Desafio de Mineração MSR: O SmartSHARK

Dados de Mineração do Repositório

1rua Alexandre Trautsch
Instituto de Ciência da Computação
Universidade de Göttingen
Göttingen, Alemanha
alexander.trautsch@cs.uni-goettingen.de

2º lugar Fabian Trautsch
ftrautsch@googlemail.com

3 dia Steffen Herbold
Instituto de Engenharia de Software e Sistemas TU Clausthal Clausthal-Zellerfeld,
Alemanha steffen.herbold@tu-clausthal.de

Resumo—Os dados de mineração do repositório SmartSHARK são uma coleção de informações ricas e detalhadas sobre a evolução dos projetos de software. Os dados são únicos em sua diversidade e contêm informações detalhadas sobre cada alteração, dados de rastreamento de problemas, dados de integração contínua, bem como dados de pull request e revisão de código. Além disso, os dados não contêm apenas dados brutos extraídos de repositórios, mas também anotações em forma de rótulos determinados por uma combinação de análise manual e heurística, bem como links entre as diferentes partes do conjunto de dados. O conjunto de dados SmartSHARK fornece uma rica fonte de dados que nos permite explorar questões de pesquisa que exigem dados de diferentes fontes e/ou dados longitudinais ao longo do tempo.

Termos de indexação – mineração de repositório, controle de versão, rastreamento de problemas, lista de discussão, integração contínua, revisão de código, métricas de software

I. INTRODUÇÃO

Nos últimos anos, investimos muito esforço para criar um conjunto de dados versátil sobre a evolução de projetos de software que combina dados de diferentes fontes com base em nossa plataforma SmartSHARK para mineração de repositório de software replicável e reproduzível [1], [2]. O núcleo dessa abordagem foi combinar todos os dados que geramos para diferentes publicações em um único banco de dados, que cresce a cada publicação. Isso não significa apenas que adicionamos mais projetos ao longo do tempo, mas também que a quantidade de informações para os projetos que já estão no banco de dados aumenta. Até agora, nosso banco de dados contém os seguintes dados:

- Dados coletados do Git, por exemplo, as mensagens de commit, autores, datas, bem como os blocos alterados. O clone do repositório Git no momento da coleta também é armazenado para permitir uma análise posterior do código-fonte.
- Dados sobre o código fonte **para cada commit** focado em Java, por exemplo, métricas de software (tamanho, complexidade, documentação, clones de código), avisos de análise estática do PMD1 e o número de nós de cada tipo no AST de um arquivo.
- Dados sobre alterações de código, ou seja, a detecção de tipos de alteração com ChangeDistiller [3], bem como refatorações com RefDiff [4] e RefactoringMiner [5].
- Dados coletados do Jira, ou seja, os problemas, comentários e alterações feitas nos problemas.
- Dados coletados do GitHub, ou seja, problemas, pull requests e revisões de código como parte de pull requests.
- Dados coletados de listas de discussão, ou seja, todos os e-mails do listas de discussão de desenvolvedores e usuários.
- Links de commits e issues, bem como links entre commits e pull requests.
- Links validados manualmente entre commits e bugs, bem como o tipo de problemas rotulados como bug para 38 projetos [6].
- Etiquetas de linha validadas manualmente que marcam quais mudanças contribuíram para uma correção de bug para 23 projetos, bem como dados parciais para cinco projetos adicionais [7].
- Anotações para commits e alterações, ou seja, alterações de correção de bugs, incluindo suas prováveis alterações indutoras, se as alterações modificaram Javadocs ou comentários inline, se TODOs foram adicionados ou removidos, se o código de teste foi alterado ou se fomos capazes de detectar refatorações.
- Logs de compilação do Travis CI e informações de status de compilação para todos os projetos que usam o Travis CI.

As identidades dos desenvolvedores são gerenciadas em uma coleção separada que não é compartilhada publicamente, a menos que especificamente solicitada com uma descrição da finalidade. Portanto, os desenvolvedores são identificados apenas pelo identificador de objeto (aleatório) no banco de dados.

II. DESCRIÇÃO DOS DADOS

Esta publicação descreve a versão 2.1 dos dados, que está disponível publicamente.2 Versões mais antigas estão disponíveis em nossa página inicial, onde também publicaremos versões futuras.3 Uma descrição sobre como configurar os dados para uso local, bem como um exemplo para acessar os dados com Python está disponível online.4

A seguir, descrevemos as fontes de dados, as ferramentas que usamos para a coleta de dados, o tamanho e o formato dos dados, o esquema de nosso banco de dados, a estratégia de amostragem que usamos e a lista dos projetos para os quais os dados estão disponíveis.

Este trabalho foi parcialmente financiado pelo DFG Grant 402774445. 1https://pmd.github.io/

2Dados completos: http://141.5.100.155/smartshark_2_1.agz
Versão pequena sem estados de entidade de código, estados de grupo de código e instâncias de clone: http://141.5.100.155/smartshark_2_1.agz Verifique https://smartshark.github.io/dbreleases/ para espelhos ou versões mais recentes.

DOIs seguem com publicação oficial.

3https://smartshark.github.io/dbreleases/

4https://smartshark.github.io/fordevs/

A. Fontes de dados

Os dados brutos foram coletados de quatro fontes diferentes.

- Os dados de controle de versão são coletados diretamente de um clone do repositório Git. Os repositórios são recuperados do GitHub.5
- Os dados de rastreamento de problemas são coletados do Apache Jira6 e GitHub.
- Os dados de solicitação pull são coletados do GitHub. • Os dados de integração contínua são coletados do Travis CI.7 Todos os dados estão disponíveis publicamente, mas os fornecedores de ferramentas podem exigir o registro para extrair os dados.

