Ministério da Educação

Universidade Tecnológica Federal do Paraná

Pró-Reitoria de Pesquisa e Pós-Graduação

Relatório de Atividades

Mapeamento Sistemático de Processos de Refatoração de Software

vinculado ao projeto

ARCHMAPE: Criação de uma arquitetura integrada de refatoração de código-fonte baseada em padrões de projeto e metapadrões.

Pedro Magnus Pedroso Nogueira

Bolsista Universidade Tecnológica Federal do Paraná

Ciência da Computação

Data de ingresso no programa: 03/2019

Profª. Drª. Simone Nasser Matos

Área do Conhecimento: Ciência Exata e da Terra – Sistemas de Computação

CAMPUS PONTA GROSSA, 2019

PEDRO MAGNUS PEDROSO NOGUEIRA

MAPEAMENTO SISTEMÁTICO DE PROCESSOS DE REFATORAÇÃO DE SOFTWARE

Relatório de Pesquisa do Programa de Iniciação Científica ou Programa de Iniciação Tecnológica da Universidade Tecnológica Federal do Paraná.

PONTA GROSSA, 2019

LISTA DE GRAFICOS

LISTA DE QUADROS

SUMÁRIO

INTRODUÇÃO

Refatoração é o processo de modificar um software de sem alterar seu comportamento externo, visando somente alterar o código para otimizar a sua estrutura interna Fowler [7]. Mesmo um sistema sendo projetado antes de ser programado, ocasionalmente problemas ou até mesmo aplicações erradas podem aparecer e esses "erros" são chamados de *bad smells.* A refatoração é aplicada para remover os *bad smells* fazendo com que o sistema tenha atributos de qualidade tais como flexibilidade, manutenibilidade e legibilidade.

A refatoração pode ser realizada por várias técnicas, como descrito neste trabalho, existem vários autores os quais criam diferentes técnicas para diferentes linguagens. A refatoração pode ser aplicada manualmente e automaticamente. A aplicação manual da refatoração não é recomendada, pois pode-se criar mais erros no código conforme a aplicação é realizada porque erros humanos acontecem, sem contar a demora na refatoração. A utilização e ferramentas é uma ótima saída para evitar erros humanos e ao mesmo tempo tornar a refatoração mais rápida [17].

Com o objetivo de conhecer o estado da arte sobre técnicas de refatoração de software, este relatório de pesquisa apresenta um mapeamento sistemático que foi realizado usando o método proposto Kitchenham [11], pois esse método é o mais citado na área de engenharia de software. Este método permite planejar o processo de revisão identificando o problema em que é definido como o alvo do mapeamento sistemático, questões que devem definidas e serem respondidas ao final de sua aplicação.

MATERIAS E MÉTODOS

Neste capitulo, serão apresentados o método utilizados para realizar o mapeamento sistemático baseados em Kitchenham [11].

a) Método de Pesquisa

Para a criação de trabalhos com uma revisão da literatura confiável é necessário estabelecer critérios sistemáticos de busca e análise dos estudos. Por isto, a revisão ou mapeamento sistemático procuram estabelecer etapas de planejamento, execução e análise para detalhamento do estado da arte em um determinado assunto.

Após a realização de pesquisas o método de revisão ou mapeamento sistemático sistemática, constatou-se que Kitchenham [11] é um método ideal para realização do estado da arte. Este método é o mais citado dentro da engenharia de software com 3.205 citações segundo o *Google Schoolar* (2019).

O Quadro 1 ilustra o resultado da busca realizada para encontrar métodos de revisão sistemática, bem como explicita o nome do artigo que aborda o método, os autores, a quantidade de citações, a área de aplicação e algumas considerações sobre o método.

Quadro 1 - Métodos para Revisão Sistemática

Nome do método	Nome do artigo	Autor	Citações	Área de aplicação	Observações
Revisão Sistemática(*Systematic Review*)	Guidelines for performing Systematic Literature Reviews in Software Engineering	B. Kitchenham	3385 – Segundo o google scholar	Engenharia de software	
Mapeamento Sistemático(*Systematic Mapping*)	Systematic Mapping Studies in Software Engineering	K Petersen	1552 – Segundo o google scholar	Engenharia de software	
Mapeamento Sistemático(*Systematic Mapping*)	Guidelines for conducting systematic mapping studies in software engineering: An update	K Petersen	404 – Segundo o google scholar	Engenharia de software	Este artigo é apenas um update do método anterior.

Revisão Sistemática(Systematic Review)	Systematic review in software engineering	J Biolchini	597 – Segundo o google scholar	Engenharia de software	O artigo é baseado no método criado por B. Kitchenham
Revisão Sistemática(Systematic Review)	Cochrane Handbook for Systematic Reviews of Interventions	Higgins JPT	39074 – Segundo o google scholar	Medicina	
PRISMA – Revisão Sistemática(Systematic Review)	the PRISMA statement for reporting systematic reviews and meta-analyses of studies that evaluate health care interventions: explanation and elaboration	Liberati A	17048 Segundo o google scholar	Medicina	Baseado no artigo acima

Fonte: Autoria própria

A seguir é descrito os passos do método escolhido [1] para realizar o mapeamento sistemático sobre refatoração de software.

b) Planejamento

O primeiro passo para realizar a revisão sistemática é verificar se a mesma é necessária. Esse passo é chamado de planejamento (*planning*). Para a realização da revisão sistemática primeiramente é preciso ver sua aplicabilidade e para isso Kitchenham [11] propõem uma série de perguntas, essas com o propósito de ajudar o pesquisador a verificar se o método é aplicável à sua pesquisa.

