Sobre a Coocorrência de Refatoração de Teste e Código Fonte

Nicholas Alexandre Nagy
Concordia University
Montreal, Canadá
nicholas.a.nagy@protonmail.com

Rabe Abdalkareem
Carleton University
Ottawa, Canadá
rabe.abdalkareem@carleton.ca

ABSTRATO

A refatoração é uma prática difundida que visa ajudar a melhorar a qualidade de um sistema de software sem alterar seu comportamento externo. Na prática, os desenvolvedores podem realizar operações de refatoração no teste e no código-fonte. No entanto, enquanto trabalhos anteriores mostraram que a refatoração de código-fonte traz muitos benefícios, poucos estudos investigaram a refatoração de código de teste e se ela co-ocorreu com o código-fonte. Para examinar as refatorações concomitantes, realizamos um estudo empírico de 60.465 commits abrangendo 77 projetos Java de código aberto. Primeiro, analisamos quantitativamente os commits desses projetos para identificar os commits de refatoração co-ocorrentes (ou seja, os commits contêm refatorações realizadas no teste e no código-fonte). Nossos resultados mostraram que, em média, 17,9% dos commits de refatoração são commits de refatoração co-ocorrentes, o que é o dobro dos commits de refatoração somente de código de teste. Além disso, investigamos o tipo de refatoração aplicada ao código de teste nesses commits simultâneos. Descobrimos que Change Variable Type e Move Class são as refatorações aplicadas mais comuns. Em segundo lugar, treinamos classificadores de floresta aleatória para prever quando o código de teste de refatoração deve ocorrer junto com o código-fonte de refatoração usando recursos extraídos do código-fonte de refatoração em dez projetos selecionados. Nossos resultados mostraram que o classificador pode prever com precisão quando o teste e a refatoração do código-fonte coocorrem com valores de AUC entre 0,67-0,92. Nossa análise também mostrou que os recursos mais importantes em nossos classificadores estão relacionados ao tamanho da refatoração e à experiência de refatoração do desenvolvedor.

PALAVRAS-CHAVE

Refatoração de código fonte e teste, estudo empírico

Formato de referência ACM:
Nicholas Alexandre Nagy e Rabe Abdalkareem. 2022. Sobre a Coocorrência de Refatoração de Teste e Código Fonte In Proceedings of MSR '22: Proceedings of the 19th International Conference on Mining Software Repositories (MSR 2022). ACM, Nova York, NY, EUA, 5 páginas. https://doi.org/10.1145/ 1122445.1122456

1. INTRODUÇÃO

A refatoração é reconhecida como uma prática fundamental para manter uma base de código saudável. Na prática, os desenvolvedores principalmente refatoram seu código para lidar com mudanças (ou seja, novos recursos e correções de bugs [15]). Além disso, o design e os cheiros do código de teste desempenham um papel importante na qualidade do código de teste [7]. Em particular, os cheiros de teste são conhecidos por terem um alto taxa de sobrevivência e são difíceis de identificar pelos desenvolvedores [20]. Portanto, trabalhos anteriores focaram na detecção de cheiros e problemas de design no código de teste e ferramentas propostas para ajudar os desenvolvedores a identificá -los [1] e abordagens para refatorá-los [8, 22].

Como o código de teste é escrito para garantir que a funcionalidade do código-fonte esteja funcionando corretamente, é justo supor que o código-fonte e o código de teste estejam de alguma forma relacionados. Portanto, quando os desenvolvedores tratam de problemas de qualidade do código-fonte refatorando o código-fonte, eles também podem precisar refatorar o código de teste. No entanto, pouco conhecimento está disponível sobre esse relacionamento e quais fatores de refatoração de código-fonte influenciam a refatoração de código de teste, nem como ele pode ser usado para sugerir oportunidades de refatoração de código de teste.

