Minerando Motivações para Aplicação de Extract Method: Um Estudo Preliminar

Jalisson Henrique
Universidade Federal da Bahia
Salvador, BA, Brasil
jalisson.santos@ufba.br

Marcos Dósea
Universidade Federal de Sergipe
Itabaiana, SE, Brasil
dosea@ufs.br

Cláudio Sant'Anna
Universidade Federal da Bahia
Salvador, BA, Brasil
santanna@dcc.ufba.br

RESUMO

As refatorações são operações realizadas no código fonte visando melhorar a capacidade de manutenção de um sistema de *software*. Apesar da literatura conter diversos estudos sobre as refatorações, são poucos os trabalhos que investigaram as motivações para tais refatorações ocorrerem. Esta pesquisa buscou investigar as motivações por trás das operações de *extract method* em um sistema real. Para isso, um estudo experimental foi conduzido, buscando minerar as refatorações do tipo *extract method* e analisar as mensagens de *commit* visando capturar as motivações por trás das refatorações. Também foi verificado a relação entre as refatorações e a métrica de número de linhas de código (LOC). Os resultados apontaram 11 motivações diferentes para a utilização do *extract method*. Além disso, foi visto que apenas 35% das refatorações que ocorreram foram com intenção de melhorar a qualidade do código. Também foi identificado que as refatorações com intenção de melhorar a qualidade do código ocorreram com maior frequência em métodos com valores de LOC na faixa de 21 a 40 e foram menos frequentes em métodos com valores entre 61 a 80.

PALAVRAS-CHAVE

Refatoração, Extract Method, Métricas de código

1 INTRODUÇÃO

Refatoração é a atividade que busca melhorar a qualidade de um código existente, mas sem alterar seu comportamento [5]. Elas são utilizadas para vários propósitos, como melhorar a capacidade de manuntenção, a compreensibilidade [3], complexidade, modularidade e reusabilidade [7].

Fowler [5] propôs um catálogo com diversos tipos de refatorações. Dentre eles, o *extract method* é um das mais populares [8] e utilizado [11]. O *extract method* tem como objetivo extrair um trecho ou fragmento de código fonte de um determinado método e levá-lo para um novo método [13].

Apesar da popularidade da refatoração de *extract method*, são poucos os trabalhos que investigaram as motivações por trás dessas refatorações. Silva et al. [11] coletaram explicações de desenvolvedores a respeito de refatorações que ocorreram, de modo a investigar as motivações para o uso de 12 tipos de refatorações, incluindo *extract method*. Já em Liu e Liu [6], os autores entrevistaram desenvolvedores profissionais buscando compreender as principais motivações para as operações de *extract method*.

Compreender as motivações por trás das operações de *extract method* é importante para auxiliar os pesquisadores a entenderem melhor como desenvolver ferramentas de recomendações de refatorações com resultados mais precisos. Deste modo, este trabalho tem como objetivo investigar as motivações para realizar as operações de *extract method*. Assim, um estudo experimental foi conduzido onde buscou-se, através da análise de mensagens de *commit* de um determinado sistema de *software*, extrair as motivações por trás das refatorações. Além disso, uma vez que as operações de *extract method* diminuem o tamanho do método ao extrair parte dele para outro método, buscou-se também avaliar se o tamanho do método pode ter relação com a motivação do desenvolvedor para usar essa refatoração. As seguintes questões de pesquisa foram formuladas:

QP1: ***As refatorações de extract method ocorrem com maior frequência em métodos com valores de LOC alto?***

Essa questão busca analisar se, independentemente dos motivos reportados nas mensagens de *commit*, as refatorações de *extract method* ocorrem com maior frequência em métodos longos, o que pode indicar que o número de linhas de código tem influência na motivação para refatorar um método.

QP2: ***O que os desenvolvedores reportam como motivação para realizar o extract method?***

Essa questão visa identificar e categorizar, com base nas mensagens de *commit*, as motivações por trás das operações de *extract method*.

