Um Mapeamento Sistemático da Literatura sobre Ferramentas de Refatoração de Software
Alternative Title: A Systematic Mapping of Literature on Software Refactoring Tools

Cleiton Silva Tavares
Universidade Federal de Minas Gerais (UFMG)
Software Engineering Laboratory (LabSoft)
Belo Horizonte, MG
cleiton.silva@dcc.ufmg.br

Fischer Ferreira
Universidade Federal de Minas Gerais (UFMG)
Software Engineering Laboratory (LabSoft)
Belo Horizonte, MG
fischerjf@dcc.ufmg.br

Eduardo Figueiredo
Universidade Federal de Minas Gerais (UFMG)
Software Engineering Laboratory (LabSoft)
Belo Horizonte, MG
figueiredo@dcc.ufmg.br

ABSTRACT

Refactoring consists of improving the internal structure of the code without changing the external behavior of a software system. However, the task of refactoring is very costly in the development of an information system. Thus, many tools have been proposed to support refactoring the source code. In order to find tools cited in the literature, this work presents a Systematic Literature Mapping about refactoring. As a result, this paper summarizes the refactoring tools that have been published in the last 5 years in terms of the tool profiles developed, which programming languages have support for refactoring and which are the main refactoring strategies that are handled by tools. It has been identified that publications on refactoring have remained constant over the past 5 years. Also, most of the refactoring works describe tools, being they for systems written in the Java language, that perform code refactoring automatically and the main refactorings are: Move Method, Pull Up Method, Extract Class and Code Clone. Finally, we performed an analysis of the data returned by the DBLP library. As a result, it was observed that the papers returned by the DBLP have a high level of similarity with the other research bases studied.

ACM Reference Format:
Cleiton Silva Tavares, Fischer Ferreira, and Eduardo Figueiredo. 2018. Um Mapeamento Sistemático da Literatura sobre Ferramentas de Refatoração de Software Alternative Title: A Systematic Mapping of Literature on Software Refactoring Tools. In *SBSI'18: XIV Brazilian Symposium on Information Systems, June 4–8, 2018, Caxias do Sul, Brazil.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3229345.3229357

1 INTRODUÇÃO

Refatorar consiste em modificar um código fonte com objetivo de criar uma estrutura que seja melhor organizada, sem alterar a saída que o código fonte originalmente apresentava mediante a uma dada entrada [7]. Ou seja, por meio da refatoração é possível tornar o código fonte de um sistema de informação mais intuitivo, claro e com um design mais elegante. O objetivo da refatoração é apoiar a fase da manutenção evolutiva e corretiva de sistemas de informação. Os benefícios dessa prática refletem diretamente na redução de custos pois, facilita o entendimento dos desenvolvedores sobre o código fonte de um sistema.

A prática manual da refatoração pode ser uma tarefa exaustiva e representar um esforço supérfluo em meio a crescente demanda para novas funcionalidades que os sistemas de informação são submetidos. Entretanto, com um apoio ferramental adequado é possível galgar os benefícios que a refatoração propicia sem ter um acréscimo expressivo de custo no ciclo de vida de um sistema de informação [22] [51].

O objetivo deste trabalho é verificar quais são as ferramentas recentemente propostas na literatura sobre refatoração e sumarizar algumas características do estado da arte sobre o assunto. Para realização desse objetivo foi realizado um Mapeamento da Literatura em busca de abordagens sobre refatoração, publicados em conferências de impacto entre os anos de 2013 e 2017.

Ainda, neste Mapeamento da Literatura optou-se em utilizar uma base de dados inovadora que permite facilitar o processo de busca de trabalhos por conferências de interesse. Essa base intitulada DBLP *Computer Science Bibliography*[1] fornece acesso às principais publicações científicas na área da Ciência da Computação.

Como resultado foi identificado que novas ferramentas para apoiar refatoração estão sendo desenvolvidas ao longo desses cinco anos. Ainda, que as publicações sobre o assunto se mantém em uma taxa constante, indicando que existe a preocupação da comunidade em apresentar novas formas de apoiar a refatoração por meio de ferramentas. Identificou-se ainda que a maioria dos trabalhos sobre refatoração descrevem ferramentas, sendo elas para sistemas escritos na linguagem Java, que realizam refatoração no código de forma automática. As principais estratégias de refatoração suportadas são *Move Method*, *Pull Up Method*, *Extract Class* e *Code Clone*.

Como resultado sobre a utilização da biblioteca DBLP no Mapeamento da Literatura foi observado um alto grau de similaridade

SBSI'18, June 4–8, 2018, Caxias do Sul, Brazil

ACM ISBN 978-1-4503-6559-8/18/06…$15.00
https://doi.org/10.1145/3229345.3229357

[1] http://dblp.uni-trier.de/

entre os resultados obtidos pela DBLP em comparação com as outras bases de dados mais tradicionais. Ainda, foram listados alguns detalhes observados com o uso da biblioteca neste trabalho.

O restante do trabalho está organizado da seguinte forma: A Seção 2 apresenta a definição de Refatoração. A Seção 3 descreve o protocolo utilizado para realização do Mapeamento da Literatura. A Seção 4 apresenta os resultados encontrados da condução do Mapeamento da Literatura. A Seção 5 apresenta uma discussão sobre os resultados e as lições aprendidas com o estudo. Ainda, na Seção 6 são apresentadas as ameaças a validade do estudo e suas limitações. Na Seção 7 são descritos os trabalhos relacionados. Finalmente, na Seção 8 são apresentados a conclusão do estudo e possibilidades para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Refatorar é o processo de alteração de um sistema de software para melhorar sua estrutura interna de tal forma que não altere o comportamento externo do código. É uma maneira disciplinada de limpar o código, melhorando a estrutura do código que foi escrito, minimizando assim as chances de introduzir erros[7].