Desenvolvimento do Protocolo

O desenvolvimento do protocolo para a revisão sistemática inclui todos os componentes de uma revisão literária, por exemplo, se você procura todos os trabalhos relacionados a um tema, diferente da revisão literária o qual se pode procurar texto que somente concordam com a sua argumentação.

O mapeamento busca analisar o conceito e como será realizada a pesquisa, procurando ter controle sobre os locais de pesquisas tal como as *keywords*. Isto permite que se tenha o resultado mais justo possível, sem possuir polarização e sempre com a melhor qualidade possível.

O processo mais importante do protocolo é a formulação da pergunta para a pesquisa, com isso Kitchenham [11] propõem seguir as *guidelines* criadas pela *Australian NHMR* [1], mas com modificações feitas por ela para as *guidelines* se ajustarem na engenharia de software, já que elas foram inicialmente criadas para a medicina.

Estrutura da Pergunta

A estrutura da pergunta é composta por 4 aspectos, população, intervenção, resultado e *designs* experimentais. Os aspectos são:

- A população na engenharia de software depende do trabalho a ser realizado, podendo ser conceitos ou sistemas.
- A intervenção são as tecnologias específicas designadas para cada problema.
- Os resultados devem ser considerados como fatores importantes para o melhoramento da confiança em um certo assunto, como citado por Kichenham (2004), para reduzir os custos de produção e tempo para mercado. Todos os resultados devem ser especificados.
- *Designs* experimentais.

c) Execução

Quando o protocolo for definido a revisão pode finalmente começar e cinco pontos são importantes e explicados a seguir.

Identificação da Pesquisa

O alvo da revisão sistemática é encontrar a maior quantidade de estudos relatados a questão de pesquisa, procurando ser o mais justo possível, de forma a evitar qualquer polarização.

O primeiro ponto na pesquisa é delimitar as *keywords* usando várias combinações derivadas da pergunta. A pesquisa *on-line* pode ser utilizada no começo da pesquisa, mas uma pesquisa manual de diferentes procedências deve ser feita. Como citado por Kichenham [1] os resultados positivos são mais aceitos do que os negativos, então, evitam-se o uso de somente o que é concordante com nossas ideias para que o trabalho se torne melhor. Para a gestão da Bibliografia um gestor de referências é de ótimo uso.

Documentando a Pesquisa

A revisão sistemática ou o mapeamento sistemático deve ser transparente e com isso se criar uma documentação de pesquisa para que se possa obter todos os detalhes necessários da pesquisa.

Para essa documentação, uma tabela de duas colunas é criada em que na primeira coluna se tem as fontes e na segunda coluna de informações sobre a documentação tais como nome da base procurada, estratégia de pesquisa para cada base, data da pesquisa e anos cobertos.

Seleção de estudo

As pesquisas devem ser baseadas na pergunta da pesquisa, os artigos selecionados devem ser fundamentados em seus títulos e resumos para uma pré-seleção. Para a decisão de quais artigos serão selecionados, recomenda-se a opinião de uma especialista da área.

Qualidade do estudo

A definição de qualidade de um estudo é difícil de ser realizada, mas Kichenham (2004) propõem uma tabela de conceitos de qualidades baseadas na CRD Guidelines [3]

e em Cochrane Reviewers' Handbook [4]. A tabela é dividida em 3 colunas e 3 linhas sendo elas:

1. A primeira coluna é o Termo, composta pelas linhas: polarização, Validação interna, Validação externa.
2. A segunda coluna Sinônimos composta, pelas linhas: erro sistemático, validade, aplicabilidade.
3. A terceira e última coluna contém a Definição responsável pela explicação de cada uma das linhas anteriores.

Extração de dados

Explicação de como se realiza a extração de dados, e o que se deve evitar, contendo as informações padrões. O objetivo é gravar as informações obtidas do estudo primário e de forma a evitar parcialidade.

d) Síntese de Dados

É na síntese de dados que se coleta e resume os resultados. Existem dois tipos de síntese: a descritiva e a quantitativa. A síntese descritiva extrai informações de um estudo as quais devem ser armazenados seguindo um padrão de informações a serem armazenadas. A síntese quantitativa obtém os resultados de diferentes estudos a fim de os comparar. Kichenham (2004) aconselha o uso de *guidelines* usadas em medicina para realizar a medição de diferentes maneiras.

e) Reportando a revisão

Neste item a revisão deve ser reportada para a publicação em uma revista ou conferência. Ela propõe uma tabela com a estrutura e conteúdo da revisão sistemática.

RESULTADOS E DISCUSSÕES

Este capítulo relata os resultados do mapeamento sistemático referente a técnicas de refatoração de software, com exceção de refatoração baseada em padrões de projetos e em agentes, pois ambos já foram abordados em outros trabalhos.

a) Questões de Pesquisa

O principal objetivo da pesquisa é levantar técnicas de refatoração de software e as questões que foram identificadas são:

- Q1. Qual o número de citação dos trabalhos?
- Q2. Quais pesquisadores desenvolveram técnicas de refatoração no período definido?
- Q3. A qual grupo de técnicas a refatoração pertence?
- Q4. Quais trabalhos futuros podem ser realizados?
- Q5. Existe alguma relação entre os trabalhos selecionados?
- Q6. Quais ferramentas foram desenvolvidas?