QP3: ***Existe relação entre as motivações para refatoração extraídas das mensagens de commits e número de linhas de código do método refatorado?***

Essa questão pretende verificar as motivações para refatorar métodos em diversas faixas de valores de LOC, de modo a identificar em qual faixa de valores é mais comum ocorrerem refatorações com intenção de melhorar a qualidade do código.

Os resultados apontaram 11 motivações para utilização do *extract method*, sendo seis delas relacionadas com intenção de melhorar a qualidade do código existente, como, por exemplo, remover código duplicado, melhorar legibilidade e reduzir o tamanho do método. E mais cinco com outras intenções, como remoção de *bug*, adicionar *features* e melhorar o desempenho. Além disso, foi observado que as refatorações com intenção de melhorar a qualidade do código ocorreram com maior frequência em métodos com valores de LOC menores do que em métodos com LOC elevados.

O restante do artigo está estruturado da seguinte forma. A Seção 2 apresenta os trabalhos relacionados. Na Seção 3, é descrito o planejamento do estudo experimental. Na Seção 4, os resultados são descritos e discutidos. A Seção 5 discute as ameaças à validade. Finalmente, a Seção 6 apresenta a conclusão e os trabalhos futuros.

2 TRABALHOS RELACIONADOS

Alguns estudos na literatura já se propuseram a investigar as motivações por trás das refatorações. Silva et al. [11] buscaram investigar as motivações reais por trás das operações de refatorações aplicadas por desenvolvedores. Para isso, os autores monitoraram projetos

Java hospedados no GitHub e pediram que desenvolvedores explicassem os motivos por trás das decisões de refatorar o código. Assim, foi desenvolvido um catálogo com 44 motivações distintas para 12 tipos de refatorações. Os resultados apontaram que as refatorações ocorrem mais por mudanças de requisitos e menos para eliminar *code smells*.

Rebai et al. [10] analisaram mensagens de *commit* para extrair informações sobre intenções de refatorações, visando utilizá-las para melhorar recomendações sobre refatorações futuras. Os resultados mostraram evidências promissoras sobre a utilidade da abordagem baseada em mensagens de *commit*. Contudo, a análise foi realizada de forma automática considerando as mensagens que continham problemas de qualidade ou refatorações com base em uma lista de 87 palavras-chave.

Para investigar as motivações para a refatoração de *extract method*, Liu e Liu [6] realizaram entrevistas com diversos desenvolvedores, buscando compreender como são conduzidas as refatorações desse tipo. Os resultados apontaram que a reutilização atual, decomposição de métodos longos, resolução de clones e a reutilização futura são as principais motivações para o uso de *extract method*.

AlOmar et al. [1] buscaram identificar como os desenvolvedores documentam suas atividades de refatorações durante o ciclo de vida do *software*. Para isso, as mensagens de *commit* foram analisadas em busca de informações relevantes em relação às refatorações aplicadas. Os resultados mostraram que os desenvolvedores tendem a mencionar explicitamente a melhoria de atributos de qualidade e *code smells* nas mensagens de *commit*. Além disso, foi visto que mensagens com esses conteúdos tendem a ter atividades de refatorações mais significativas do que aquelas sem.

Paixão et al. [9] buscaram compreender o contexto e as motivações nos quais os desenvolvedores realizam rafatorações na revisão de código moderna. Para isso, um estudo experimental foi desenvolvido, onde mensagens de *commit* e discussões entre desenvolvedores foram inspecionadas. Os resultados apontaram que em apenas 31% das revisões de código que empregaram operações de refatorações, os desenvolvedores tinham intenção explícita de refatorar. No entanto, a abordagem considerou as refatorações de modo geral, não verificando tipos específicos de refatorações.

Nosso trabalho difere dessas pesquisas, pois buscamos analisar manualmente as mensagens de *commit* onde ocorreram refatorações específicas de *extract method*, buscando identificar as motivações por trás dessas operações e analisar a métrica de número de linhas de código para avaliar a relação entre as motivações e o tamanho do método.