Nos Algoritimos 1 e 2 pode ser observado o uso de uma estratégia de refatoração, o *Extract Method*, utilizado para extrair um pedaço de código reutilizável de um único lugar e chamar esse código extraído por meio de um método em vários outros lugares dos quais o código anteriormente estava duplicado. Caso seja preciso usar o mesmo código em um novo método, pode ser utilizada a refatoração *Extract Method*, para fornecer o reúso correto de código. Assim, o código escrito em apenas um lugar servirá para os fragmentos de código que requisitarem, sem que seja preciso duplicar o mesmo fragmento de código escrevendo-o em outras partes do código.

Algoritmo 1 Código fonte em Java

```
1  void printOwing (double amount){
2    print Banner();
3    //print details
4      System.out.println("name: " + _name);
5      System.out.println("amout: " + amount);
6  }
```

Algoritmo 2 Código fonte em Java Refatorado

```
1  void printOwing (double amount){
2    print Banner();
3      printDetails(amount);
4  }
5  void printDetails(double amount){
6      System.out.println("name: " + _name);
7      System.out.println("amout: " + amount);
8  }
```

Desta forma, evita-se redundâncias de código. A redundância torna o código extremamente complicado para manter. Pois, algumas mudanças no código duplicado acarretam em modificações de várias partes do código. Além de ser custoso encontrar esses fragmentos espalhados por todo o código, pode ocorrer de alguma parte ficar sem ser alterada, ocasionando uma inconsistência e eventuais falhas no sistema de informação [43].

3 CONFIGURAÇÃO DO ESTUDO

Nesta seção são descritas questões de pesquisa que direcionam esse trabalho. São demonstradas as bases de dados utilizadas no estudo além dos critérios de inclusão e exclusão dos artigos encontrados.

3.1 Questão de Pesquisa

Qual é o perfil das ferramentas de refatoração propostas recentemente?

Essa questão de pesquisa visa levantar um questionamento sobre qual é o estado da arte em ferramentas que apoiam a refatoração publicadas nos últimos cinco anos. Com esse objetivo, investiga-se algumas das principais características das ferramentas propostas recentemente na literatura. Assim, serão sumarizadas características tais como: nome, artigo referência, linguagens apoiadas, disponibilidade para uso, característica de funcionamento, apoio de documentação, exemplos de uso e estratégias de refatoração apoiadas.

A motivação para isso é apoiar os desenvolvedores de sistemas de informação interessados em escolher uma ferramenta de refatoração que melhor se ajuste às suas necessidades. Portanto, esse estudo contribui com um catálogo de ferramentas de refatoração recentemente publicadas na literatura.

3.2 Bases de Dados e String de Busca

Sete bases de dados foram utilizadas para extração dos estudos primários que estão disponíveis nas bibliotecas digitais citadas na literatura, sendo elas: *ACM Digital Library, DBLP Computer Science Bibliography, IEEExplore, Springer Link, Scopus, Engineering Village* [2]. Essas bases foram escolhidas por indexar periódicos de eventos de impacto na área. Por meio da *String* de busca: "refactor*", foi conduzida uma pesquisa nas bases selecionadas segundo a peculiaridade de cada uma. Na busca, foi incluído apenas artigos com o título que contivesse o fragmento "refactor" e suas variantes e publicados entre os anos de 2013 e 2017.

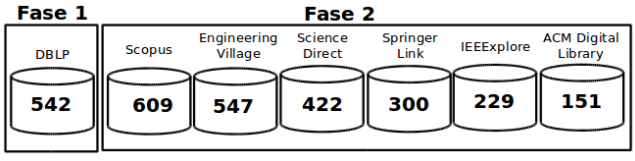

Figura 1: Distribuição de artigos por base de dados

A presente pesquisa foi realizada em Janeiro de 2018, desta forma foi obtido o total de 2800 artigos, conforme descrito na Figura 1. Como pode ser observado nesta figura, existem trabalhos selecionados em duas fases. Na primeira fase, foram pesquisados trabalhos

[2]http://dl.acm.org/ | http://dblp.uni-trier.de/ | http://ieeexplore.ieee.org/ http://link.springer.com/ | http://www.sciencedirect.com/ | http://scopus.com/ https://www.engineeringvillage.com

oriundos da base de dados DBLP. Na segunda fase, foram pesquisados trabalhos composto pelas bases de dados descritas na Figura 1. Desta forma, foram encontrados para a Fase 1 o total de 542 artigos e para a Fase 2 foram obtidos um total de 2258 artigos, dispostos da seguinte forma: *Scopus* com 609, *Engineering Village* com 547, *SciencDirect* com 422, *Springer Link* com 300 artigos, *IEEExplore* com 229 e *ACM Digital Library* com 151 artigos.

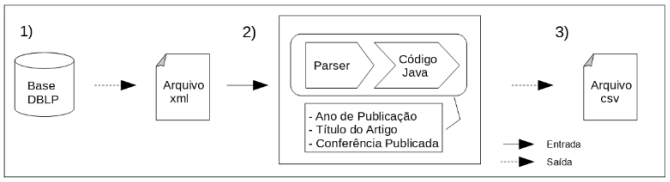

Figura 2: Dados DBLP

A Figura 2 apresenta os três passos principais descritos a seguir para utilização da Biblioteca DBLP. A motivação para o uso da Biblioteca DBLP se dá por ela ser usada em muitos países como uma versão simplificada de currículo de pesquisadores em computação. Desta forma, espera-se que ela possa indexar artigos relevantes da área.

- ***Passo1***: Os dados foram adquiridos através de um arquivo xml [3] contendo todas as referências disponíveis pela base de dados. A filtragem dos artigos realizada no próximo passo é feita tendo o arquivo xml como entrada.
- ***Passo2***: Foi desenvolvido uma ferramenta em Java, utilizando o exemplo de *parser* disponível pelo DBLP [4]. Esta ferramenta obtém informações utilizando alguns critérios para seleção dos artigos desejados, sendo eles:
  (1) *Ano de Publicação* - definido sendo os cinco últimos anos (2013, 2014, 2015, 2016 e 2017) para assim apresentar resultados atuais;
  (2) *String de Busca* - definida pela *String* "refactor" que foi buscada através do título de todos os trabalhos publicados;
- ***Passo3***: A saída da ferramenta desenvolvida foi um arquivo csv contendo todos os artigos encontrados que atenderam todos os critérios definidos.