Além dos objetivos principais outros benefícios podem ser extraídos das perguntas criadas como: a) Levantar quais técnicas de refatoração foram pouco exploradas pelos autores podendo encontrar uma tendência na escolha; b) Elencar os principais autores da área facilitando o processo de busca; c) Elaborar um critério de relevância dos trabalhos por citações em conjunto ao ano de publicação.

b) Bibliotecas e Modos de Pesquisa

Para encontrar os artigos a serem selecionados na revisão foram utilizadas repositórios de busca. A busca utilizando palavras-chave será usada para obter a maior quantidade de resultados possíveis, para que se possa ter uma maior abrangência dos trabalhos selecionados. As bibliotecas escolhidas foram, *Springer, ACM, SCIENCE DIRECT, WILEY, IEEEXplore* e *Google Scholar*.

A língua escolhida para a realização da pesquisa foi o Inglês. A escolha da língua é estratégica, pois o inglês é uma linguagem universal como descrito Nassi-Calò [5]:

> O inglês é indubitavelmente a língua franca da ciência mundial e mesmo que possa soar de certa forma injusto a autores e leitores de países cujo idioma nativo não é o inglês, é extremamente conveniente, pois permite que pesquisadores de todo o mundo se comuniquem, cooperem entre si e compartilhem o conhecimento.

c) Palavras-Chave e String de Busca

As palavras-chave foram discutidas e testadas nas bibliotecas. As palavras escolhidas foram: *Refatoring technique*, *bad smells*, *software refacoring*. A *string* de busca foi composta com base nas palavras-chave. Após alguns testes, a *string* de busca ficou sendo “Refactoring technique” AND “Bad smell”.

d) Critérios de Inclusão e Exclusão

Os seguintes critérios de inclusão de trabalhos foram adotados:

- possuir as palavras-chaves no título ou *abstract*,
- conteúdos relacionados com a refatoração de software por técnicas em seu *abstract* ou caso disponível introdução.

Para a exclusão foram considerados os seguintes critérios:

- Artigos duplicados;
- Artigos que não foram disponibilizados pelos autores;
- Artigos que não englobam os critérios de inclusão.

Para a escolha da melhor *string* todas as opções foram pesquisadas nas bibliotecas escolhidas. Os resultados estão disponíveis no quadro 2, mostrando a quantidade de artigos encontrados em cada biblioteca usando a respectiva *string*.

Quadro 2 - Lista de bibliotecas e trabalhos selecionados.

Keywords	Google Scholar	Springer	ACM	ScienceDirect	Wiley	Core
"Refactoring techniques"	1920	495	30	84	0	376
"Software Refactoring techniques"	43	595	2	0	0	5
"Refactoring technique" AND "Bad smell"	150	64	1	18	3	13

Fonte: Autoria Própria

Após a escolha da *string* de pesquisa, todos os trabalhos passaram método Inclusão/Exclusão. Foram encontrados 23 trabalhos referentes ao assunto da revisão sistemática e estão disponíveis na tabela 3 possuindo os autores, número de citações, ano de publicação e título.

Quadro 3 - Trabalhos Incluídos para Análise

Código	Autor	Título	Ano de Pulicação	Citações
S1	Mkaouer M, Kessentini M, Cinnéide M, Hayashi S, Deb K	*A robust multi-objective approach to balance severity and importance of refactoring opportunities*	2017	17
S2	Sangeetha M, Chandrasekar C	*An empirical investigation into code smells rectifications through ADA_BOOSTER*	2019	0
S3	Moghadam I, Cinnéide M	*Automated Refactoring Using Design Differencing*	2012	45
S4	Bavota G, De Lucia A, Marcus A, Oliveto R	*Automating extract class refactoring: an improved method and its evaluation*	2014	70
S5	Miguel Pessoa Monteiro	*Catalogue of Refactorings for AspectJ*	2004	13
S6	Ubayashi N, Piao J, Shinotsuka S, Tamai T	*Contract-Based Verification for*	2008	13

		Aspect-Oriented Refactoring		
S7	Srivisut K	*Definition and detection of bad smells of aspect-oriented program*	2007	9
S8	Rani A, Eng H	*Detection of bad smells in source code according to their object oriented metrics*	2014	3
S9	Rieß C, Heino N, Tramp S, Auer S	*EvoPat – Pattern-Based Evolution and Refactoring of RDF Knowledge Bases*	2010	33
S10	Kaur S, Res H	*Identification and refactoring of bad smells to improve code quality*	2015	3
S11	Huang J, Carminati F, Betev L, Luzzi C, Lu Y, Zhou D	*Identifying composite refactorings with a scripting language*	2011	1
S12	Schrijvers T, Serebrenik A	*Improving Prolog Programs: Refactoring for Prolog*	2004	16
S13	Panamoottil Krishnan G	*Improving the Unification of Software Clones using Tree and Graph Matching Algorithms*	2014	0
S14	Mkaouer M, Kessentini M, Bechikh S, Ó Cinnéide M, Deb K	*On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach*	2016	29
S15	Donald B Roberts	*Practical Analysis for Refactoring*	1999	444
S16	Zhang M, Baddoo N, Wernick P, Hall T	*Prioritising Refactoring Using Code Bad Smells*	2011	10
S17	Fowler M, Beck K	*Refactoring : improving the design of existing code*	1999	8364
S18	Correa A, Werner C	*Refactoring object constraint language specifications*	2007	29
S19	Opdyke W	*Refactoring object-oriented frameworks*	1992	1568

S20	Hanenberg S	*Refactoring of aspect-oriented software*	2013	141
S21	Wake W	*Refactoring workbook*	2004	191
S22	Wongpiang R, Muenchaisri P	*Selecting sequence of refactoring techniques usage for code changing using greedy algorithm*	2013	9
S23	Monteiro M, Fernandes J	*Towards a catalog of aspect-oriented refactorings*	2005	196
S24	Tom Schrijvers, Alexander Serebrenik, Bart Demoen	Refactoring Prolog code	2004	5

Fonte: Autoria Propria

e) Respostas as Questões

Os resultados para um mapeamento sistemático são apresentados respondendo as perguntas criadas no início de seu processo, os trabalhos serão referenciados pela chave atribuída no quadro 3.