3 PROJETO DO ESTUDO

Esta seção descreve as configurações do estudo realizado.

3.1 Sistema Alvo

A primeira etapa para realização deste estudo foi a escolha do sistema a ser analisado. Desse modo, buscamos um sistema orientado a objetos, desenvolvido na linguagem Java e com código disponível no repositório GitHub. O sistema utilizado nesse estudo foi o WebAnno[1], descrito em Castilho et al. [4], que se trata de uma ferramenta *Web*, *open source*. Esse *software* possui 802 classes, 5109 métodos e 48464 linhas de código. Como esse é um estudo preliminar, decidimos ainda usar apenas um sistema.

[1]https://github.com/webanno/webanno

3.2 Apoio Ferramental

Para identificar as refatorações realizadas no sistema alvo, foi utilizada uma ferramenta de mineração de refatorações chamada *RefactoringMiner*, desenvolvida por Tsantalis et al [12]. A *RefactoringMiner* analisa as alterações de código em repositório *Git* para detectar refatorações aplicadas ao longo da evolução do sistema. Ela suporta a detecção de até 40 tipos de refatoração para 6 tipos diferentes de elementos de código. Entretanto, este estudo considerou apenas a refatoração *extract method*. Mais especificamente, consideramos o que a *RefactoringMiner* indentifica como *EXTRACT OPERATION* ou *EXTRACT AND MOVE OPERATION*. As duas operações referem-se ao *extract method*, mas, na segunda, além de ser extraído, o método é movido para outra classe. Em um estudo, realizado pelos autores da ferramenta, que envolveu 7226 instâncias de refatorações diferentes [12], ela apresentou precisão de 99,6% e *recall* de 94%, de acordo com a validação de quatro especialistas em refatoração. Esse resultados foram melhores que todas as outras ferramentas avaliadas.

3.3 Procedimentos do Estudo

Para a coleta de dados, a RefactoringMiner foi utilizada no histórico de *commits* do sistema alvo desse estudo. Com isso, foram identificadas as refatorações de *extract method* que ocorreram ao longo dos *commits* no sistema. A ferramenta captura as refatorações e apresenta informações referentes a elas como: nome do método refatorado; nome da classe; nome do *commit*; tipo de refatoração; número de linhas de código; método para o qual o trecho de código foi movido; e a mensagem de *commit*.

Ao todo, a ferramenta detectou 411 refatorações de *extract method* que ocorreram no sistema. Com isso, buscamos em uma primeira etapa verificar a distribuição dos valores de LOC dos métodos refatorados, visando identificar em quais faixas de LOC as refatorações ocorrem com maior frequência.

Em um segundo momento, as mensagens de *commit* foram analisadas qualitativamente, de modo a extrair as motivações por trás dessas refatorações. Para isso, a análise temática [2] foi utilizada como metodologia. Essa análise foi desenvolvida da seguinte forma pelo primeiro autor do trabalho. Primeiramente, foi realizada uma leitura inicial de todas as mensagens de *commit*. Depois, foram atribuídos códigos para cada mensagem. Em seguida, foi feita uma busca por temas entre os códigos gerados anteriormente. Então, uma revisão dos temas foi realizada buscando juntar temas semelhantes e, finalmente, foram definidos os temas finais.

A partir dessa análise foi possível classificar as refatorações de *extract method* com base em categorias das motivações para a sua realização. A análise também permitiu identificar as refatorações cujas mensagens apontavam intenção de melhorar a qualidade do código e refatorações onde não havia essa intenção. Os códigos 'Sim' e 'Não' foram atribuídos para essas categorias, respectivamente.