3.3 Critérios de Inclusão e Exclusão

Após escolhidas as bases de dados a serem utilizadas, foram definidos os critérios de inclusão e exclusão dos trabalhos. Como descrito pela Figura 3 na Redução 1, para as Fases 1 e 2, foi realizada a remoção dos artigos duplicados entre as bases de dados e ainda selecionados os trabalhos julgados importantes a partir do nome do artigo. Ainda, a Figura 3 demonstra a Redução 2, nessa etapa, foram selecionados apenas os trabalhos que contém as informações necessárias para montar o perfil das ferramentas discutidas na questão de pesquisa.

Dessa forma, foi realizada a leitura do resumo dos artigos e selecionados aqueles que continham as informações de interesse. Caso as informações não estivessem dispostas no resumo, foi analisado o artigo para realizar a extração das informações. Por meio dessa

[3]http://dblp.uni-trier.de/xml/
[4]http://dblp.uni-trier.de/faq/How+to+parse+dblp+xml

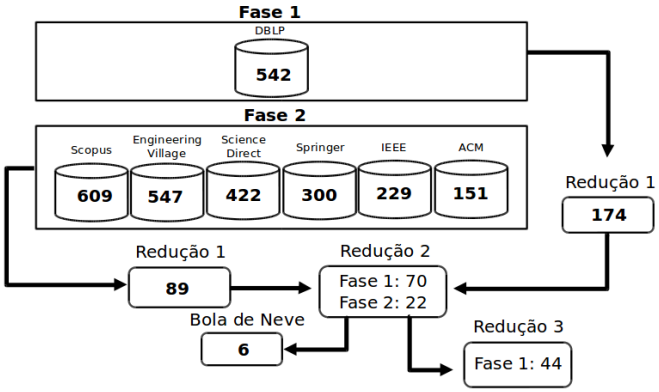

Figura 3: Seleção dos Artigos

etapa, foi alcançado um total de 70 artigos para a Fase 1 e 22 artigos para a Fase 2. Somando as duas fases foram obtidos o total de 92 artigos.

Por fim, utilizou-se a técnica de Bola de Neve *(Snowballing)*[49] para incluir trabalhos que os artigos selecionados referenciam. Por meio dessa técnica foram obtidos mais 06 artigos, ficando o total de 98 trabalhos. Finalmentem na Redução 3, foram selecionados apenas os artigos que apresentavam novas ferramentas de refatoração. Nesta redução, foi encontrado um total de 44 novas ferramentas que são descritas nas demais etapas deste trabalho. A lista dos artigos encontra-se no website do projeto[5]

4 RESULTADOS ALCANÇADOS

Nesta seção são apresentados os resultados alcançados por meio do Mapeamento da Literatura bem como algumas discussões em respostas à questão de pesquisa apontada na Seção 3.1. Esta seção está organizada como se segue. A Seção 4.1 descreve o panorama das publicações sobre refatoração. Na Seção 4.2 é demonstrado o perfil das ferramentas de refatoração.

4.1 Panorama dos Estudos Selecionados

Esta seção demonstra a caracterização dos artigos selecionados neste trabalho. São demonstradas a distribuição dos artigos por ano e as conferências onde foram publicados.

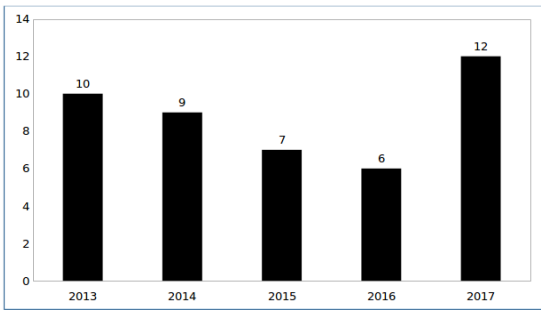

Figura 4: Distribuição de artigos por ano

[5]http://labsoft.dcc.ufmg.br/doku.php?id=about:systematic_mapping

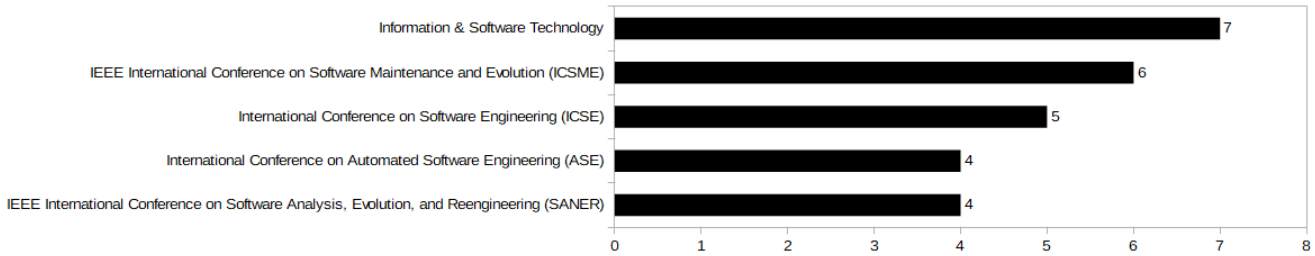

Figura 5: Distribuição dos artigos por conferências

A Figura 4 demonstra a distribuição de artigos ao longo dos anos. Foram considerados os anos entre 2013 e 2017 para delimitar esse trabalho. Como pode ser observado, os estudos sobre refatoração continuam sendo publicados, em destaque para o ano de 2017 que teve um ligeiro aumento em relação aos anos anteriores. Isso demonstra a preocupação da comunidade sobre o assunto.

Outro resultado sobre a visão geral dos artigos publicados sobre as ferramentas de refatoração são as conferências em destaque com o maior número de publicações sobre o assunto. A Figura 5 demonstra as cinco conferências e periódicos com o maior número de publicações sobre refatoração encontrados na literatura. Como por ser observado nesta figura, veículos de grande relevância da literatura possuem trabalhos sobre refatoração. Os veículos em destaque são *Information & Software Technology* e *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. Elas possuem 7 e 6 artigos publicados respectivamente sobre ferramentas de refatoração nos últimos cinco anos.