Para tanto, iniciamos nosso estudo identificando os commits de refatoração em 77 projetos do conjunto de dados SmartSHARK [16]. Em seguida, determinamos se eles aplicaram refatorações de código-fonte, refatorações de código de teste ou ambos (ou seja, commits de refatorações co-ocorrentes) usando RefactoringMiner [19]. Nos casos de confirmação de refatoração concomitante, examinamos quais tipos de refatoração são aplicados ao código de teste. Em seguida, extraímos recursos de refatorações de código-fonte tanto dos commits que refatoram apenas o código-fonte quanto daqueles que ocorrem simultaneamente em dez projetos selecionados. Em seguida, treinamos classificadores Random Forest nesses recursos para prever se os commits devem ocorrer simultaneamente e refatorar o código de teste. Além disso, extraímos a importância do recurso de nossos classificadores para determinar quais recursos são mais valiosos na previsão de quando os commits devem ocorrer simultaneamente. Formulamos nosso estudo nos dois RQs a seguir:

RQ1: Com que frequência a refatoração de código-fonte e de teste ocorre simultaneamente **e quais são os tipos de refatoração aplicados ao código de teste?** Descobrimos que a maioria dos commits de refatoração apenas refatoram o código-fonte, representando 73,9% dos commits de refatoração para o projeto médio. Os commits de refatoração concomitantes também representam 17,9%, o que surpreendentemente é mais que o dobro dos 8,2% dos commits que refatoram apenas o código de teste. Além disso, descobrimos que Change Variable Type, Move Class e Rename Method são as refatorações mais comuns aplicadas ao código de teste nesses commits de refatoração coocorrentes.

RQ2: Podemos prever quando a refatoração do código de teste deve ocorrer em conjunto com a refatoração do código-fonte? Por meio de nossos classificadores construídos, pudemos prever com precisão quando a refatoração de teste e código-fonte ocorre com valores de AUC entre 0,67-0,92 e Precisão mediana, Recall e F1-score iguais a 0,70, 0,63, 0,66, respectivamente nos dez estudados projetos. Além disso, nossa análise mostrou que os recursos mais importantes para nosso classificador preditivo estão relacionados ao tamanho da refatoração, experiência de refatoração do desenvolvedor e histórico de refatoração de arquivos.

Por fim, disponibilizamos nossos scripts e dataset online [12].

2 PROJETO DE ESTUDO DE CASO

Nosso principal objetivo é investigar a coocorrência do código-fonte e testar as alterações de refatoração de código com base na granularidade do nível de confirmação. Assim, primeiro queríamos diferenciar entre commits que executam

MSR 2022, 23 a 24 de maio de 2022, Pittsburgh, PA,
EUA © 2022 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/AA/MM. . . $ 15,00
https://doi.org/10.1145/1122445.1122456

refatorações para testar código, código-fonte ou ambos. Em seguida, investigamos as ocorrências de cada tipo de commit de refatoração e os tipos de operações de refatoração aplicadas para testar o código em commits coocorrentes. Em segundo lugar, extraímos recursos relacionados ao commit para construir classificadores para prever quando a refatoração do código de teste deve acompanhar a refatoração do código-fonte e examinamos os recursos mais importantes.

2.1 Projetos estudados Como nossa análise visava investigar a coocorrência de oportunidades de refatoração de teste e código-fonte, precisávamos estudar projetos com histórico de desenvolvimento suficiente. Assim, recorremos ao conjunto de dados SmartSHARK [16]. O SmartSHARK é um conjunto de dados disponível publicamente que fornece informações detalhadas sobre o histórico de desenvolvimento de 77 projetos Java. Obtivemos o dump de dados SmartSHARK MongoDB publicado em 2021-09-08 [17]. O conjunto de dados combina metadados de várias fontes, como Git e issue sistemas de rastreamento. Além disso, o conjunto de dados fornece informações sobre alterações de código, principalmente as refatorações realizadas, que são essenciais para nosso estudo. Clonamos e analisamos os 77 projetos.

2.2 Identificando commits relacionados à refatoração Uma vez que obtivemos os 77 projetos, queríamos determinar quais commits nesses projetos realizavam refatorações. Como o conjunto de dados SmartSHARK fornece essas informações, começamos examinando as informações de refatoração coletadas. O conjunto de dados SmartSHARK detectou refatorações no nível de commits usando duas ferramentas diferentes, RefactoringMiner [19] e RefDiff [14]. Neste estudo, decidimos focar no estudo das refatorações identificadas pelo Refactor ingMiner, uma vez que 1) trabalhos anteriores mostraram que ele tem melhor recall e precisão e 2) a versão mais recente (ou seja, versão 2.2 [18]) suporta a identificação de 85 diferentes tipos de refatoração, que não são suportados na versão mais recente do RefDiff [19].

