitetos de software na tarefa de descoberta de componentes arquiteturais.



	C_1	C_2	C_3	C_4	C_5	C_6
C_1	-					
C_2	0.6	-				
C_3	0.8	0.98	-			
C_4	0.1	0.1	0.1	-		
C_5	0.2	0.05	0.2	0.8	-	
C_6	0.1	0.1	0.1	0.1	0.1	-

Figura 1. Abordagem Proposta

3.1. Metodologia

A similaridade estrutural – conforme descrita na Seção 2.1 – é medida pelo coeficiente PTMC em que cada classe é representada como o conjunto de tipos que ela estabelece dependência. Já a similaridade semântica – conforme descrita na Seção 2.2 – é medida pela similaridade de cosseno em que cada classe é representada como um vetor em que cada elemento representa o peso de um termo de indexação.

Pré-processamentos: A representação de um documento com base em algum tipo de informação geralmente requer algum pré-processamento, pois há informações que podem ser consideradas ruídos ou são comuns a todos documentos e não representam as especificidades de cada documento. Portanto, alguns pré-processamentos são realizados para atingir a representação estrutural e semântica dos documentos analisados.

Nas informações estruturais, são desconsideradas as dependências com tipos comuns, tais como tipos primitivos e classes de propósito gerais da linguagem Java (e.g., *wrappers*, coleções).

Nas informações semânticas, é utilizado todo o documento da classe, incluindo os comentários, e são realizados os seguintes pré-processamentos:

i. dividir literais *CamelCase*, e.g., de *"gerarNotaFiscal()"* para *"gerar Nota Fiscal"*;
ii. converter para caixa baixa, e.g., de *"gerar Nota Fiscal"* para *"gerar nota fiscal"*;
iii. remover marcações, e.g., de *"[...] Pereira.
 Iniciando com [...]"* para *"[...] Pereira. Iniciando com [...]"*;
iv. ignorar símbolos especiais, acentos e espaços adicionais, e.g., de *"if (get value() instanceof throwable) { [...] }"* para *"if get value instanceof throwable"*;

v. remover *stop words* (palavras que não tem relevância, são comuns em textos) e remover palavras reservadas da linguagem Java, e.g., de *"if get value instanceof throwable"* para *"value throwable"*; e
vi. converter as palavras em seus respectivos radicais (*stemmer*). *e.g.*, de *"result finished value throwable"* para *"result finish valu throwabl"*.

Para verificar a aplicabilidade de técnica de clusterização sobre informações estruturais e semânticas, utilizou-se o sistema JHotDraw 7.0.6[5] que é um sistema de código aberto com uma arquitetura estável e bem documentada. Antes dos experimentos e com base nos documentos arquiteturais do JHotDraw, foi criado um oráculo ($oracle$) que especifica os módulos do sistema e suas classes. A ideia é executar o algoritmo de clusterização sobre as similaridades estruturais e semânticas, analisar os resultados e propor uma abordagem híbrida de clusterização que considere tanto estrutura quanto semântica.

Embora tenham sido definidas como são calculadas as similaridades entre duas classes, é importante descrever como similaridades de um mais alto nível de abstração são medidas. A similaridade entre uma classe C e um *cluster* cl é a média aritmética simples da similaridade de C com cada classe de cl, conforme formalizado pela Equação 16.

$$sim(C, cl) = \frac{\sum_{c' \in cl} sim(C, c')}{|cl|} \tag{16}$$

E a similaridade de *cluster* cl é a média aritmética simples da similaridade entre suas classes, conforme formalizado pela Equação 17.

$$sim(cl) = \frac{\sum(\forall c', c'' \in cl(c' \neq c'') \rightarrow sim(c', c''))}{\binom{|cl|}{2}} \tag{17}$$

O oráculo define os componentes M_1, M_2, ..., M_n do sistema e quais as classes de cada módulo. Portanto, a conformidade de um resultado de uma clusterização r em relação ao oráculo é o somatório para cada módulo do número de classes comuns entre o resultado e o oráculo sobre o número total de classes[6], conforme formalizado pela Equação 18.[7]

$$conf(r, oracle) = \frac{\sum_{i=1}^{n} |classes(r, i) \cap classes(oracle, i)|}{|classes(oracle)|} \tag{18}$$

onde $classes(S)$ retorna as classes de um sistema S e $classes(S, i)$ retorna as classes do módulo i de um sistema S.