Por fim, uma análise quantitativa foi realizada considerando a métrica de número de linhas de código (LOC) para analisar sua relação com as categorias encontradas nas mensagens analisadas, buscando investigar em quais níveis de LOC as refatorações com

intenção de melhorar a qualidade do código ocorrem com maior frequência. Todos os dados considerados no estudo estão disponíveis publicamente[2].

4 RESULTADOS

Nessa seção são apresentados os resultados obtidos no estudo experimental visando responder as questões de pesquisa.

4.1 Relação entre Refatorações e LOC

Esta seção procura responder QP1: As refatorações de *extract method* ocorrem com maior frequência em métodos com valores de LOC alto? Para isso, os valores de LOC dos métodos refatorados foram analisados através de um gráfico de distribuição de frequência, visto na Figura 1. No gráfico, os valores dos intervalos do eixo x foram gerados automaticamente através da ferramenta Google Sheets. Desse modo, é possível verificar que as refatorações ocorreram com maior frequência em métodos com valores de LOC na faixa entre 29 a 42. Contudo, é possível observar que um número grande de refatorações ocorreu em métodos com valores de LOC abaixo de 29 e também com valores acima de 42.

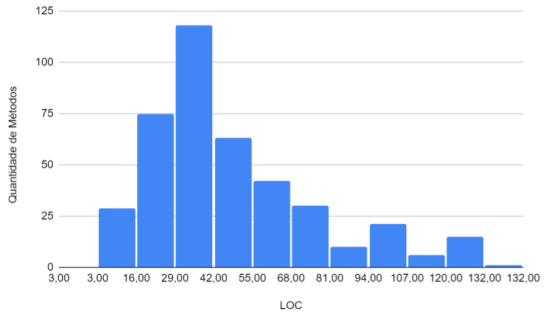

Figura 1: LOC de métodos refatorados

Curiosamente, esses resultados revelam que as refatorações não ocorreram apenas em métodos com valores de LOC altos, ou seja, em métodos longos, que a literatura considera como um *code smell* que pode ser alvo de *extract method* para ser removido [5]. Muitas das refatorações também ocorreram em métodos com valores de LOC baixo. Visto isso, identificar as motivações por trás das refatorações pode ajudar a compreender se há diferentes razões para os desenvolvedores refatorarem métodos com LOC baixo e métodos com LOC alto.

4.2 Motivações para Realizar *Extract Method*

Esta seção visa responder QP2: O que os desenvolvedores reportam como motivação para realizar o *extract method*? A partir da codificação e análise das mensagens de *commit* dos métodos refatorados, foi possível identificar as motivações que levaram à utilização do *extract method*. A Tabela 1 apresenta as onze diferentes motivações encontradas. A Tabela 1 também classifica as motivações nas categorias 'Sim' e 'Não', onde 'Sim' representa as refatorações onde as motivações apresentavam intenção explícita de melhorar a qualidade do código e 'Não' indica as refatorações que não tinham intenção de melhorar a qualidade do código. Além disso, para cada motivação é apresentado a descrição e o número de ocorrências.

Dentre as motivações identificadas, seis delas visam melhorar a qualidade do código. Dentro desse grupo, a motivação *Remover duplicação* é a que aparece com maior frequência (38 vezes). Ela representa refatorações para excluir trechos de código redundantes que se repetem em mais de um local. Alguns exemplos de trechos de mensagens de *commit* ilustram essa motivação: *"Extração de código redundante em métodos auxiliares como getCorrctionCas() ou actionRefreshDocument()"* ou *"Código movido para carregar restrições no serviço de repo em vez de tê-lo redundantemente em CorrectionPage, AnnotationPage, AutomationPage e CurationPage"*.

A segunda motivação mais frequente é a *Limpeza de código* (37 ocorrências), que representa as refatorações que ocorreram buscando remover trechos de códigos desnecessários ou que se encontravam legados. Isso pode ser visto em comentários como *"Código legado removido"* e também em *"Código refatorado para torná-lo mais conciso e remover código desnecessário"*.