Como queríamos identificar se uma refatoração é aplicada ao código-fonte ou ao teste, precisávamos da confirmação associada e das informações de arquivo para as quais ela foi identificada. No conjunto de dados SmartSHARK, cada refatoração tem um commit-id, o que nos permitiu descobrir em qual commit o refator foi identificado. No entanto, encontrar os arquivos afetados por uma refatoração daqueles que o RefactoringMiner identificou não foi uma etapa simples. Assim, decidimos usar as informações de campo hunks fornecidas no conjunto de dados para cada operação de refatoração, que o RefactoringMiner forneceu.

No entanto, ao examinar as informações de hunk no conjunto de dados SmartSHARK , notamos que algumas informações de hunk estão ausentes (ou seja, alguns documentos de refatoração na coleção estavam faltando campos de hunk). Isso talvez se deva à complexidade de vincular as alterações de refatoração identificadas pelo RefactoringMiner aos hunks identificados em cada commit pela ferramenta vcsSHARK [16].

Portanto, não podíamos usar de forma confiável os fragmentos da coleção de refatoração do conjunto de dados, então recorremos à execução do RefactoringMiner nos commits de refatoração identificados dos 77 projetos estudados. Ao final desta etapa, conseguimos extrair as informações do arquivo para cada operação de refatoração, que usamos na próxima etapa para determinar se uma refatoração em um commit foi realizada no teste, no código-fonte ou em ambos. A Tabela 1 mostra as estatísticas resumidas de todos os commits e os commits relacionados à refatoração identificados e sua porcentagem nos 77 projetos. Observamos que na mediana 16,7% (média = 16,9%) dos commits nos projetos estudados estão realizando refatorações.

Tabela 1: Resumo de todos os commits e commits de refatoração.

Comprometer-se	Min.	Mediana	Significa	Máx.
Total	3.380	4.856	21.365	(16,9) 3.239 (36,2)
Refatorar (%)	1 (0,0)	579 (16,7)	758,2	

2.3 Categorizando commits relacionados à refatoração Uma vez que identificamos os commits que realizam refatorações, agora queríamos determinar se as refatorações executadas são feitas no código de teste, código-fonte ou ambos, e identificar os tipos de refatoração aplicados. Conforme descrito na Seção 2.2, primeiro executamos o RefactoringMiner em cada confirmação de refatoração identificada pelo conjunto de dados. Esta etapa resultou em uma lista de refatorações detectadas para cada commit e suas informações associadas, como o tipo (por exemplo, move class, etc.), locais do lado esquerdo (por exemplo, locais do commit pai), locais do lado direito (por exemplo, locais de commit filho) e a descrição da refatoração. Além disso , com base nas informações do local da refatoração, conseguimos determinar quais arquivos foram afetados pela refatoração e usamos o caminho para esses arquivos para identificar se a refatoração está sendo aplicada ao código de teste ou ao código-fonte.

Para determinar se uma refatoração foi realizada no código de teste ou no código-fonte, aplicamos a mesma abordagem de detecção usada em [3, 13] em cada arquivo tocado pelas operações de refatoração . A abordagem é composta principalmente de duas etapas. O primeiro passo é examinar se o nome do arquivo começa ou termina com o palavra “teste”. Segundo, se este for o caso, então usamos JavaParser [21] para analisar o arquivo Java (ou seja, arquivos que terminam em “.java”). Então, com o arquivo analisado, conseguimos eliminar arquivos de teste com erros de sintaxe e métodos de teste baseados em JUnit detectados, o que reduz significativamente os arquivos de teste falso-positivos [3]. Seguindo essas etapas, se algum dos arquivos tocados pela refatoração for identificado como um arquivo de teste, classificamos a refatoração como uma refatoração de teste.

Uma vez que temos os dados de refatoração para cada commit, começamos a categorizar todos os commits com base na atividade de refatoração aplicada. Em nosso estudo, identificamos três categorias para os commits examinados: commits de refatoração de teste, commits de refatoração de origem e commits de refatoração coocorrentes. Cada confirmação em nosso conjunto de dados será categorizada como um desses três tipos. Abaixo, fornecemos mais detalhes sobre cada tipo de confirmação: • Confirmação de refatoração de teste: Para esse tipo de confirmação, todas as operações de refatoração aplicadas são aplicadas no código de teste. • Source Refactoring Commit: Para este tipo de commit, todos aplicados operações de refatoração são aplicadas no código-fonte.
• Confirmação de refatoração concomitante: para esse tipo de confirmação, existem diferentes operações de refatoração que são aplicadas no código-fonte e no código de teste.