4.2 Perfil das Ferramentas de Refatoração

Ao realizar o levantamento sobre as abordagens encontradas na literatura foram identificados 51 trabalhos que descrevem estudos feitos sobre refatoração, sem fornecer ou deixar evidente a apresentação de uma nova ferramenta de refatoração. Porém, 44 artigos encontrados indicam abordagens que remetem a uma nova ferramenta de refatoração. Como pode ser observado, muitas abordagens foram proposta na literatura e, por essa razão, esse trabalho procura sumarizar as principais características das novas ferramentas de refatoração identificadas.

Segundo os artigos primários analisados no Mapeamento da Literatura, foi sumarizado mediante a leitura dos trabalhos as características das ferramentas de refatoração encontradas. Os itens analisados foram: **Plugin:** Nesta coluna foi demonstrado se a ferramenta é apresentada como um *plugin*. **Protótipo:** Nesta coluna foi demonstrado se a ferramenta é apresentada como um protótipo. **Característica:** Nesta coluna foi demonstrado quais as funcionalidades das ferramentas, tais como sugestão de refatoração, realiza refatoração, detecta refatoração, detecta erro na refatoração, estima impacto da refatoração e prediz a necessidade de realizar refatoração. **Operação:** Nesta coluna foi demonstrado como a ferramenta realiza a operação: de forma automatizada, semi-automatizada, pseudo-automatizada ou passo a passo. **Documentação:** Nesta coluna foi indicado o link para acesso a documentação da ferramenta.

A Tabela 1 demonstra as características das ferramentas de refatoração analisadas. Vale ressaltar que essas informações foram apresentadas de acordo com o que foi disponibilizado pelos artigos e, quando não foi encontrado a informação buscada no artigo, não foi preenchido nenhuma informação na tabela.

Ainda, foi feito uma sumarização das estratégias de refatoração que as ferramentas dão suporte. A Tabela 2 descreve o nome das ferramentas analisadas bem como as estratégias de refatoração que ela suporta.

Foram definidos 5 grupos com as estratégias de refatorações em que as ferramentas oferecem maior suporte, sendo que em cada um dos grupos foram utilizadas as siglas para identificar cada uma delas. Vale ressaltar que o preenchimento dessa tabela segue o mesmo padrão de preenchimento da tabela anterior. As informações contidas nestas tabelas foram apresentadas de acordo com o que foi disponibilizado pelos artigos e, quando não foi encontrada a informação buscada no artigo, não foi preenchido nenhuma informação na tabela.

5 DISCUSSÕES SOBRE OS RESULTADOS ALCANÇADOS

Nesta seção são relatadas algumas discussões em cima dos resultados alcançados neste trabalho. A organização dessa seção é a seguinte. A Seção 5.1 demonstra as principais discussões sobre as ferramentas de refatoração recentemente publicadas e é respondida a questão de pesquisa do presente trabalho. Por fim, a Seção 5.2 demonstra as discussões com relação a biblioteca DBLP.

5.1 Uma análise sobre as ferramentas de refatoração

Como pode ser observado na Tabela 1, a linguagem Java dispontou como a linguagem mais apoiada pelas ferramentas de refatoração. Um total de 37 ferramentas apoiam Java de 44 ferramentas encontradas. Ainda, destaca-se que dentre as 37, duas ferramentas dão suporte as linguagens Java e C. Foi encontrada apenas uma ferramenta exclusiva para a linguagem C.

A Tabela 1 demonstra o total de 44 ferramentas analisadas. Algumas características das ferramentas podem ser destacadas: 27 ferramentas não são *plugins*, 40 ferramentas não são protótipos, 23 ferramentas realizam refatoração, 20 ferramentas realizam a refatoração de forma automática e 10 ferramentas fornecem sugestão para realização de refatoração.

[5]Migrate Skeletal Implementation to Interface

Tabela 1: Perfil das ferramentas analisadas

Programa	Ling.	Plug.	Prot.	Caract.	Oper.	Link
LambdaFicator[12]	Java	Sim		SUG/REFAC		http://refactoring.info/tools/LambdaFicator/
Cider[41]	Java			DETEC		
HAN[13]	Java			DETEC		
PILGRIM[47]	Java			REFAC		
Obey[15]	Java	Sim		REFAC		
WebDyn[38]	PHP			REFAC	AUT	
KRISHNAN[23]	Java			REFAC		
JSRefactor[5]	JavaScript	Sim			SEMI	http://www.brics.dk/jsrefactor/plugin.html
GLIGORIC[11]	Java e C	Sim		ERR		
Xll[30] [31]	Java			REFAC	AUT	http://xll.pst.ifi.lmu.de/
MONDAL[36]	Java e C			DETEC	AUT	
BAVOTA[2]	Java			REFAC	AUT	
Css-analyser[32]	CSS	Sim		SUG/REFAC	AUT	http://dmazinanian.me/conference-papers/fse/2014/06/16/fse14.html
Morex[9]	Java			SUG		
Ripe[3]	Java			IMPAC		https://seers.utdallas.edu/projects/ripe/
JExtract[42]	Java			SUG		http://aserg-ufmg.github.io/jextract/
WANG[48]	Java			SUG	AUT	
TAO[45]	Java			REFAC	AUT	
HAN 2[14]	Java			SUG	AUT	
GAITANI[8]	Java	Sim	Sim	REFAC	AUT	
Rename Expander[28]	Java	Sim	Sim	SUG		~liuhui04/tools/rename/
Morpheus[24]	C			REFAC		http://fosd.net/morpheus/
Asynchronizer[27] [25]	Android	Sim		REFAC	AUT	http://refactoring.info/tools/asynchronizer/
Asyncdroid[25] [26]	Android	Sim		REFAC	AUT	http://refactoring.info/tools/asyncdroid/
Reflective Refactoring[20] [21]	Java	Sim		REFAC	AUT/SEMI	
YANG[53]	Java			REFAC		
Ad-Room[19]	Ecore	Sim		DETEC	AUT	https://youtu.be/4OJ8zHtfnq8
B-Refactoring[52]	Java			REFAC	AUT	https://github.com/Spirals-Team/banana-refactoring
MORALES[37]	Java			REFAC		
MAZINANIAN[33]	Java	Sim		DETEC		https://www.youtube.com/watch?v=K_xAEqIEJ-4&feature=youtu.be
OUNI[39]	Java			SUG	AUT	
Null Terminator[34]	Java	Sim	Sim		PSE	https://github.com/StefanMedeleanu/NullTerminator
KEBIR[16]	Java			REFAC	AUT	
MKAOUER[35]	Java			REFAC	AUT	
KHATCHADOURIAN[17]	Java	Sim		REFAC	AUT	https://github.com/ponder-lab/Migrate-Skeletal-Implementation-to-Interface-Refactoring https://openlab.citytech.cuny.edu/interfacerefactoring/
ZAFEIRIS[54]	Java	Sim		REFAC	AUT	
C-JrefRec[46]	Java			SUG		
MSITI[5][18]	Java	Sim	Sim			https://github.com/ponder-lab/Migrate-Skeletal-Implementation-to-Interface-Refactoring
More[40]	Java			SUG	AUT	
MANSOOR[29]	Java			REFAC		
DALLAL[4]	Java			PRE		http://www.isc.ku.edu.kw/drjehad/PredictingMMR.htm
RefDiff[44]	Java			DETEC		https://github.com/aserg-ufmg/RefDiff
Spartanizer[10]	Java	Sim		REFAC	AUT	https://github.com/SpartanRefactoring/Main
FENSKE[6]	Java			REFAC	SEMI/PP	