Por sua vez, as refatorações para *Organizar o código* aparecem em terceiro lugar (34 ocorrências) dentre aquelas com intenção de melhorar a qualidade do código. Tal motivação visa organizar a estrutura do código, movendo métodos ou trechos de código para locais mais adequados no sistema. Comentários como *"Alguns métodos foram movidos para ficarem mais próximos de métodos semelhantes"* e *"Mover o método de renderização de BratAnnotationEditor para AnnotationEditorBase"* evidenciam essa motivação.

Outra motivação importante foi *Melhorar a legibilidade*, que ocorre em 15 ocasiões. São refatorações com intenção de tornar o código mais fácil de ser compreendido. Isso pode ser visto na mensagem: *"Refatorando a estrutura do projeto - rumo a actionSpanAnnotation() mais compreensível"*.

Além disso, as mensagens de *commit* possibilitaram verificar que muitas das refatorações que ocorreram não tinham a intenção de melhorar a qualidade do código. Essas refatorações ocorreram durante algumas manutenções que aconteceram no sistema. Dentro desse grupo, a *Atualização de Features* aparece com maior frequência (79 vezes). Elas são refatorações que ocorreram quando uma funcionalidade do sistema estava sendo modificada. O comentário a seguir exemplifica esse comportamento: *"Atualizar o painel do editor de recursos para levar em conta os efeitos das restrições, como slots adicionados automaticamente para vincular recursos"*.

Em seguida, *Adicionar Features* aparece na sequência com 37 ocorrências, representando as refatorações que ocorreram quando novas funcionalidades foram inseridas no *software*. Isso pode ser ilustrado pela mensagem: *"Novo tipo de recurso "link com função": Pode definir o slot agora, selecionando uma anotação existente ou selecionando um intervalo de texto"*.

A análise identificou que refatorações também ocorreram durante *Correção de bug* (21 ocorrências) e em manutenções que buscavam *Melhorar o Desempenho* do sistema (13 ocorrências). Isso pode ser visto, respectivamente, nas seguintes mensagens: *"Bug corrigido: os recursos do rótulo nunca foram definidos no adaptador diff"* e *"Melhore os tempos de carregamento do CAS evitando chamadas para getJCas() depois que o CAS é criado"*. Por fim, a motivação *Remoção*

[2]https://doi.org/10.5281/zenodo.5234317

Tabela 1: Motivações para utilização do *extract method*

Categoria	Motivação	Descrição	Ocorrrência
Sim	Remover duplicação	Remover trechos de código que se repetem em vários locais no sistema.	38 (9,25%)
Sim	Limpeza de código	Remoção de trechos de código legado ou desnecessário.	37 (9%)
Sim	Organizar código	Modificações para organizar a estrutura do sistema, como mover trechos de um método para outro método ou classe mais adequada.	34 (8,27%)
Sim	Melhorar a legibilidade	Alteração no método para torná-lo mais legível e fácil de compreender.	15 (3,65%)
Sim	Melhorar a testabilidade	Refatorações conduzidas a fim de tornar o método mais fácil de se testar.	14 (3,41%)
Sim	Reduzir o método	Refatorar o método para resolver o problema de smell Long Method.	5 (1,22%)
Não	Atualização de *Featues*	Refatorações que ocorreram durante a manutenção das funcionalidades.	79 (19,22%)
Não	Adicionar *Features*	Refatorações que ocorreram durante a inserção de novas funcionalidades.	37 (9%)
Não	Correção de *Bug*	Refatorações realizadas durante o processo de correção de bugs.	21 (5,11%)
Não	Melhorar o Desempenho	Refatorações que ocorreram em ações para melhorar a eficiência do processamento de determinadas funcionalidades	13 (3,16%)
Não	Remoção de *Features*	Remoção de funcionalidades no sistema.	12 (2,92%)
	Sem Categoria	Refatorações das quais não foi possível identificar seus objetivos.	106 (25,79%)

de Features aparece com 12 ocorrências. A Tabela 1 também apresenta o grupo *Sem Categoria* que representa as refatorações onde não foi possível identificar as motivações através das mensagens de *commit*, seja por falta de informações ou por pobreza de conteúdo.