3 RESULTADOS DO ESTUDO DE CASO

Nesta seção, apresentamos os resultados do nosso estudo de caso.

RQ1: Com que frequência a refatoração de código-fonte e de teste ocorre simultaneamente e quais são os tipos de refatoração aplicados ao código de teste?

Motivação: A refatoração é um assunto muito estudado entre os pesquisadores, e a maioria dos trabalhos anteriores examinaram a refatoração quando aplicada ao código-fonte (por exemplo, [2, 11]). No entanto, há pouco trabalho feito sobre as diferentes refatorações aplicadas ao código de teste, particularmente no código de teste que ocorre em conjunto com a refatoração do código-fonte (por exemplo, [3]). Além disso, como a qualidade do código de teste é importante [4, 10, 20], nos propusemos a

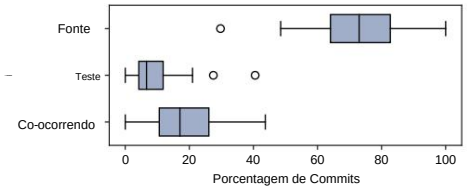

Figura 1: Boxplots representando a % de commits com essa classificação sobre os projetos com commits de refatoração. investigar refatorações de código de teste que ocorrem concomitantemente com refatorações de código-fonte.

Abordagem: Conforme explicado anteriormente, aplicamos uma abordagem semelhante de estudos anteriores que usaram os padrões JUnit para definir o que é um arquivo de teste [3, 13]. Nessa abordagem, usamos o RefactoringMiner para primeiro determinar quais arquivos cada refatoração tocou e sinalizar aqueles que tocaram nos arquivos de teste. Em seguida, conseguimos identificar quais refatorações são aplicadas aos testes e quais são aplicadas apenas ao código-fonte.

Com este método, ao analisar os commits com o Refactor ingMiner, categorizamos cada commit em nosso conjunto de dados em; testar confirmações de refatoração, confirmações de refatoração de origem ou confirmações de refatoração simultâneas. Em seguida, calculamos a porcentagem de cada tipo de confirmação em cada projeto em nosso conjunto de dados. Além disso, para determinar os tipos de refatorações aplicadas a testes que ocorrem simultaneamente com refatorações de código-fonte, usamos as refatorações de teste que ocorrem em commits que também aplicam refatorações ao código-fonte. Assim, extraímos o relatório de tipo de refatoração do RefactoringMiner e, para cada tipo, calculamos sua frequência por projeto em nosso conjunto de dados.

Resultado: A Figura 1 mostra as distribuições da porcentagem dos commits de refatoração de teste, fonte e coocorrência em todos os projetos estudados. Embora a maioria das refatorações sejam aplicadas no código-fonte, instâncias de todos os tipos de refatoração detectáveis pelo RefactoringMiner podem ser encontradas no código de teste que ocorre concomitantemente com a refatoração do código-fonte tocar. Conforme mostrado na Figura 1, além de alguns valores discrepantes, a maioria dos projetos refatora o código-fonte com mais frequência do que o código de teste. Apesar disso, observamos uma grande variedade de tipos de refatoração sendo aplicados. Na Figura 1, os commits de refatoração de código-fonte representam 73,9% em média, em comparação com os 8,2% e 17,9% em média para commits de refatoração de teste e commits simultâneos, respectivamente. Apesar disso, ainda há uma pequena parcela de projetos que levam a sério a refatoração de código de teste. Mesmo ao refatorar o código-fonte simultaneamente, todo tipo de refatoração é observado na refatoração do código de teste. Com isso em mente, ainda há uma quantidade limitada de tipos de refatoração que ocorrem simultaneamente no código de teste na maioria dos projetos. Na Figura 2, mostramos os dez principais tipos de refatoração mais coocorrentes por projeto com base na mediana. Observe que mostramos apenas os dez primeiros devido ao espaço limitado do artigo (uma lista completa está em nosso pacote de replicação). A Figura 2 mostra que o tipo de refatoração mais frequente aplicado ao código de teste em confirmações de refatoração coocorrentes são o tipo de variável de mudança, a classe de movimento e o método de renomeação.

> Para a maioria dos projetos, commits de refatoração coocorrentes são infrequentes em comparação com refatorações de código-fonte, e a variedade de refatorações aplicadas a esses commits é limitada.