Os estudos foram selecionados e a o seu conteúdo relevante foi colocado no formulário de extração. O formulário contém as seguintes informações: título, autores, ano de publicação, número de citações, linguagem utilizada, se possui uma ferramenta, técnicas utilizadas, se a técnica é baseada em outro autor, a contribuição do trabalho, trabalhos futuros, experimento. Este formulário não está disponível no neste relatório, porque seus dados estão disponíveis nas respostas das questões.

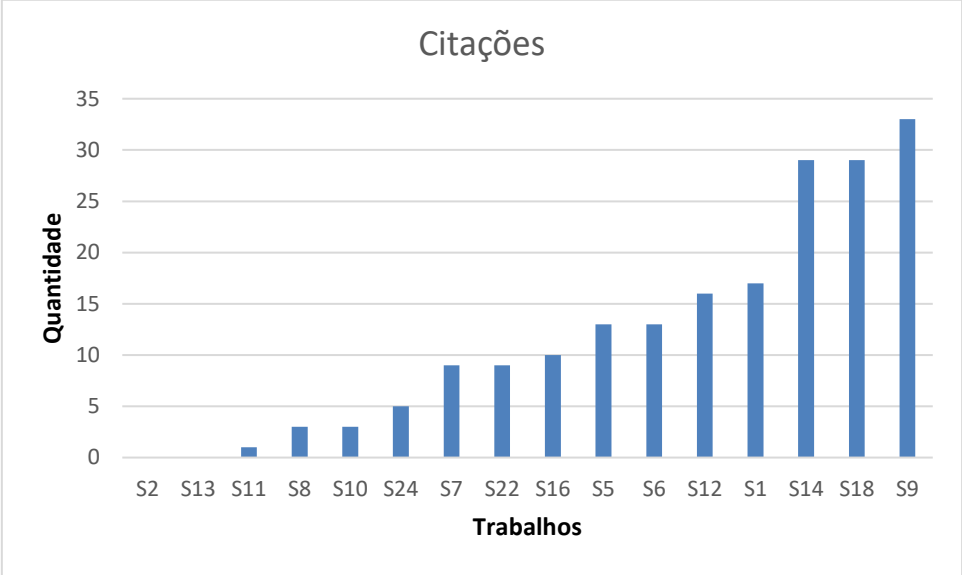
RQ1: Qual o número de citação dos trabalhos?

Dentre os artigos selecionados, os artigos que não possuem nenhuma citação são:

- *An empirical investigation into code smells rectifications through ADA_BOOSTER* (2019).
- *Improving the Unification of Software Clones using Tree and Graph Matching Algorithms* (2014).

Os trabalhos que possuem citações menores que 50 estão exibidas na tabela 1 e totalizam 17(dezessete) estudos.

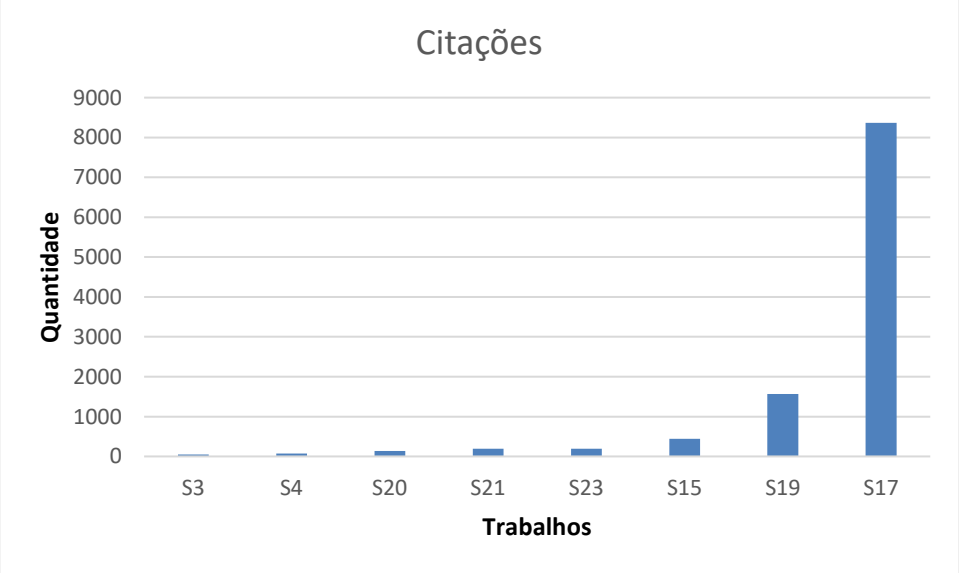
Gráfico 1 - Quantidade de citações menores que 50



Fonte: Autorai Própria

O Gráfico 2 exibe os demais trabalhos, em que as citações foram maiores do que cinquenta (50).

Gráfico 2 - Quantidade de citações maiores que 50



Fonte: Autoria Própria

Dois trabalhos possuem mais citações que os demais, S17 Fowler [7] com 8.364 citações, e S19 Opdyke [18] com 1.568 citações. Pesquisas realizadas por Fowler [7] e Opdyke [18] são referências importantes na área de refatoração de software.

RQ2: Quais pesquisadores desenvolveram técnicas de refatoração no período definido?