[5] http://www.jhotdraw.org/

[6] A conformidade arquitetural também pode ser interpretada como revocação.

[7] É importante observar que essa fórmula é executada para todos os possíveis mapeamentos entre os módulos do resultado r e do oráculo $oracle$.

3.2. Processo de Clusterização

Com base nos documentos arquiteturais, o oráculo ($oracle$) do JHotDraw 7.0.6 foi definido com nove módulos denominados *app*, *app-action*, *draw*, *draw-action*, *geom*, *gui*, *undo*, *util* e *xml* .

Executou-se o algoritmo de clusterização PIC usando tanto similaridade estrutural (r_{est}) quanto semântica (r_{sem}). A conformidade dos resultados obtidos com o oráculo foi como a seguir:

$$\begin{aligned} conf(r_{est}, oracle) = &\ 0.5725 \\ conf(r_{sem}, oracle) = &\ 0.3473 \end{aligned}$$

Com um viés qualitativo, a Tabela 2 reporta o número de classes de cada um dos nove módulos do JHotDraw que a abordagem estrutural e a semântica foram capazes de identificar corretamente. O objetivo é verificar se essas técnicas são complementares, isto é, se fossem combinadas os resultados seriam melhores. Conforme pode ser observado, quatro módulos foram melhores agrupados pelas informações estruturais enquanto que três foram melhores agrupados pelas informações semânticas. Dessa forma, propõe-se um algoritmo de clusterização híbrido que leve em consideração com igual peso – porém normalizados – os valores das similaridades estrutural e semântica, conforme formalizado pelas Equações 19 e 20.

$$sim_{proposta}(C, cl) = \frac{\sum_{c' \in cl} \frac{1}{2} sim_{est}(C, c') + \frac{1}{2} sim_{sem}(C, c')}{|cl|} \tag{19}$$

$$sim_{proposta}(cl) = \frac{\sum(\forall c', c'' \in cl(c' \neq c'') \rightarrow \frac{1}{2} sim_{est}(c', c'') + \frac{1}{2} sim_{sem}(c', c''))}{\binom{|cl|}{2}} \tag{20}$$

Tabela 2. Análise qualitativa JHotDraw

	Oráculo	**Acertos**	
Módulo	**No. de classes**	**Estrutural**	**Semântica**
app	12	3 (25%)	**9 (75%)**
app-action	36	**22 (61%)**	12 (33%)
draw	120	**90 (75%)**	44 (37%)
draw-action	33	**27 (82%)**	9 (27%)
geom	8	0 (0%)	0 (0%)
gui	6	0 (0%)	**3 (50%)**
undo	3	0 (0%)	0 (0%)
util	7	**2 (29%)**	1 (14%)
xml	37	6 (16%)	**13 (35%)**
Total	**262**	**150**	**91**

4. Avaliação

Esta seção tem como objetivo avaliar a abordagem de clusterização híbrida proposta em dois sistemas de código aberto.

4.1. Sistemas alvo

Foram utilizados dois sistemas de código aberto para avaliar o processo de clusterização. `myAppointments` é um sistema de gerenciamento de informação pessoal simples implementado para ilustrar técnicas de conformidade arquitetural. O sistema possui funcionalidades primárias aos usuários, como criar, recuperar, atualizar e excluir contatos pessoais. Apesar de seu tamanho e sua complexidade serem simplificados, seu conjunto de restrições de dependência são provavelmente utilizados em muitos casos de conformidade arquitetural de projetos reais [21]. `MyWebMarket` é um sistema web de comércio eletrônico previamente utilizado em estudos de remodularização [31]. O sistema gerencia clientes e produtos, realiza vendas, gera relatórios, etc. Mais importante, o sistema foi cuidadosamente projetado e implementado para assemelhar-se, em uma escala menor, a arquitetura de um sistema de gestão de recursos humanos com mais de 200 KLOC [30]. Esses sistemas foram utilizados neste artigo, pois possuem documentação de sua arquitetura, seus arquitetos estavam disponíveis e também já foram utilizados em estudos prévios de arquitetura de software.