Em resumo, em aproximadamente 35% das refatorações, a mensagem de *commit* apresentou intenção de melhorar a qualidade do código (categoria 'Sim'), enquanto que 40% delas foram realizadas quando a intenção principal não era melhorar a qualidade do código (categoria 'Não'). Por fim, não foi possível identificar o objetivo de 25% das refatorações.

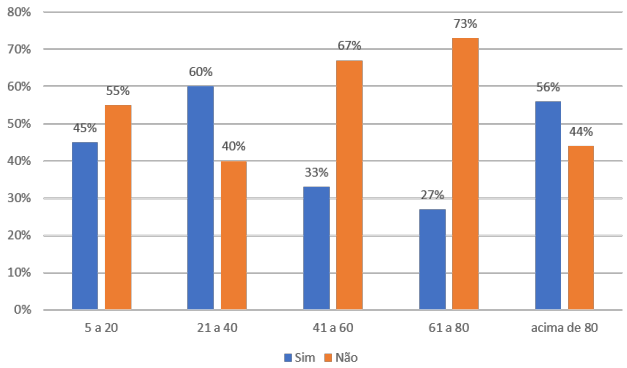

Figura 2: Intervalos de LOC

4.3 Relação entre Motivações e LOC

Esta seção visa responder QP3: Existe relação entre as motivações para refatoração extraídas das mensagens de *commits* e número de linhas de código do método refatorado? Para isso, foram definidos intervalos de valores de LOC para comparar os percentuais de refatorações de *extract method* que ocorreram com intenção ou sem intenção de melhorar a qualidade do código (categorias 'Sim' e 'Não' da Tabela 1, respectivamente). A Figura 2 apresenta um gráfico com cinco intervalos de LOC e a porcentagem de vezes em que refatorações de cada categoria aparece em cada intervalo. Os intervalos de LOC do gráfico foram definidos pelos próprios autores do artigo, por considerar, com base em sua experiência como programadores, que tais intervalos representam valores que diferenciam bem os tamanhos dos métodos entre si.

O gráfico mostra que, no primeiro intervalo (5 a 20), onde os valores de LOC são menores, 45% das refatorações buscaram melhorar a qualidade do código, enquanto que 55% não tinham essa intenção. Já o segundo intervalo, de métodos com valores de LOC entre 21 a 40, apresentou um percentual maior de refatorações com intenção de melhorar a qualidade: 60% contra 40%. Os intervalos com LOC de 41 a 60 e com LOC de 61 a 80 apresentaram baixo o número de refatorações classificadas como 'Sim', pois as refatorações que não tinham intenção de melhorar a qualidade do código ocorrerram em um percentual de 67% e 73%, respectivamente. Finalmente, o último intervalo (acima de 80) apresentou percentual de 56% para as refatorações classificadas como 'Sim', e 44% para aquelas classificadas como 'Não'.

Esses resultados possibilitaram observar que, mesmo em métodos com valores menores de LOC, ocorreram muitas refatorações com intenção de melhorar o código. Por outro lado, também foi identificado que, em métodos com altos valores de LOC, a maioria das refatorações que ocorreram não visavam melhorar a qualidade do código. Isso mostra um ponto curioso, pois esperava-se que, devido ao alto valor de LOC, as refatorações que seriam aplicadas nesses métodos seriam para buscar melhorá-los. Entretanto, foi visto que as refatorações com essa motivação ocorreram com maior frequência em métodos com tamanho na faixa de 21 a 40 linhas de código do que nos intervalos com valores maiores.