Dentre os artigos selecionados 19 (dezenove) autores criaram sua própria técnica de refatoração. Esses artigos são: S1, S2, S4, S5, S7, S8, S9, S10, S11, S12, S13, S14, S17, S18, S20, S21, S22, S23, S24.

Dentre os trabalhos que não criaram o método foram:

- S3: Neste artigo tem suas refatorações baseadas em Fowler [7], mas o artigo tem o foco na ordem de aplicação das refatorações. O mesmo foi testado com 6 aplicações *open-source* e 92% das refatorações propostas podiam ser aplicadas.
- S6: Neste artigo as refatorações são baseadas em Monteiro [15]. A proposta do artigo é verificar se a refatoração foi aplicada corretamente. Uma ferramenta foi criada e testada, nos testes foram descobertos que a ferramenta pode aplicar 67% do catalogo de Monteiro [15].
- S15: Neste artigo as refatorações foram baseadas no catalogo de Opdyke's [18]. A proposta do artigo é estender o trabalho de Opdykes's [18] adicionando pós-condições e criando uma ferramenta para aplicá-las.
- S16: Neste artigo as refatorações são baseadas em Fowler [7]. O trabalho procura fornecer ao desenvolvedor os *bad smells* para aplicação da refatoração.
- S19: Neste artigo as refatorações foram baseadas em Opdyke's [18]. O trabalho adiciona 3 *enabling conditions* e transforma as refatorações orientadas a objetos em refatorações para orientado a aspectos.

Como verificado acima os trabalhos com mais citações S19 e S17 são os trabalhos em que os pesquisadores mais se baseiam para a criação de uma ferramenta ou para melhor o método.

RQ3: A qual grupo de técnicas a refatoração pertence?

Existem vários tipos de refatoração, como de: classe, campo e método. Dentro dessas refatorações existem outras classes de refatoração, como refatoração de extração,

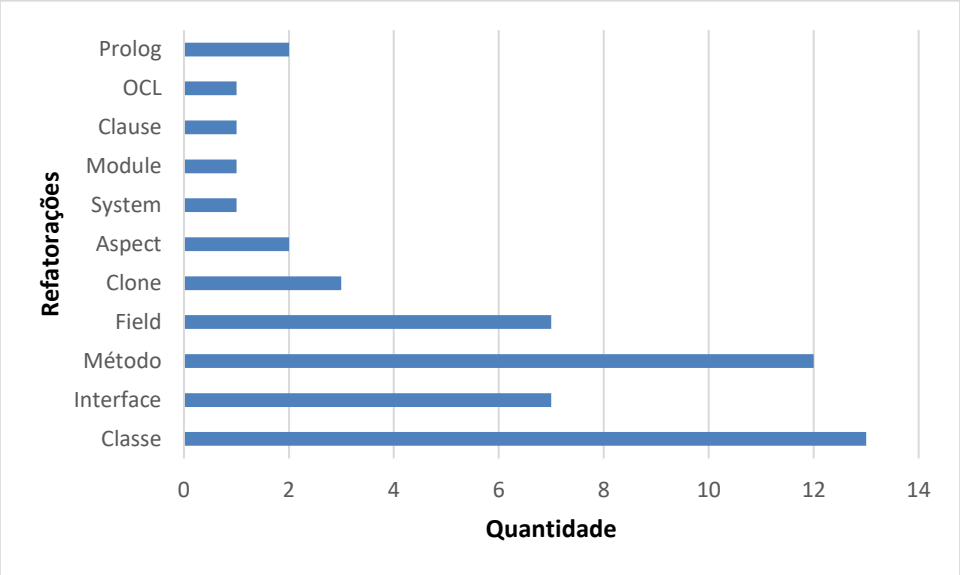
reestruturação e generalização. Para a classificação nesse trabalho será utilizada como classificação o “local” em que a refatoração é aplicada.

As refatorações são divididas em 11 grupos, refatoração de método, refatoração de classe, refatoração de interface e refatoração de campo (*field*), aspecto, *system*, *module*, *clause*, *clone*, OCL e Prolog.

Os trabalhos que possuem técnicas de refatorações do grupo de classe são: S1, S2, S3, S4, S5, S6, S8, S10, S14, S15, S17, S19, S21. Os que possuem técnicas refatoração do grupo de interface são: S1, S5, S6, S14, S15, S16, S17. Os estudos que possuem técnicas refatoração do grupo de método são: S1, S3, S5, S6, S8, S10, S14, S15, S17, S20, S21, S22. Os trabalhos que possuem técnicas refatoração do grupo de *field* são: S3, S5, S6, S14, S15, S16, S17. Os que possuem técnicas refatoração do grupo de *clone* são: S13, S16, S24. Os trabalhos que possuem técnicas refatoração do grupo de a*spect* são: S2, S7. O estudo que apresenta refatoração do grupo *system, module, clause* é: S12. O trabalho que possui técnicas refatoração do grupo de *OCL* é: S18. Os estudos que possuem técnicas refatoração do grupo de Prolog são: S16, S24.

O gráfico 3 exibe a quantidade de trabalhos por grupo de refatoração.

Gráfico 3 - Quantidade grupos de refatoração



Fonte: Autoria Própria

RQ4: Quais trabalhos futuros podem ser realizados?

Dentre os todos os trabalhos selecionados em 7(sete) não houve comentários sobre trabalhos futuros, sendo eles: S2, S4, S5, S16, S17, S20, S21. Os demais trabalhos tiveram comentários explicando os trabalhos futuros, sendo eles.