(SUG) Fornece Sugestão, (REFAC) Realiza Refatoração, (DETEC) Detecta Refatoração, (ERR) Detecta Erros em Refatoração, (IMPAC) Estima o Impacto das Refatorações, (PRE) Prediz se é Necessário Realizar Refatoração, (AUT) Automatizada, (SEMI) Semi-Automatizada, (PSE) Pseudo Automatizada, (PP) Passo a Passo

Ainda sobre a Tabela 2, pode ser observado que entre as estratégias de refatoração pode-se destacar: *Move Method*, *Pull Up Method*, *Extract Class* e *Clone* como sendo as principais estratégias que as ferramentas analisadas fornecem suporte.

Com a sumarização dos dados apresentados nas Tabelas 1 e 2, pode-se responder a questão de pesquisa: **Qual é o perfil das ferramentas de refatoração propostas recentemente?** Podemos destacar que a refatoração pode ser apoiada por ferramentas consolidadas na literatura. Principalmente para sistemas escritos na linguagem Java, várias ferramentas realizam a refatoração no código de forma automática. As estratégias de refatoração mais apoiadas são: *Move Method*, *Pull Up Method*, *Extract Class* e *Code Clone*.

5.2 Uma Análise do Uso da Biblioteca DBLP

Foi sumarizado a distribuição percentual de artigos das conferências selecionadas por base de dados. Artigos que foram encontrados apenas na biblioteca DBLP representam o percentual de 1,5%. Para artigos exclusivos das demais bases, o percentual é de 12,7%. Portanto, a interseção entre a DBLP e demais bases apresentadas foi de 85,7% dos trabalhos.

A biblioteca DBLP apresentou um percentual maior que 80% dos artigos de interesse retornados pelas outras bases e foi capaz de retornar alguns artigos que as demais bases não disponibilizaram. Portanto, foi considerado que essa biblioteca apresenta resultado satisfatório se comparada com as bases de dados tradicionais.

Tabela 2: Estratégias de refatoração suportadas

Nome do Programa	Move	Rename	Pull Up	Extract	Clone	Outros
LambdaFicator[12]						2
Cider[41]					Clone	
HAN[13]	MM					1
PILGRIM[47]		Rename	PUFE			1
Obey[15]	MM, MF					
WebDyn[38]						
KRISHNAN[23]					Clone	
JSRefactor[5]		Rename				
GLIGORIC[11]	Move, MTNF	Rename	Pull Up	EM, EL, ECO, ESUPER, EI, EC, EF, ELV		14
Xll[30] [31]		Rename				
MONDAL[36]					Clone	
BAVOTA[2]				EC		
Css-analyser[32]					Clone	
Morex[9]	MF, MM	RM, RP	PUM, PUFI	EC, EI		4
Ripe[3]	MF		PUM	EM, EC		
JExtract[42]				EM		
WANG[48]					Clone	
TAO[45]						3
HAN 2[14]	MM					
GAITANI[8]						1
Rename Expander[28]		Rename				
Morpheus[24]		RI		EF		1
Asynchronizer[27] [25]						
Asyncdroid[25] [26]						
Reflective Refactoring[20] [21]						
YANG[53]						1
Ad-Room[19]	MP			ESUPER, EC		
B-Refactoring[52]						
MORALES[37]						
MAZINANIAN[33]					Code Duplication	
OUNI[39]	MM, MF, MC		PUM, PUFI	EC, EM, EI		3
Null Terminator[34]						1
KEBIR[16]				ECOM, EI		1
MKAOUER[35]	MM ,MF, MC		PUFI, PUM	EC, EI		3
KHATCHADOURIAN[17]						
ZAFEIRIS[54]						1
C-JrefRec[46]	MM					
MSITI[5][18]						
More[40]	MM, MF, MC		PUFI, PUM	EM, EC, ESUPER, ESUB, EI		3
MANSOOR[29]		RC, RM, RF	PUM, PUFI	ESUB, ESUPER, EC		7
DALLAL[4]	MM					
RefDiff[44]	MT, MM, MF	RT, RM	PUM, PUFI	ESUPER, EM		3
Spartanizer[10]						
FENSKE[6]						2