RQ2: Podemos prever quando a refatoração do código de teste deve ocorrer com a refatoração do código-fonte?

Motivação: Como pode ser visto na RQ1, a refatoração de código-fonte é uma atividade mais comum para desenvolvedores do que a refatoração de código de teste.

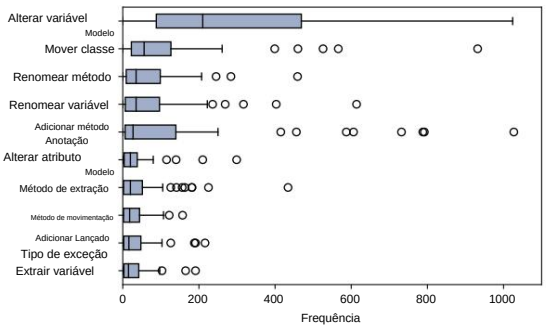

Figura 2: Boxplots de tipos de refatoração aplicados a testes em commits de refatoração coocorrentes.

Apesar disso, o código de teste ainda é vulnerável a vários cheiros que impedem a qualidade geral dos testes (por exemplo, [20]). Para esse fim, tentamos usar as informações dos commits de refatoração do código-fonte para ver se podemos prever se a refatoração do código-fonte deve ser acompanhada de refatorações de teste.

Abordagem: Treinamos classificadores que podem identificar automaticamente commits de refatoração concomitantes. Primeiro, nossos classificadores recebem como entrada uma lista de commits que são rotulados como commits de refatoração de código-fonte ou commits de refatoração coocorrentes (veja a Figura 1). Em seguida, nossos classificadores são treinados com base em recursos extraídos das refatorações aplicadas apenas no código-fonte. Neste estudo, propusemos usar diferentes recursos que acreditamos que podem dar uma boa indicação de se o código de teste deve ser refatorado quando o código-fonte é. Nós primeiro utilizou todas as informações fornecidas pela ferramenta RefactoringMiner [19]. Em segundo lugar, extraímos outros recursos (por exemplo, as experiências de refatoração dos desenvolvedores). É importante notar que todos os recursos estudados são extraídos das refatorações do código-fonte em commits de refatoração de código-fonte ou commits de refatoração coocorrentes. Por uma questão de concisão, apresentamos os recursos usados na

Como nosso conjunto de dados contém vários projetos e com base em nossos resultados para o primeiro RQ, alguns desses projetos têm um número muito pequeno

Tabela 2: Recursos extraídos das alterações aplicadas ao código-fonte somente em commits de refatoração de código-fonte e commits de refatoração coocorrentes. *Esses recursos descrevem um conjunto de recursos de confirmação em oposição a um único recurso.

Características Definição	Nº de refatorações Nº de
refatorações de código-fonte observadas no commit pai. # de locais do lado direito # de locais tocados por refatorações do commit filho.	# de locais do lado esquerdo # de locais tocados por refatorações
LOC lado esquerdo	Linhas de código tocadas no commit pai por refatorações.
LOC lado direito	Linhas de código tocadas no commit filho por refatorações. # of files # de arquivos tocados por refatorações.
Número médio de arquivos	Número médio de arquivos tocados em refatorações em um commit. # de tipos de refatoração exclusivos # de elementos de código exclusivos de refatoração exclusivos aplicados em um commit.
	# de elementos de código exclusivos identificados por cada refatoração.
# de refatorações anteriores	# de refatorações que cada arquivo teve anteriormente com a média obtida de todos os arquivos. # de dias desde a última refatoração para cada arquivo, com a média
Idade de refatoração	tomado como seu valor.
Experiência de refatoração do desenvolvedor	# de refatorações aplicadas pelo desenvolvedor anteriormente a este commit.
Experiência de confirmação de refatoração do desenvolvedor	# de commits de refatoração feitos pelo desenvolvedor anteriormente a este commit.
Contagem de tipos de refatoração*	Um recurso para cada tipo de refatoração no commit, junto com seu número de ocorrências como seu valor.
Tipo de elemento de código contar*	Um recurso para cada tipo de elemento de código no commit, junto com seu número de ocorrências como seu valor.