- S1: Adicionar suporte a outros *code smells* ao método.
- S3: Automação do processo e melhoria no design.
- S6: Criar uma composição automática do modelo que leva em conta os estados intermediários da refatoração.
- S7: Criar uma ferramenta, validar os *bad smells* propostos com outro AO software e outros atributos de qualidade.
- S8: Realizar uma comparação entre a ferramenta utilizada e a disponibilizada na IDE eclipse, as refatorações seriam aplicadas para remover os *bad smells*.
- S9: Recolher informações de recursos publicados na *Data Web.*
- S10: Realizar uma comparação entre a ferramenta utilizada e a disponibilizada na IDE eclipse, calcular mais métricas e detectar mais *bad smells*. Os métodos de refatoração serão aplicados na base dos *bad smells.*
- S11: Melhorar a automatização da identificação da refatoração.
- S12: Melhorar a automação da ferramenta para certas refatorações.
- S13: Ampliar a avalição técnica proposta no caso de *more challenging* de *type3-colnes.*
- S14: Melhorar o algoritmo NSGA-III.
- S15: Estender a pesquisa para mais linguagens, como Java e criar um ambiente visual que exponha várias partes do framework de transformação ao usuário.
- S18: Incorporar refatorações envolvendo semânticas complexas, desenvolver um agente refatorado para analisar e especificar os *bad smells* e adicionar um suporte semiautomático.
- S19: Melhorar o modelo criado, estudar técnicas de análise de programas entendendo as propriedades e componentes exclusivos da linguagem, estender as refatorações para modelos mais específicos de C++, refatoração de *designs* de interface.

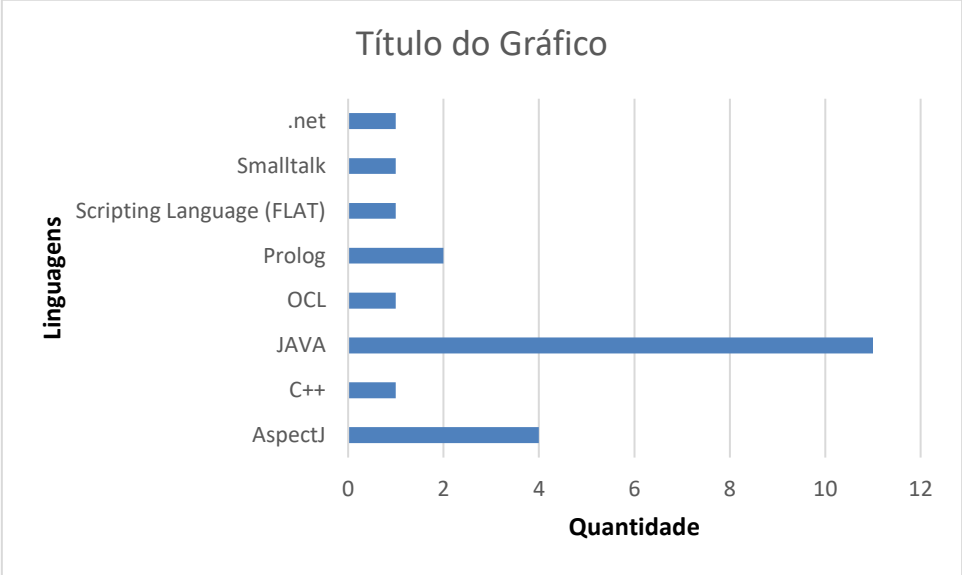
- S22: Comparar o algoritmo com outras técnicas de busca heurísticas sendo elas: A*, *hill climbing*, *Simulated Annealing* etc. Avaliar a abordagem com vários softwares para classificar o código-fonte para analisar qual característica pode usar técnicas de refatoração para melhorar o software e quais características não podem ser melhoradas.
- S23: Descobrir novos *smells*, testar as refatorações com mais casos, pensar em novas ideias de refatorações, cobrir aspectos específicos da linguagem e suporte a interfaces.
- S24: Aumentar o nível de automação de determinadas refatorações, informações adicionais como, tipos e modos que podem ser usados.

RQ5: Existe alguma relação entre os trabalhos selecionados?

Sim, 11(onze) trabalhos focam suas refatorações em códigos escritos em Java, 4 (quatro) em AspectJ, 2 em Prolog, 1em C++, 1 Smalltalk , 1 em .net, 1 em FLAT como apresentado no gráfico 4. Sendo:

1. Os artigos S1, S3, S6, S8, S9, S10, S13, S14, S16, S17, S21 utilizaram refatorações para códigos em Java.
2. Os artigos S5, S6, S20, S24 utilizam refatorações para códigos em AspectJ.
3. Os artigos S24 e S12 usam refatorações para códigos em Prolog
4. O artigo S19 utilizou refatorações para código em C++
5. O artigo S11 criou uma linguagem em formato *script* para aplicação de refatorações. Essa linguagem é chamada de FLAT.
6. O artigo S7 suporta refatorações para .net.
7. O artigo S15 utilizou refatorações para código em *Smalltalk*.
8. O artigo S18 utilizou refatorações para código em OCL.