4.2. Metodologia

A metodologia de avaliação é dividida nos seguintes três passos:

1. *Oráculo*: Com base nos documentos arquiteturais dos sistemas alvo, é especificado um oráculo ($oracle$) para cada sistema alvo.

2. *Clusterização*: É executado o algoritmo de clusterização híbrido proposto nos sistemas previamente descritos e medida a conformidade com o oráculo especificado em (1).

3. *Comparativo*: A abordagem híbrida é comparada com o algoritmo de clusterização utilizando somente as informações estruturais ou somente as informações semânticas.

4.3. Clusterização

Com base nos documentos arquiteturais, foram definidos os oráculos ($oracle_{myapp}$ e $oracle_{myweb}$) do `myAppointments` e do `MyWebMarket`. `myAppointments` foi definido com quatro módulos denominados *controller*, *view*, *model* e *util*. `MyWebMarket` foi definido com sete módulos denominados *action*, *persistence*, *scheduling*, *reporting*, *mailing*, *entity* e *util*.

Dessa forma executou-se o algoritmo de clusterização híbrido proposto nos dois sistemas. A conformidade dos resultados obtidos para o `myAppointments` e para o `MyWebMarket` com os respectivos oráculos foram 0.4737 e 0.5676.

No intuito de verificar se a abordagem híbrida traz melhores resultados, a Tabela 3 compara os resultados da abordagem híbrida proposta com o algoritmo de clusterização utilizando somente as informações estruturais ou somente as informações semânticas.

Tabela 3. Abordagem híbrida x estrutural x semântica

	Conformidade	
Abordagem	`myAppointments`	`MyWebMarket`
Híbrida	0.4737	**0.5676**
Estrutural	0.3684	0.5405
Semântica	**0.6316**	0.4865

O algoritmo de clusterização híbrido foi melhor que o estrutural e o semântico no `MyWebMarket` e foi o segundo melhor no `MyAppointments`, inferior apenas ao semântico. Uma análise qualitativa realizada pelo autor deste artigo indica que o `myAppointments` possui uma organização modular muito orientada às suas entidades de domínio, enquanto que no `MyWebMarket` isso não foi observado.

Dado os resultados, este artigo argumenta que a abordagem híbrida proposta traz resultados satisfatórios na generalidade, isto é, em sistemas cuja organização modular tenha sido realizada considerando tanto aspectos conceituais quanto de dependências entre elementos do código fonte. Diante dessa argumentação, a abordagem proposta foi padronizada estabelecendo igual peso aos valores das similaridades estruturais e semânticas, porém agora permitindo que o peso estabelecido possam variar caso o arquiteto tenha noções de como o sistema foi inicialmente organizado em módulos.

4.4. Limitações

Algumas limitações devem ser consideradas. Embora previsto como trabalho futuro, a abordagem não identifica a quantidade de módulos do sistema, portanto é necessário que uma pessoa com conhecimento da arquitetura do sistema informe a quantidade de módulos para a abordagem. A abordagem semântica considera que os identificadores nomeados pelos desenvolvedores sigam o padrão *CamelCase* e que sejam expressivos; se isso não ocorre no projeto, trará impactos negativos na acurácia da abordagem.

4.5. Ameaças a validade

Pelo menos duas ameaças a validade devem ser discutidas. Primeiramente, a criação dos oráculos pode não refletir a real organização modular dos sistemas. No entanto, os oráculos foram criados a partir de documentos arquiteturais disponibilizados pelos próprios arquitetos e foram conferidos com o arquiteto responsável pelos sistemas (validade de construção).