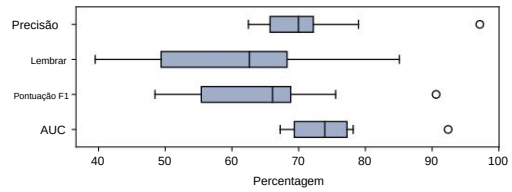

Figura 3: Boxplots dos valores resultantes das métricas utilizadas para avaliar os classificadores Random Forest construídos.

número de commits de refatoração (consulte a Tabela 1). Além disso, como queríamos estudar a possibilidade de determinar commits de refatoração coocorrentes, precisávamos estudar projetos com um número suficiente de commits de refatoração coocorrentes. Assim, para construir nossos classificadores, selecionamos dez projetos de nosso conjunto de dados que mantêm um conjunto de dados de boa qualidade em termos de número de commits (ou seja, código-fonte e commits de refatoração coocorrentes). A lista de projetos pode ser encontrada em nosso pacote de replicação [12]. Além disso, para lidar com o problema de desequilíbrio em nosso estudo, aplicamos a técnica de sobreamostragem minoritária sintética (SMOTE), que é uma estratégia de sobreamostragem e pode efetivamente aumentar o desempenho de um classificador em nosso caso [5]. No entanto, vale a pena notar que aplicamos a técnica de amostragem apenas ao conjunto de dados de treinamento e não a aplicamos ao conjunto de dados de teste.

Uma vez que nosso conjunto de dados foi escolhido, começamos a aplicar vários classificadores de aprendizado de máquina. Para comparar os classificadores entre si e determinar o desempenho de cada classificador, computamos as métricas de avaliação bem conhecidas, que são Recall, Precision, F1-score e Area Under the ROC Curve (AUC). Neste estudo, aplicamos vários classificadores que incluem Logistic Regression , Support Vector Machine e Random Forest, em suas configurações padrão por projeto, usando validação cruzada de 10 vezes [6]. Descobrimos que o classificador Random Forest obteve o melhor desempenho.

Além disso, neste RQ, examinamos a importância de cada uma de nossas características. Para isso, usamos a importância do recurso do sci-kit com base na diminuição média da impureza, uma vez que nosso conjunto de dados não possui nenhum recurso categórico e, portanto, não possui nenhum recurso de alta car dinalidade, mas pode ter recursos correlacionados, o que tem um impacto negativo na importância baseada em permutação de características alternativas [9]. Em seguida, usando essa métrica, avaliamos a importância de cada recurso para todos os classificadores Random Forest criados para cada conjunto de dados do projeto. Por fim, agregamos a importância do recurso para demonstrar quais recursos desempenham os papéis mais importantes na previsão de confirmações de refatoração co-ocorrentes.

Resultado: A Figura 3 mostra os resultados dos classificadores Random Forest construídos. A Figura apresenta a distribuição dos valores de precisão, recall, F1-score e AUC para todos os dez projetos examinados.

Como podemos ver, os classificadores Random Forest construídos atingem um F1-score médio de 0,65 (mediana = 0,66) e uma AUC média de 0,75 (mediana = 0,74). No geral, embora estes possam parecer números de desempenho modestos, eles são bastante significativos, dada a natureza desequilibrada do conjunto de dados estudado (ou seja, apenas uma pequena parte dos commits são commits de refatoração concomitantes). Interessante, existem dois projetos que alcançaram um desempenho extremamente alto, Kafka e Phoenix , que atingiram AUC de 0,78 (F1-score = 0,76) e AUC de 0,92 (F1-score = 0,91), respectivamente.

Além disso, examinamos o recurso mais importante para determinar quando o código de teste deve ser refatorado com refatorações de código-fonte . A Figura 4 mostra a distribuição dos dez mais importantes

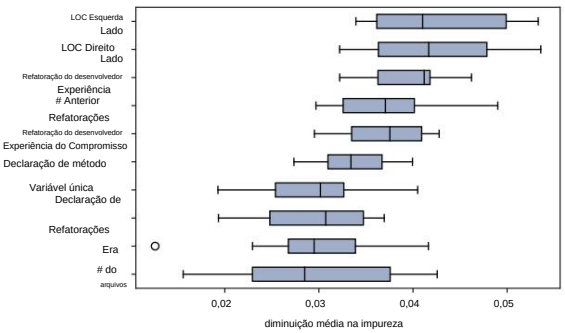

Figura 4: Boxplot dos principais recursos, escolhidos por importância nos classificadores Random Forest construídos.

recursos usados nos classificadores Random Forest construídos. Dos 10 principais recursos, eles podem ser agrupados em recursos de tamanho de confirmação (LOC lado esquerdo, LOC lado direito, nº de arquivos, nº de refatorações), recursos de experiência (experiência de refatoração dev, experiência de confirmação de refatoração dev), recursos de histórico (# Refatorações anteriores, idade) e elementos de código (declarações de método e variável única). Cada grupo tem seu próprio significado, mas isso pode demonstrar que cada um desses grupos de recursos tem uma influência sobre se o código de teste deve ser refatorado quando o código-fonte é refatorado.