Gráfico 4 - Linguagens de programação utilizadas



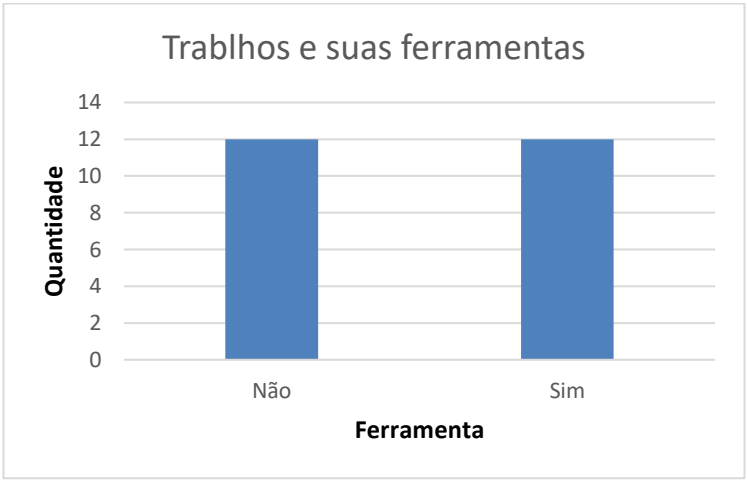
Fonte: Autoria Própria

Além dos artigos terem semelhanças nas suas linguagens, eles possuem um objetivo em comum, o de ajudar os desenvolvedores a melhorar seu código-fonte, utilizando uma ferramenta ou de realiza-la de forma manual.

RQ6: Quais ferramentas foram desenvolvidas?

Em relação aos trabalhos analisados, 12 (doze) eles criaram ferramentas para refatoração: S1, S3, S5, S7, S9, S11, S17, S18, S19, S21, S22 e S23. Os estudos que não desenvolveram ferramenta são: S2, S4, S6, S8, S10, S12, S13, S14, S15, S16, S20, S24.

Gráfico 5 - Trabalhos utilizando ferramentas.



Fonte: Autoria Própria.

Os trabalhos que criaram ferramentas têm o seguinte objetivo:

- S2: A ferramenta foi criada para testar as refatorações, ela não foi explicada no texto.
- S4: Sugere automaticamente uma decomposição adequada da classe original enquanto identifica o número apropriado de classes para refratorar.
- S6: Verifica se a refatoração aplicada foi usada corretamente.
- S8: Detecta *bad smells* automaticamente.
- S10: A ferramenta detecta *bad smells* e os refatora automaticamente com a técnica correta, ela possui interface gráfica.
- S12: Uma ferramenta implementada no editor VIM faz refatorações para Prolog.
- S13: Foi implementada no plug-in *JDeodorant*.
- S14: A ferramenta realiza refatorações utilizando o algoritmo NSGA-III para defini-las. A ferramenta demonstra a relevância do processo de refatoração.
- S15: Aplica a refatoração ao código.
- S16: Encontra *bad smells* e os orienta a prioridade de refatoração dos *smells*.
- S20: Refatora código em AspectJ em um *pug-in* para eclipse.
- S22: Refatora código em Prolog.

f) Trabalhos Relacionados

Durante a realização do mapeamento foram percebidos alguns pontos em comum nos trabalhos, como sua linguagem de programação, suas técnicas de refatoração e se eles utilizam alguma ferramenta.

A maior parte dos trabalhos utiliza a linguagem Java para refatoração com exceção dos trabalhos: S5, S6, S20, S24, S19, S12, S11, S15, S18. No Trabalho S2 não é comentado a utilização de uma linguagem de programação para refatorar.

Em relação as técnicas de refatoração grande parte utiliza refatorações do tipo classe, interface, método e *field*. Todos os detalhes das refatorações foram descritas na resposta à pergunta 3.

Em relação a ferramentas foi observado que metade dos trabalhos não possuem ferramentas e sim uma criação de uma técnica ou a melhoria de uma técnica já criada, mas em alguns desses trabalhos como S6 e S7 se propõem em criar uma ferramenta automática em trabalhos futuros.

CONCLUSÃO

Com a realização do mapeamento sistemático foi possível compreender e identificar as técnicas de refatoração que melhoram os código-fonte. A maioria das técnicas de refatorações foram criadas para linguagens orientada a objetos seguido por linguagem orientada a aspecto. As refatorações são aplicadas a *fields*, classes, código ou códigos repetidos. Notou-se que a grande maioria das técnicas foram criadas por Fowler [7].

Além de criar novas técnicas os pesquisadores estão desenvolvimento ferramentas para facilitar a aplicação da refatoração. O problema é que a maioria das ferramentas não possui uma iteração com o usuário, elas simplesmente aplicam a refatoração ao código sem realizar uma análise de qualidade no código em termos de atributos tais como manutenibilidade, flexibilidade e legibilidade .

AGRADECIMENTOS

Agradecimento à Universidade Tecnológica Federal do Paraná pela bolsa concedida para realização deste projeto. A Professora Dr.ª Simone Nasser Matos pela oportunidade, conhecimento compartilhado, comentários, críticas, sugestões e apoio.

REFERÊNCIAS

[1] AUSTRALIAN NATIONAL HEALTH AND MEDICAL RESEARCH COUNCIL. **How to review the evidence: systematic identification and review of the scientific literature.** National Health and Medical Research Council: Australia, 2000.

[2] BAVOTA, G. et al. Automating extract class refactoring: an improved method and its evaluation. **Empirical Software Engineering**, v. 19, n. 6, p. 1617–1664, 4 dez. 2014.

[3] CENTRE FOR REVIEWS AND DISSEMINATION. **Undertaking Systematic Review of Research on Effectiveness. CRD's Guidance for those Carrying Out or Commissioning Reviews.** Dissertação - University of York, 2001.

[4] CHANDLER, J; et al. **Cochrane Reviewers' Handbook.** The Cochrane Collaboration, 2003.

[5] CORREA, A.; WERNER, C. Refactoring object constraint language specifications. **Software & Systems Modeling**, v. 6, n. 2, p. 113–138, 4 jun. 2007.

[6] DONALD B ROBERTS. **Practical Analysis for Refactoring**. 127 p. 1999. Dissertação (Doutorado) - University of Illinois at Urbana-Champaign Champaign, IL, USA, 1999.