Em segundo lugar, como é comum em estudos empíricos em Engenharia de Software, os resultados não podem ser extrapolados (validade externa). Como a avaliação foi realizada em apenas dois sistemas de pequeno porte, é importante a avaliação em um número maior de sistemas, preferencialmente de médio e grande porte.

5. Trabalhos Relacionados

Esta seção apresenta trabalhos que propõem ou avaliam técnicas de descoberta arquitetural de software.

Santos et al. [26] propõem uma clusterização semântica para avaliar remodularizações em arquiteturas de sistema de software. Os autores avaliam seis remodularizações

reais de quatro sistemas de softwares usando métricas conceituais. A clusterização semântica utilizada é a proposta por Kuhn et al. [14], onde realiza a extração dos pesos e indexação dos termos, seguido pela clusterização via a técnica de clusterização hierárquica. Esse estudo conclui que as métricas conceituais descrevem uma melhoria na maioria dos casos em que a operação de remodularização *decomposição de módulo* é aplicada. Esse resultado indica que a criação de pacotes de alta coesão implica na dispersão dos conceitos por novos pacotes, revelando que desenvolvedores organizam as classes em pacotes de acordo com um conceito em comum.

Garcia et al. [11] propõem uma técnica de recuperação arquitetural baseada em informação textual e estrutural. A técnica consiste em extrair conceitos do código fonte por meio de um modelo estatístico utilizado em Recuperação de Informação chamado *Latent Dirichlet Allocation* (LDA) [5]. A partir das informações estruturais do código e por meio de aprendizado supervisionado de máquina, a clusterização se baseia na similaridade estrutural e na probabilidade dos conceitos. Esse estudo, além de recuperar a arquitetura do sistema de software, recupera os conceitos relacionados aos módulos da arquitetura, a classificação do conceito/módulo em independente ou específico da aplicação, e a classificação do módulo em componente ou *conector*.

Lutellier et al. [16] realizam uma avaliação de nove variações dentre seis técnicas de recuperação arquitetural. Um ponto ressaltado no trabalho é que o uso de dependências de símbolos (quando uma entidade do código fonte utiliza um símbolo definido em outra entidade) geram arquiteturas mais precisas que o uso de dependências de inclusão (quando uma entidade declara a inclusão de outra entidade). A avaliação desse trabalho é realizada em cinco projetos de código livre. Das seis técnicas de recuperação arquitetural avaliadas, ACDC [32], LIMBO [1], WCA [18] e Bunch [17]) utilizam dependências para determinar os *clusters*, enquanto que ARC [11] e ZBR [7] utilizam informação textual [16]. Esse estudo não objetiva determinar a melhor técnica de recuperação arquitetural, mas sim, provê apenas uma orientação. Os autores argumentam que o uso de ACDC, ARC, WCA e Bunch é indicado para grandes sistemas de software devido a escalabilidade. Já para um determinado nível de abstração, recomendam as técnicas WCA, LIMBO e ARC, pois permitem que o usuário defina a quantidade de *clusters* da arquitetura recuperada.

Avelino e Valente [2] apresentam uma técnica de descoberta arquitetural com base no grafo de dependências do código fonte, onde foram avaliadas duas técnicas de clusterização. Uma baseada no algoritmo de otimização Hill Climbing [25] e outra baseada no algoritmo de Girvan-Newman [13], que define os *clusters* com base na centralidade das arestas do grafo. Na métrica de avaliação proposta pelos autores, o algoritmo de Hill Climbing tem um desempenho melhor, mas os resultados apresentados pelo algoritmo de Girvan-Newman mostraram mais próximos da modularização adequada.

Silva et al. [27] apresentam um estudo empírico para entender a coesão através de análise conceitual, além de realizar uma comparação com a análise estrutural. O estudo de coesão pela análise conceitual proposto é avaliado em mais de três mil módulos de seis sistemas de médio e grande porte por métricas, onde a maioria depende de técnicas de mineração de texto como rede semântica, *Latent Semantic Indexing* (LSI) [9] e LDA. Os autores concluem que a coesão conceitual assimila uma dimensão diferente de coesão em relação ao aspecto estrutural, e que o aspecto conceitual compreende melhor a visão de coesão dos desenvolvedores.