(MM) *Move Method*, (MF) *Move Field*, (MTNF) *Move Type To New File*, (MP) *Move Property*, (MC) *Move Class*, (MT) *Move Type*, (RM) *Rename Method*, (RP) *Rename Parameter*, (RI) *Rename Identifier*, (RC) *Rename Class*, (RF) *Rename Field*, (RT) *Rename Type*, (PUFE) *Pull Up Feature*, (PUM) *Pull Up Method*, (PUFI) *Pull Up Field*, (EM) *Extract Method*, (EL) *Extract Local*, (ECO) *Extract Constant*, (ESUPER) *Extract Superclass*, (EI) *Extract Interface*, (EC) *Extract Class*, (EF) *Extract Function*, (ELV) *Extract Local Variable*, (ECOM) *Extract component*, (ESUB) *Extract Subclass*

6 AMEAÇAS A VALIDADE

Foi realizado um mapeamento da literatura e sumarização das ferramentas de refatoração disponíveis na literatura, conforme apresentado nas seções anteriores. No entanto, algumas ameaças à validade podem afetar nossos resultados de pesquisa. A seguir, discutimos cada um dos quatro tipos de ameaças listadas por Wolin et al. [50]: validade interna, conclusão, construção e validade externa.

Validade de construção. Realizamos nossos estudos para serem aplicados nas diferentes bases de dados sem realizar alterações nos padrões de buscas, sendo necessário se adequar apenas as normas de utilização da biblioteca, a fim de evitar a descaracterização do estudo. Realizamos a coleta dos dados de maneira similar nas bibliotecas disponíveis, um caso particular pôde ser observado na utilização da biblioteca DBLP, conforme foi descrito na Seção 3.2 as peculiaridades de sua utilização. Para proporcionar imparcialidade nos dados, foram utilizados os mesmos padrões de busca realizado nas demais bibliotecas. Portanto, é esperado o retorno dos dados mais uniformes possíveis. Finalmente, é proposto uma separação das informações apresentadas como Fase 1 e Fase 2, também apresentados na Seção 3.2 baseando o retorno dos dados para cada Fase identificada.

Validade interna. Os autores coletaram dados referentes as ferramentas de refatoração que foram encontradas na literatura. Porém, esses dados podem ser afetados pela classificação realizada pelos autores sobre a relevância dos artigos encontrados. Para minimizar esse problema a coleta e classificação dos dados foi realizada pelo autor 1 e validados pelo autor 2, detalhes sobre essa etapa pode ser identificado na Seção 3.3.

Validade de conclusão. Foi realizado a sumarização das novas ferramentas de refatoração encontradas, as quais foram disponibilizadas nas Tabelas 2 e 3. No entanto, esse procedimento pode ser afetado pela compreensão dos autores em relação a cada tópico proposto. Para minimizar esse problema, somente foi documentado as informações que estavam explícitas nos artigos utilizados no

estudo em caso de dificuldade de entender ou julgar a informação das ferramentas. As descobertas também podem ser afetadas pelas informações disponíveis nas bases de dados, podendo não indexar todos os artigos relevantes para o foco do estudo. Para minimizar esse problema, foram selecionadas 07 bases de dados diferentes e discutidas na Seção 3.3 quais as etapas utilizadas para realizar a junção dos dados.

Validade externa. Alguns fatores podem impedir a generalização dos dados encontrados na pesquisa. Por exemplo, as 44 ferramentas de refatoração sumarizadas neste estudo, podem não representar adequadamente todas as ferramentas de refatoração desenvolvidas. Restringimos as buscas realizadas neste estudo no período entre 2013 e 2017, podendo essa restrição afetar os resultados alcançados, uma vez que exista ferramentas de refatoração que foram desenvolvidas em outros períodos. No entanto, nossos estudos foram realizados com o objetivo de coletar informações recentes sobre novas ferramentas de refatoração que foram propostas na literatura. A *String* de busca utilizada para identificação primária dos artigos nas bases de dados utilizadas pode afetar os resultados encontrados, uma vez que o artigo apresentado seja de uma ferramenta de refatoração e seu título não apresentar a *String* buscada, afetará as informações encontradas. A data da pesquisa realizada na biblioteca pode afetar os resultados, tendo em vista que um artigo pode ter sido indexado após a data de coleta das informações utilizadas neste estudo.

7 TRABALHOS RELACIONADOS

Nesse trabalho foi apresentado um panorama das publicações com relação a refatoração dos últimos cinco anos, sendo o objetivo principal identificar as principais características das ferramentas que apoiam a refatoração em sistemas de informação. Desta forma, é apresentado nesta seção de trabalhos relacionados dois aspectos: I) impacto da refatoração para apoiar a qualidade de sistema de informação, II) revisão sistemática da literatura sobre refatoração.

I) Impacto da refatoração para apoiar a qualidade de sistema de informação: Existem vários trabalhos na literatura que remetem aos benefícios da refatoração no código fonte de um sistema de informação [3, 13, 29, 37]. Como por exemplo: Silva et al. [43] descreve um monitoramento de projetos Java hospedados no GitHub para detectar as refatorações feitas nesses projetos. Sabendo quais estratégias de refatorações foram aplicadas, o trabalho apresenta a sumarização do motivo por trás da decisão de refatorar o código, segundo as respotas dos desenvolvedores dos projetos. Como por exemplo em reposta a refatoração *Extract Method* que ajuda a retirar a redundância do código, pois o código duplicado precisa ser alterado em vários pontos causando inconsistência caso algum ponto não for alterado. No presente trabalho foi objetivo apresentar um catálogo das principais características das ferramentas de refatoração recentemente publicadas.

II) Revisão sistemática da literatura sobre refatoração: Dallal [1] descreve uma revisão sistemática da literatura sobre refatoração. O principal objetivo desse trabalho foi encontrar evidencias na literatura que indique as principais estratégias de refatoração utilizadas. Como resultado foi demonstrado que as estratégias de refatoração *Move Method*, *Extract Class* e *Extract Method* são mais frequentemente utilizadas em comparação a outras estratégias. O diferencial comparado com o presente trabalho foi identificar as principais estratégias de refatoração apoiadas por ferramentas. Desta forma, é possível que o desenvolvedor de sistema de informação possa escolher qual ferramenta utilizar segundo a necessidade de estratégias de refatoração específicas.