Além da análise com base na Figura 2, analisou-se também se existia diferença significativa em valores de LOC dos métodos que sofreram refatoração cuja mensagem de *commit* apresentava intenção explícita de melhorar a qualidade do código em comparação com os métodos onde a mensagem de *commit* não indicava essa intenção. Assim, foi construído um *boxplot*, visto na Figura 3, comparando

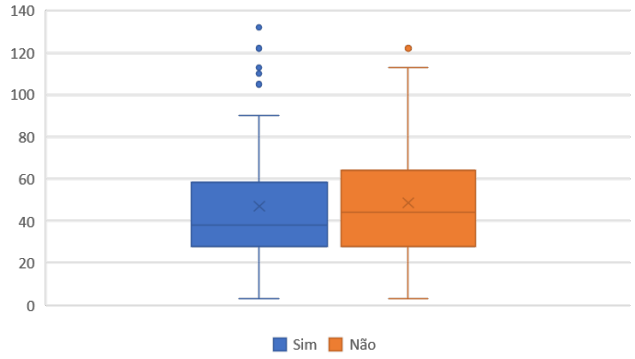

Figura 3: Distribuição de LOC

os valores de LOC dos métodos de ambas categorias. O gráfico mostra que não existe diferença significativa de LOC dos métodos analisados. Apesar das refatorações com intenção de melhorar a qualidade do código ocorrerem mais em métodos com valores de LOC menores, essa diferença não é estatisticamente significativa.

5 AMEAÇAS À VALIDADE

Esta seção apresenta as ameaças à validade do estudo e ações tomadas para minimizá-las.

Validade de Construto: Uma ameaça para esse estudo foi encontrar sistemas cujas mensagens de *commits* tivessem a descrição das atividades realizadas pelos colaboradores durante a manutenção. Em muitos sistemas, as mensagens de *commit* não apresentam conteúdos ou são pobres de informações, o que dificultaria nossa análise. Desse modo, avaliamos alguns sistemas antes de selecionarmos um que apresentava mensagens de *commit* que informavam relativamente bem o que foi feito naquele *commit*.

Validade de Conclusão: Aqui, a principal ameaça é a pequena quantidade de sistemas e *commits* considerados no estudo. Porém, esse é um estudo preliminar que servirá de base para outro estudo de larga escala, considerando diversos sistemas de *software*.

Validade Externa: Os resultados encontrados neste estudo não devem ser generalizados para outros sistemas. Para isso, seriam necessários mais estudos com uma quantidade maior de sistemas, considerando diferentes contextos e domínios.

6 CONCLUSÃO E TRABALHOS FUTUROS

Neste trabalho foi realizado um estudo experimental para analisar as refatorações de *extract method*, buscando compreender suas motivações e a relação dessas motivações com a quantidade de linhas de código dos métodos refatorados. Os resultados preliminares indicaram 11 tipos de motivações para a realização de *extract method*. Dentre os tipos de motivação, foi visto que seis deles visavam melhorar a qualidade do código. Dentre eles, os mais frequentes foram: remover código duplicado, realizar limpeza, organizar código e melhorar a legibilidade. Além disso, foi visto que muitas refatorações de *extract method* ocorreram durante atividades como inserção de uma nova funcionalidade, atualização de uma funcionalidade e até mesmo em manutenções para melhorar o desempenho do sistema ou para corrigir *bugs*.

O estudo também permitiu observar que, das refatorações que ocorreram, apenas 35% visavam melhorar o qualidade do código, enquanto que 40% apresentavam outras motivações. As mensagens de *commit* das 25% restantes não permitiram identificar a motivação. Por fim, os resultados apontaram que as refatorações ocorreram com maior frequência em métodos com valores de LOC na faixa de 29 a 42, e que as refatorações realizadas com intenção de melhorar a qualidade do código foram mais frequentes em métodos com números de linha de código entre 21 a 40 e menos frequente entre métodos com valores de 61 a 80.