6. Conclusão

Quando inexistente ou desatualizado, o projeto arquitetural de sistema de software pode ser obtido a partir da arquitetura implementada, i.e., código fonte. Este artigo propõe uma abordagem de clusterização arquitetural baseada em similaridades estrutural e semântica dos elementos do código fonte.

Adotou-se (i) os tipos que uma classe depende e o coeficiente PTMC para mensurar similaridade estrutural; (ii) o vetor de pesos de termos da classe e a similaridade de cosseno para mensurar similaridade semântica; e (iii) o algoritmo de clusterização *Power Iteration Cluster* para agrupar as classes baseados nos valores obtidos em (i) e (ii).

Com base em um oráculo com nove módulos cuidadosamente especificados no JHotDraw, PIC com dados estruturais atingiu uma conformidade com o oráculo de 57,25%, enquanto que com dados semânticos atingiu 34,73%. Como a análise qualitativa reportou que quatro módulos foram melhores agrupados pelas informações estruturais e três pelas informações semânticas, foi proposta uma abordagem de clusterização arquitetural híbrida que leve em consideração com igual peso – porém normalizados – os valores das similaridades estrutural e semântica dos elementos do código fonte.

A abordagem proposta foi avaliada em dois outros sistemas agrupando corretamente 47,37% e 56,76% das classes dos sistemas `myAppointments` e `MyWebMarket`, respectivamente. Porém, reivindica-se que a abordagem híbrida como proposta atua bem em sistemas cuja organização modular tenha sido realizada considerando tanto aspectos conceituais quanto de dependências entre elementos do código fonte. Caso contrário, o peso para as similaridades estrutural e semântica deve ser ajustado (não mais com igual peso).

Como trabalhos futuros, pretende-se: (i) avaliar em mais sistemas reais; (ii) desenvolver *plug-ins* para IDEs; (iii) avaliar em sistemas para dispositivos móveis; (iv) avaliar informações para a definição do interesse de uma classe em outra, e.g. dados históricos, mudanças temporais e carregamento de classes em tempo de execução; (v) avaliar técnicas para redução de dimensionalidade e clusterização, e.g., *Nonnegative matrix approximation* (NNMA) [28], método de Louvain [6] para detecção de comunidade em redes e técnicas de clusterização em redes complexas [8]; (vi) avaliar uso do *PageRank* [20] para determinar a relevância das classes nos módulos; (vii) avaliar métricas de equivalência, em vez da similaridade do cosseno, como o coeficiente de correlação de Pearson [4]; (viii) avaliar formas de representação vetorial reduzida pela utilização de técnicas *word embedding*, e.g., *Word2vec* [19]; e (ix) elaborar um algoritmo iterativo que descubra o melhor número de componentes, o que retira a obrigatoriedade de o usuário dizer em quantos componentes o sistema deve ser modularizado.

Referências

[1] Periklis Andritsos, Panayiotis Tsaparas, Renée J Miller, and Kenneth C Sevcik. Limbo: Scalable clustering of categorical data. In *9th International Conference on Extending Database Technology (EDBT)*, pages 123–146, 2004.

[2] Guilherme A Avelino and Marco Tulio Valente. ArchGraph: Modularização automática de sistemas usando clusterização de grafos de dependência. In *II Workshop de Visualização, Evolução e Manutenção de Software (VEM)*, 2014.

[3] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval: The Concepts and Technology Behind Search*. Addison-Wesley Publishing Company, USA, 2nd edition, 2011.

[4] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[5] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of Machine Learning Research*, 3(1):993–1022, 2003.

[6] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[7] Anna Corazza, Sergio Di Martino, Valerio Maggio, and Giuseppe Scanniello. Investigating the use of lexical information for software system clustering. In *15th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 35–44, 2011.

[8] Guilherme F de Arruda, Luciano da Fontoura Costa, and Francisco A Rodrigues. A complex networks approach for data clustering. *Physica A: Statistical Mechanics and its Applications*, 391(23):6174–6183, 2012.