8 CONCLUSÃO

Neste artigo é realizado um mapeamento da literatura para identificação e caracterização de novas ferramentas de refatoração propostas na literatura. Para este propósito, foram coletados os dados disponíveis em bibliotecas digitais citadas na literatura e que indexam periódicos de eventos de impacto na área de estudo deste trabalho. Para obter informações mais atuais mantivemos o foco em publicações que foram realizadas entre os anos de 2013 e 2017. E, para coleta dos artigos foi utilizada uma *String* de busca "refactor*" para retornar os artigos que a continham em seu título.

Como resultado, inicialmente foi encontrado um total de 2800 artigos disponíveis pelas bibliotecas pesquisadas. Após as etapas de redução que foram realizadas, foram identificadas 44 ferramentas de refatoração, que foram caracterizadas e sumarizadas. Das ferramentas encontradas 37 delas oferecem suporte para a linguagem Java. Foi identificado também que 17 das ferramentas encontradas são *plugins*, sendo que 4 são protótipos; 21 ferramentas de refatoração já oferecem suas operações para serem realizadas de forma automatizada. Foi identificado que as ferramentas de refatoração identificadas oferecem grande suporte para estratégias de refatoração do tipo *Move, Rename, Pull Up, Extract* e *Clone*. Portanto, conclui-se que a área de refatoração ainda é discutida na literatura, novas ferramentas de refatoração estão sendo desenvolvidas e que em um número relevante estão sendo disponibilizadas para sua utilização de forma automatizadas.

Como trabalho futuro serão aplicadas algumas ferramentas em um estudo de caso no contexto industrial, para medir melhorias no código fonte, validando assim a viabilidade, usabilidade, e qualitatividade das ferramentas de refatoração que foram identificadas neste estudo.

ACKNOWLEDGMENTS

This research was partially supported by Brazilian funding agencies: CNPq (Grant 424340/2016-0) and FAPEMIG (Grant PPM-00651-17)

REFERÊNCIAS

[1] Jehad Al Dallal. 2015. Identifying refactoring opportunities in object-oriented code: A systematic literature review. *Information and Software Technology* 58 (2015), 231–249.
[2] Gabriele Bavota, Andrea De Lucia, Andrian Marcus, and Rocco Oliveto. 2014. Automating extract class refactoring: an improved method and its evaluation. *Empirical Software Engineering* (2014), 1617–1664.
[3] O. Chaparro, G. Bavota, A. Marcus, and M. D. Penta. 2014. On the Impact of Refactoring Operations on Code Quality Metrics. In *ICSME*. 456–460.
[4] Jehad Al Dallal. 2017. Predicting move method refactoring opportunities in object-oriented code. *IST* (2017), 105 – 120.
[5] Asger Feldthaus and Anders Moller. 2013. Semi-automatic Rename Refactoring for JavaScript. *SIGPLAN* (2013), 323–338.
[6] W. Fenske, J. Meinicke, S. Schulze, S. Schulze, and G. Saake. 2017. Variant-preserving refactorings for migrating cloned products to a product line. In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 316–326. https://doi.org/10.1109/SANER.2017.7884632
[7] Martin Fowler and Kent Beck. 1999. *Refactoring: improving the design of existing code.* Addison-Wesley Professional.

[8] Maria Anna G. Gaitani, Vassilis E. Zafeiris, N.A. Diamantidis, and E.A. Giakoumakis. 2015. Automated refactoring to the Null Object design pattern. *IST* (2015), 33 – 52.

[9] Adnane Ghannem, Ghizlane El Boussaidi, and Marouane Kessentini. 2014. Model refactoring using examples: a search-based approach. *JSEP* (2014), 692–713.

[10] Y. Gil and M. Orrù. 2017. The Spartanizer: Massive automatic refactoring. In *SANER*. 477–481.

[11] Milos Gligoric, Farnaz Behrang, Yilong Li, Jeffrey Overbey, Munawar Hafiz, and Darko Marinov. 2013. Systematic Testing of Refactoring Engines on Real Software Projects. In *ECOOP*.

[12] Alex Gyori, Lyle Franklin, Danny Dig, and Jan Lahoda. 2013. Crossing the Gap from Imperative to Functional Programming Through Refactoring. In *ESEC/FSE*. 543–553.

[13] Ah-Rim Han and Doo-Hwan Bae. 2013. Dynamic profiling-based approach to identifying cost-effective refactorings. *IST* (2013), 966 – 985.

[14] Ah-Rim Han, Doo-Hwan Bae, and Sungdeok Cha. 2015. An efficient approach to identify multiple and independent Move Method refactoring candidates. *IST* (2015), 53 – 66.

[15] H. C. Jiau, L. W. Mar, and J. C. Chen. 2013. OBEY: Optimal Batched Refactoring Plan Execution for Class Responsibility Redistribution. *TSE* (2013), 1245–1263.

[16] Salim Kebir, Isabelle Borne, and Djamel Meslati. 2017. A genetic algorithm-based approach for automated refactoring of component-based software. *IST* (2017), 17 – 36.

[17] R. Khatchadourian and H. Masuhara. 2017. Automated Refactoring of Legacy Java Software to Default Methods. In *ICSE*. 82–93.

[18] Raffi Khatchadourian and Hidehiko Masuhara. 2017. Defaultification Refactoring: A Tool for Automatically Converting Java Methods to Default. In *ASE*. 984–989.

[19] Djamel Eddine Khelladi, Reda Bendraou, and Marie-Pierre Gervais. 2016. ADROOM: A Tool for Automatic Detection of Refactorings in Object-oriented Models. In *ICSE*.

[20] J. Kim, D. Batory, and D. Dig. 2015. Scripting parametric refactorings in Java to retrofit design patterns. In *ICSME*. 211–220.

[21] Jongwook Kim, Don Batory, Danny Dig, and Maider Azanza. 2016. Improving Refactoring Speed by 10X. In *ICSE*. 1145–1156.

[22] Miryung Kim, Thomas Zimmermann, and Nachiappan Nagappan. 2012. A field study of refactoring challenges and benefits. In *SIGSOFT*. 50.