Nossos classificadores podem determinar efetivamente a coocorrência de commits de refatoração de teste e código-fonte com uma AUC mediana de 0,74. Além disso, nossos resultados mostraram que o tamanho da refatoração e a experiência de refatoração do desenvolvedor são os recursos mais importantes.

4 AMEAÇAS À VALIDADE

Existem algumas limitações significativas em nosso trabalho que precisam ser consideradas ao interpretar nossos achados. A primeira limitação é que contamos com a precisão da ferramenta RefactoringMiner para identificar, classificar e fornecer recursos significativos para nossos classificadores. No entanto, com uma precisão esperada de 99,6% e um recall de 94%, esperamos que o impacto seja bastante insignificante [19]. Outra limitação do nosso estudo pode ser o nosso conjunto de dados. Nosso trabalho se concentra apenas em 77 projetos, todos escritos na mesma linguagem de programação, Java, e da organização Apache. Isso significa que nossas descobertas podem não extrapolar bem para outras linguagens de programação ou outras organizações.

5. CONCLUSÕES

Neste trabalho, estudamos a co-ocorrência de código-fonte e oportunidades de refatoração de código de teste. Começamos analisando 77 projetos para identificar a prevalência de refatorações coocorrentes entre o código-fonte e o código de teste. Nossas descobertas mostraram que, em média, 17,9% dos commits de refatoração são commits de refatoração coocorrentes. Também descobrimos que Change Variable Type e Move Class são as refatorações mais comuns aplicadas ao código de teste em commits de refatoração coocorrentes. Além disso, criamos classificadores preditivos para determinar quando a refatoração do código de teste pode ocorrer em conjunto com a refatoração do código-fonte para dez projetos. Nossas descobertas mostraram que os classificadores construídos podem prever com precisão quando o teste e a refatoração do código-fonte co-ocorrem com valores de AUC entre 0,67-0,92. Por fim, nossa análise também mostrou que os recursos mais importantes para nossos classificadores preditivos estão relacionados ao tamanho da refatoração e à experiência de refatoração do desenvolvedor.

REFERÊNCIAS

[1] Wajdi Aljedaani, Anthony Peruma, Ahmed Aljohani, Mazen Alotaibi, Mo hamed Wiem Mkaouer, Ali Ouni, Christian D Newman, Abdullatif Ghallab e Stephanie Ludi. 2021. Testar ferramentas de detecção de cheiro: Um estudo de mapeamento sistemático. Avaliação e Avaliação em Engenharia de Software (2021), 170–180.

[2] Eman Abdullah AlOmar, Mohamed Wiem Mkaouer e Ali Ouni. 2021. Rumo à classificação automática de refatoração auto-afirmada. Journal of Systems and Software 171 (2021), 110821.

[3] Eman Abdullah AlOmar, Anthony Peruma, Mohamed Wiem Mkaouer, Christian Newman, Ali Ouni e Marouane Kessentini. 2021. Como refatoramos e como documentamos? Sobre o uso de algoritmos de aprendizado de máquina supervisionados para classificar a documentação de refatoração. Sistemas especialistas com aplicativos 167 (2021), 114176. https://doi.org/10.1016/j.eswa.2020.114176

[4] Gabriele Bavota, Abdallah Qusef, Rocco Oliveto, Andrea Lucia e Dave Binkley. 2015. Os cheiros de teste são realmente prejudiciais? Um Estudo Empírico. Softw. Eng. 20, 4 (agosto de 2015), 1052-1094. https://doi.org/10.1007/s10664-014-9313-0 [5] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall e W Philip Kegelmeyer. 2002. SMOTE: técnica de sobreamostragem minoritária sintética. Jornal de pesquisa de inteligência artificial 16 (2002), 321-357.

[6] Bradley Efron. 1983. Estimando a taxa de erro de uma regra de previsão: melhoria na validação cruzada. Jornal da associação estatística americana 78, 382 (1983), 316-331.