Esses achados são importantes para melhorar a caracterização da aplicação de refatorações de *extract method*, o que pode contribuir para o desenvolvimento de ferramentas que forneçam maior produtividade e assertividade aos desenvolvedores de *software*. Como trabalhos futuros, pretende-se estender esse estudo para considerar uma quantidade maior de sistemas e também outros tipos de refatorações. Além disso, também seria interessante analisar atributos de qualidade como compreensibilidade e manutenibilidade, para verificar se as refatorações com intenção de melhorar a qualidade de código apresentam de fato melhorias significativas na qualidade do código em comparação com refatorações que não tem essa intenção.

Agradecimentos: O presente trabalho foi realizado com apoio da FAPESB (Fundação de Amparo à Pesquisa do Estado da Bahia).

REFERÊNCIAS

[1] E. AlOmar, M. W. Mkaouer, and A. Ouni. 2019. Can Refactoring Be Self-Affirmed? An Exploratory Study on How Developers Document Their Refactoring Activities in Commit Messages. In *2019 IEEE/ACM 3rd International Workshop on Refactoring (IWoR)*. 51–58. https://doi.org/10.1109/IWoR.2019.00017

[2] D. S. Cruzes and T. Dyba. 2011. Recommended Steps for Thematic Synthesis in Software Engineering. In *2011 International Symposium on Empirical Software Engineering and Measurement*. 275–284. https://doi.org/10.1109/ESEM.2011.36

[3] B. Du Bois, S. Demeyer, and J. Verelst. 2005. Does the "Refactor to Understand"reverse engineering pattern improve program comprehension?. In *Ninth European Conference on Software Maintenance and Reengineering*. 334–343. https://doi.org/10.1109/CSMR.2005.25

[4] R. Eckart de Castilho, E. Mújdricza-Maydt, S. Muhie, S. Hartmann, I. Gurevych, A. Frank, and C. Biemann. 2016. A Web-based Tool for the Integrated Annotation of Semantic and Syntactic Structures. The COLING 2016 Organizing Committee, Osaka, Japan, 76–84.

[5] M. Fowler. 1999. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Longman Publishing Co., Inc., USA.

[6] W. Liu and H. Liu. 2016. Major motivations for extract method refactorings: analysis based on interviews and change histories. *Frontiers of Computer Science* 10, 4 (2016), 644–656.

[7] T. Mens and T. Tourwe. 2004. A survey of software refactoring. *IEEE Transactions on Software Engineering* 30, 2 (2004), 126–139. https://doi.org/10.1109/TSE.2004.1265817

[8] E. Murphy-Hill, C. Parnin, and A. P. Black. 2012. How We Refactor, and How We Know It. *IEEE Transactions on Software Engineering* 38, 1 (2012), 5–18. https://doi.org/10.1109/TSE.2011.41

[9] M. Paixão, A. Uchôa, A. C. Bibiano, D. Oliveira, A. Garcia, J. Krinke, and E. Arvonio. 2020. Behind the Intents: An In-Depth Empirical Study on Software Refactoring in Modern Code Review. In *Proceedings of the 17th International Conference on Mining Software Repositories*. Association for Computing Machinery, 12. https://doi.org/10.1145/3379597.3387475

[10] S. Rebai, M. Kessentini, V. Alizadeh, O. Ben Sghaier, and R. Kazman. 2020. Recommending refactorings via commit message analysis. *Information and Software Technology* 126 (2020), 106332. https://doi.org/10.1016/j.infsof.2020.106332

[11] D. Silva, N. Tsantalis, and M. T. Valente. 2016. Why We Refactor? Confessions of GitHub Contributors. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Seattle, WA, USA) *(FSE 2016)*. Association for Computing Machinery, New York, NY, USA, 858–870.

[12] N. Tsantalis, A. Ketkar, and D. Dig. 2020. RefactoringMiner 2.0. *IEEE Transactions on Software Engineering* (2020), 1–1.

[13] M. T. Valente. 2020. *Engenharia de Software Moderna*. Independente. https://engsoftmoderna.info/