[7] FOWLER, M.; BECK, K. **Refactoring :** improving the design of existing code. Addison-Wesley: Boston, 1999.

[8] HANENBERG, S**. Refactoring of aspect-oriented software.** 17 p. 2003.Dissertação - University of Duisburg-Essen. Essen, 2003.

[9] HUANG, J. et al. Identifying composite refactorings with a scripting language. In: INTERNATIONAL CONFERENCE ON COMMUNICATION SOFTWARE AND NETWORKS, Xi'an. **Proceedings**... Xi'an (China): IEEE, 2011, p. 572-577.

[10] KAUR, S.; KAUR, H. **Identification and refactoring of bad smells to improve code quality.** 4 p. 2015. Dissertação - UCOE Punjabi University, Patiala, 2015.

[11] KITCHENHAM, B.; CHARTERS, S. **Guidelines for performing Systematic Literature Reviews in Software Engineering**. 2007.

[12] MKAOUER, M. W. et al. On the use of many quality attributes for software refactoring: a many-objective search-based software engineering approach. **Empirical Software Engineering**, v. 21, n. 6, p. 2503–2545, 23 dez. 2016.

[13] MKAOUER, M. W. et al. A robust multi-objective approach to balance severity and importance of refactoring opportunities. **Empirical Software Engineering**, v. 22, n. 2, p. 894–927, 4 abr. 2017.

[14] MOGHADAM, I. H.; CINNÉIDE, M. Ó. Automated Refactoring Using Design Differencing. In: EUROPEAN CONFERENCE ON SOFTWARE MAINTENANCE AND REENGINEERING, Szeged. **Proceedings**... Szeged (Hungary): IEEE, 2012.

[15] MONTEIRO, M, P. **Catalogue of Refactorings for AspectJ**. p. 50. 2004. Dissertação (Doutorado) - Escola Superior de Tecnologia de Castelo Branco Instituto Politécnico de Castelo Branco, Castelo Branco, 2004.

[16] MONTEIRO, M. P.; FERNANDES, J. M. **Towards a catalog of aspect-oriented refactorings**. In: INTERNATIONAL CONFERENCE ON ASPECT-ORIENTED SOFTWARE DEVELOPMENT, New York. **Proceedings**... New York, (USA): ACM Press, 2005, p. 111-122.

[17] MENS, T.; TOURWE, T. A survey of software refactoring. **IEEE Transactions on Software Engineering**, v. 30, n. 2, p. 126–139, fev. 2004.

[18] OPDYKE, W. **Refactoring object-oriented frameworks.** 206 p. Dissertação (Doutorado) - University of Illinois at Urbana-Champaign, Champaign 1992.

[19] PANAMOOTTIL KRISHNAN, G. **Improving the Unification of Software Clones using Tree and Graph Matching Algorithms.** 79 p. Dissertação (Mestrado) - Concordia University, Québec, 2014.

[20] RANI, A.; KAUR, H. Detection of bad smells in source code according to their object oriented metrics. **International Journal For Technological Research In Engineering**, June. 2014

[21] RIESS, C. et al. EvoPat – Pattern-Based Evolution and Refactoring of RDF

Knowledge Bases. In: INTERNATIONAL SEMANTIC WEB CONFERENCE, Shanghai. **Proceedings …** Shangahai (China): Springer, Berlin, Heidelberg, 2010. p. 647–662.

[22] SANGEETHA, M.; CHANDRASEKAR, C. An empirical investigation into code smells rectifications through ADA_BOOSTER. **Ain Shams Engineering Journal**, fev. 2019.

[23] SCHRIJVERS, T.; SEREBRENIK, A. Improving Prolog Programs: Refactoring for Prolog. In: INTERNATIONAL CONFERENCE ON LOGIC PROGRAMMING. Saint-Malo. **Proceedings**... Saint-Malo (France): Springer, Berlin, Heidelberg, 2004. p. 58–72.

[24] SCHRIJVERS, T.; SEREBRENIK, A.; DEMOEN, B. **Refactoring Prolog code**. 12 p. Dissertação - K.U. Leuven, Celestijnenlaan, 2004.

[25] SRIVISUT, K. Definition and detection of bad smells of aspect-oriented program. In: INTERNATIONAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE. Beijing. **Proceedings**… Beijing (China): IEEE, 2007.

[26] UBAYASHI, N. et al. Contract-Based Verification for Aspect-Oriented Refactoring. In: INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION, AND VALIDATION, Lillehammer. **Proceedings**... Lillehammer (Norway): IEEE, abr. 2008.

[27] WAKE, W. C. **Refactoring workbook**. Addison-Wesley: Boston, 2004.

[28] WONGPIANG, R.; MUENCHAISRI, P. Selecting sequence of refactoring techniques usage for code changing using greedy algorithm. In: INTERNATIONAL CONFERENCE ON ELECTRONICS INFORMATION AND EMERGENCY COMMUNICATION, Beijing. **Proceedings**... Beijing (China)IEEE, 2013.

[29] ZHANG, M. et al. Prioritising Refactoring Using Code Bad Smells. In: INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION AND VALIDATION WORKSHOPS, Berlin. **Proceedings**... Berlin (Germany): IEEE, 2011.

[30] ZIMMER, J. A. Graph Theoretical Indicators and Refactoring. In: CONFERENCE ON EXTREME PROGRAMMING AND AGILE METHODS, New Orleans. **Proceedings**… New Orleans (USA): Springer, Berlin, Heidelberg, 2003. p. 62–72.