[7] Giovanni Grano, Cristian De Iaco, Fabio Palomba e Harald C. Gall. 2020. Pizza versus Pinsa: Sobre a Percepção e Mensurabilidade da Qualidade do Código de Teste Unitário. Em 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). 336-347. https://doi.org/10.1109/ICSME46990.2020.00040 [8] Eduardo Martins Guerra e Clovis Torres Fernandes. 2007. Refatoração de código de teste com segurança. Na Conferência Internacional sobre Avanços em Engenharia de Software (ICSEA 2007). IEEE, 44-44.

[9] Giles Hooker, Lucas Mentch e Siyu Zhou. 2021. Permutação irrestrita força a extrapolação: a importância da variável requer pelo menos mais um modelo, ou não há importância da variável livre. Estado. Computar. 31, 6 (2021), 82. https: //doi.org/10.1007/s11222-021-10057-z [10] Dong Jae Kim, Tse-Hsun Peter Chen e Jinqiu Yang. 2021. A vida secreta dos cheiros de teste - um estudo empírico sobre evolução e manutenção do cheiro de teste. Engenharia de Software Empírica 26, 5 (2021), 1–47.

[11] Tom Mens e Tom Tourwé. 2004. Uma pesquisa de refatoração de software. IEEE Transactions on software engineering 30, 2 (2004), 126–139.

[12] Nicholas Alexandre Nagy e Rabe Abdalkareem. 2022. Refatoração Co ocorrência Scripts de Replicação + Dados. https://doi.org/10.5281/zenodo.5979790 [13] Anthony Peruma, Khalid Almalki, Christian D. Newman, Mohamed Wiem Mkaouer, Ali Ouni e Fabio Palomba. 2019. Sobre a distribuição de cheiros de teste em aplicativos Android de código aberto: um estudo exploratório. In Proceedings of the 29th Annual International Conference on Computer Science and Software Engineering (Toronto, Ontário, Canadá) (CASCON '19). IBM Corp., EUA, 193–202.

[14] Danilo Silva, João Paulo da Silva, Gustavo Santos, Ricardo Terra e Marco Túlio Valente. 2021. RefDiff 2.0: Uma ferramenta de detecção de refatoração em vários idiomas. Transações IEEE em Engenharia de Software 47, 12 (2021), 2786–2802. https://doi.org/10.1109/TSE.2020.2968072

[15] Danilo Silva, Nikolaos Tsantalis e Marco Túlio Valente. 2016. Por que refatoramos ? Confissões de contribuidores do GitHub. In Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (Seattle, WA, EUA) (FSE 2016). Association for Computing Machinery, Nova York, NY, EUA, 858-870. https://doi.org/10.1145/2950290.2950305 [16] Alexander Trautsch, Fabian Trautsch e Steffen Herbold. 2021. Desafio de Mineração MSR: Os Dados do Repositório SmartSHARK. In Proceedings of the International Conference on Mining Software Repositories (MSR 2022).

[17] Alexander Trautsch, Fabian Trautsch e Steffen Herbold. 2021. SmartSHARK 2.1 Completo. https://doi.org/10.25625/7OZ1SP

[18] Nikolaos Tsantalis. 2021. tsantalis/RefactoringMiner em 2.2.0. https://github.com/ tsantalis/RefactoringMiner/tree/2.2.0. (acessado em 05/02/2022).

[19] Nikolaos Tsantalis, Ameya Ketkar e Danny Dig. 2020. RefactoringMiner 2.0. Transações IEEE em Engenharia de Software (2020), 21 páginas. https://doi.org/10.1109/TSE.2020.3007722

[20] Michele Tufano, Fabio Palomba, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrea De Lucia e Denys Poshyvanyk. 2016. Uma investigação empírica sobre a natureza dos cheiros de teste. In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (Cingapura, Cingapura) (ASE 2016). Association for Computing Machinery, Nova York, NY, EUA, 4–15. https://doi.org/10.1145/2970276.2970340

[21] Danny van Bruggen. 2008. JavaParser - Home. https://javaparser.org/. (acessado em 05/02/2022).

[22] Arie Van Deursen, Leon Moonen, Alex Van Den Bergh e Gerard Kok. 2001. Refatorando o código de teste. In Anais da 2ª conferência internacional sobre programação extrema e processos flexíveis em engenharia de software (XP2001). Citéer, 92-95.