[9] Susan T Dumais. Latent semantic analysis. *Annual review of information science and technology*, 38(1):188–230, 2004.

[10] Brian S. Everitt, Sabine Landau, Morven Leese, and Daniel Stahl. *Cluster Analysis*. Wiley, 5th edition, 2011.

[11] Joshua Garcia, Daniel Popescu, Chris Mattmann, Nenad Medvidovic, and Yuanfang Cai. Enhancing architectural recovery using concerns. In *26th International Conference on Automated Software Engineering (ASE)*, pages 552–555, 2011.

[12] David Garlan and Mary Shaw. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[13] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *National Academy of Sciences*, 99(12):7821–7826, 2002.

[14] Adrian Kuhn, Stéphane Ducasse, and Tudor Gírba. Semantic clustering: Identifying topics in source code. *Information and Software Technology*, 49(3):230–243, 2007.

[15] Frank Lin and William W. Cohen. Power iteration clustering. In *27th International Conference on Machine Learning (ICML)*, pages 655–662, 2010.

[16] Thibaud Lutellier, Devin Chollak, Joshua Garcia, Lin Tan, Derek Rayside, Nenad Medvidović, and Robert Kroeger. Comparing software architecture recovery techniques using accurate dependencies. In *37th International Conference on Software Engineering (ICSE)*, pages 69–78, 2015.

[17] Spiros Mancoridis, Brian S Mitchell, Yihfarn Chen, and Emden R Gansner. Bunch: A clustering tool for the recovery and maintenance of software system structures. In *7th International Conference on Software Maintenance (ICSM)*, pages 50–59, 1999.

[18] Onaiza Maqbool and Haroon Atique Babri. The weighted combined algorithm: A linkage algorithm for software clustering. In *8th European Conference on Software Maintenance and Reengineering (CSMR)*, pages 15–24, 2004.

[19] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[20] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[21] Leonardo Passos, Ricardo Terra, Renato Diniz, Marco Tulio Valente, and Nabor Mendonça. Static architecture-conformance checking: An illustrative overview. *IEEE Software*, 27(5):82–89, 2010.

[22] Arthur F. Pinto and Ricardo Terra. Better similarity coefficients to identify refactoring opportunities. In *XI Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS)*, pages 1–10, 2017.

[23] Arthur Ferreira Pinto. Empirically supported similarity coefficients for the identification of refactoring opportunities. Master's thesis, Universidade Federal de Lavras, Ricardo Terra (advisor), 2018.

[24] H. Charles Romesburg. *Cluster Analysis for Researchers*. Lulu Press, 2005.

[25] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson, 2016.

[26] Gustavo Santos, Marco Tulio Valente, and Nicolas Anquetil. Remodularization analysis using semantic clustering. In *2014 Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, pages 224–233, 2014.

[27] Bruno Silva, Cláudio Sant'Anna, and Christina Chavez. *Understanding Software Cohesion Metrics: Experimental Assessment of Conceptual Cohesion*. PhD thesis, Universidade Federal da Bahia, Universidade Estadual de Feira de Santana e Universidade Salvador, 2015.

[28] Suvrit Sra and Inderjit S Dhillon. Generalized nonnegative matrix approximations with bregman divergences. In *Advances in neural information processing systems*, pages 283–290, 2006.

[29] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to data mining*. Pearson, 2005.

[30] Ricardo Terra and Marco Tulio Valente. A dependency constraint language to manage object-oriented software architectures. *Software: Practice and Experience*, 39(12):1073–1094, 2009.

[31] Ricardo Terra, Marco Tulio Valente, Krzysztof Czarnecki, and Roberto S. Bigonha. Recommending refactorings to reverse software architecture erosion. In *16th European Conference on Software Maintenance and Reengineering (CSMR), Early Research Achievements Track*, pages 335–340, 2012.

[32] Vassilios Tzerpos and Richard C Holt. ACDC: an algorithm for comprehension-driven clustering. In *7th Working Conference on Reverse Engineering (WCRE)*, pages 258–267, 2000.