[23] G. P. Krishnan and N. Tsantalis. 2013. Refactoring Clones: An Optimization Problem. In *ICSME*. 360–363.

[24] J. Liebig, A. Janker, F. Garbe, S. Apel, and C. Lengauer. 2015. Morpheus: Variability-Aware Refactoring in the Wild. In *ICSE*. 380–391.

[25] Y. Lin and D. Dig. 2015. Refactorings for Android Asynchronous Programming. In *ASE*. 836–841.

[26] Y. Lin, S. Okur, and D. Dig. 2015. Study and Refactoring of Android Asynchronous Programming (T). In *ASE*. 224–235.

[27] Yu Lin, Cosmin Radoi, and Danny Dig. 2014. Retrofitting Concurrency for Android Applications Through Refactoring. In *SIGSOFT*. 341–352.

[28] H. Liu, Q. Liu, Y. Liu, and Z. Wang. 2015. Identifying Renaming Opportunities by Expanding Conducted Rename Refactorings. *TOSEM* (2015), 887–900.

[29] Usman Mansoor, Marouane Kessentini, Manuel Wimmer, and Kalyanmoy Deb. 2017. Multi-view refactoring of class and activity diagrams using a multi-objective evolutionary algorithm. *SQJ* (2017), 473–501.

[30] Philip Mayer and Andreas Schroeder. 2013. Towards Automated Cross-language Refactorings Between Java and DSLs Used by Java Frameworks. In *WRT*. 5–8.

[31] Philip Mayer and Andreas Schroeder. 2014. Automated Multi-Language Artifact Binding and Rename Refactoring between Java and DSLs Used by Java Frameworks. In *ECOOP*, Richard Jones (Ed.). 437–462.

[32] Davood Mazinanian, Nikolaos Tsantalis, and Ali Mesbah. 2014. Discovering Refactoring Opportunities in Cascading Style Sheets. In *SIGSOFT*. 496–506.

[33] Davood Mazinanian, Nikolaos Tsantalis, Raphael Stein, and Zackary Valenta. 2016. JDeodorant: Clone Refactoring. In *ICSE*. 613–616.

[34] Ş. Medeleanu and P. F. Mihancea. 2016. NullTerminator: Pseudo-Automatic Refactoring to Null Object Design Pattern. In *ICSME*. 601–603.

[35] Mohamed Wiem Mkaouer, Marouane Kessentini, Mel Ó Cinnéide, Shinpei Hayashi, and Kalyanmoy Deb. 2017. A robust multi-objective approach to balance severity and importance of refactoring opportunities. *ESE* (2017), 894–927.

[36] M. Mondal, C. K. Roy, and K. A. Schneider. 2014. Automatic Identification of Important Clones for Refactoring and Tracking. In *SCAM*. 11–20.

[37] R. Morales, A. Sabane, P. Musavi, F. Khomh, F. Chicano, and G. Antoniol. 2016. Finding the Best Compromise Between Design Quality and Testing Effort During Refactoring. In *SANER*. 24–35.

[38] H. A. Nguyen, H. V. Nguyen, T. T. Nguyen, and T. N. Nguyen. 2013. Output-Oriented Refactoring in PHP-Based Dynamic Web Applications. In *ICSME*. 150–159.

[39] Ali Ouni, Marouane Kessentini, Houari Sahraoui, Katsuro Inoue, and Kalyanmoy Deb. 2016. Multi-Criteria Code Refactoring Using Search-Based Software Engineering: An Industrial Case Study. *TOSEM* (2016), 23:1–23:53.

[40] Ali Ouni, Marouane Kessentini, Mel Ó Cinnéide, Houari Sahraoui, Kalyanmoy Deb, and Katsuro Inoue. 2017. MORE: A multi-objective refactoring recommendation approach to introducing design patterns and fixing code smells. *JSEP* (2017), 1843–1863.

[41] Mati Shomrat and Yishai A. Feldman. [n. d.]. Detecting Refactored Clones. In *ECOOP*, Giuseppe Castagna (Ed.). 502–526.

[42] Danilo Silva, Ricardo Terra, and Marco Tulio Valente. 2014. Recommending Automated Extract Method Refactorings. In *ICPC*. 146–156.

[43] Danilo Silva, Nikolaos Tsantalis, and Marco Tulio Valente. 2016. Why we refactor? confessions of github contributors. In *SIGSOFT*. ACM, 858–870.

[44] D. Silva and M. T. Valente. 2017. RefDiff: Detecting Refactorings in Version Histories. In *MSR*. 269–279.

[45] B. Tao and J. Qian. 2014. Refactoring Java Concurrent Programs Based on Synchronization Requirement Analysis. In *ICSME*. 361–370.

[46] N. Ujihara, A. Ouni, T. Ishio, and K. Inoue. 2017. c-JRefRec: Change-based identification of Move Method refactoring opportunities. In *SANER*. 482–486.

[47] J. von Pilgrim, B. Ulke, A. Thies, and F. Steimann. 2013. Model/code co-refactoring: An MDE approach. In *ASE*. 682–687.

[48] W. Wang and M. W. Godfrey. 2014. Recommending Clones for Refactoring Using Design, Context, and History. In *ICSME*. 331–340.

[49] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. ACM, 38.

[50] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[51] Zhenchang Xing and Eleni Stroulia. 2006. Refactoring practice: How it is and how it should be supported-an eclipse case study. In *ICSME*. 458–468.

[52] Jifeng Xuan, Benoit Cornu, Matias Martinez, Benoit Baudry, Lionel Seinturier, and Martin Monperrus. 2016. B-Refactoring: Automatic test code refactoring to improve dynamic analysis. *IST* (2016), 65 – 80.

[53] J. Yang, K. Hotta, Y. Higo, and S. Kusumoto. 2015. Towards purity-guided refactoring in Java. In *ICSME*. 521–525.

[54] Vassilis E. Zafeiris, Sotiris H. Poulias, N.A. Diamantidis, and E.A. Giakoumakis. 2017. Automated refactoring of super-class method invocations to the Template Method design pattern. *IST* (2017), 19 – 35.