B. Ferramentas de Coleta de Dados

A Figura 1 mostra as ferramentas de coleta de dados que utilizamos. Todas as ferramentas estão disponíveis no GitHub.8 O vcsSHARK baixa um clone do repositório Git e coleta metadados sobre commits. CoastSHARK, mecoSHARK, changeSHARK, refSHARK e rminerSHARK usam o clone do repositório para coletar métricas de software e detectar refatorações. O memeSHARK remove métricas de software duplicadas e reduz o volume de dados. O travisSHARK coleta dados do Travis e os vincula aos commits. O prSHARK coleta pull requests incluindo revisões e comentários do GitHub e os vincula a commits. O mailingSHARK coleta E-Mails de listas de discussão. O issueSHARK coleta dados de rastreamento de problemas do Jira e do GitHub. O linkSHARK estabelece links entre os problemas e os commits. O labelSHARK usa esses links, as diferenças textuais das alterações e as alterações nas métricas de código para calcular os rótulos dos commits. Esses rótulos são usados pelo inducingSHARK para encontrar as prováveis mudanças que estão induzindo para os rótulos, por exemplo, para bugs.

O visualSHARK é usado para validação manual de dados, por exemplo, de links entre commits e problemas, tipos de problemas e alterações que contribuem para correções de bugs. Essas informações são usadas pelo labelSHARK e pelo inducingSHARK para melhorar os dados que dependem dessas informações, por exemplo, rótulos de bug para commits. Para completar, também mencionamos o identitySHARK, que pode ser usado para mesclar várias identidades da mesma pessoa em nossos dados (por exemplo, nome de usuário diferente, mesmo e-mail). No entanto, esses dados não fazem parte de nosso conjunto de dados públicos e só serão disponibilizados mediante solicitação se o uso desejado for claramente especificado e não levantar nenhuma preocupação ética ou relacionada à privacidade de dados.

C. Tamanho e Formato

O conjunto de dados contém atualmente 77 projetos, as validações manuais estão disponíveis para um subconjunto de 38 projetos. No geral, esses projetos têm 366.322 commits, 163.057 problemas, 47.303 pull requests e 2.987.591 emails. Todos os dados são armazenados em um MongoDB. O tamanho do MongoDB completo é de 1,2 Terabyte. Esse tamanho cai drasticamente para cerca de 40 Gigabytes, se omitirmos as coleções com dados de clone de código e software

5https://www.github.com/
6https://issues.apache.org/jira/
7https://www.travis-ci.com/ 8https://github.com/smartshark/

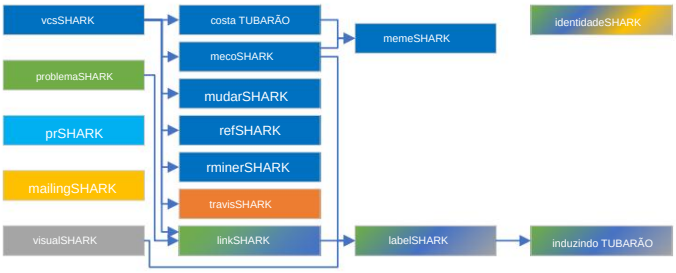

Fig. 1. Visão geral das ferramentas de coleta de dados. As setas indicam dependências entre ferramentas. As cores indicam que diferentes fontes de dados são usadas (azul: repositório Git; verde: problemas do Jira e do GitHub; azul claro: solicitações de pull do GitHub; amarelo: arquivo da lista de discussão; laranja: Travis CI; cinza: validação manual). Uma mistura de cores significa que são necessários dados de diferentes fontes, conforme indicado pelas dependências e cores.

Métricas. Os dados ainda estão crescendo e projetos adicionais também serão disponibilizados por meio de lançamentos subsequentes do conjunto de dados.

Drivers para MongoDB estão disponíveis para muitas linguagens de programação.9 Além disso, fornecemos ping de mapa relacional de objeto (ORM) para Python e Java.

D. Visão geral do esquema de banco de dados

Atualmente, temos dados de quatro tipos de repositórios: sistemas de controle de versão, sistemas de rastreamento de problemas, sistemas de pull request e listas de discussão. A Figura 2 fornece uma visão geral do nosso esquema de banco de dados. Uma documentação completa está disponível online.10 Cada projeto tem uma entrada com o nome e id. Os repositórios de software são atribuídos aos projetos por seu id.

O sistema mais simples são as listas de discussão. Os emails das listas de endereçamento são armazenados na coleção de mensagens. Os dados de rastreamento do problema são armazenados em três coleções: o problema armazena os dados sobre o problema em si, por exemplo, o título, a descrição e o tipo; issue_comment armazena a discussão do problema; e event armazena qualquer atualização feita no problema, por exemplo, alterações do tipo, status ou descrição. Dessa forma, o ciclo de vida completo, incluindo todas as atualizações, fica disponível no banco de dados.

As solicitações de pull são organizadas de forma semelhante, mas requerem sete coleções, devido à relação direta com o código-fonte e as revisões de código associadas: pull_request armazena os metadados da solicitação de pull, por exemplo, o título, a descrição e as ramificações associadas; pull_request_comment armazena a discussão do pull request; pull_request_event armazena qualquer atualização feita no pull request; pull_request_file e pull_request_commit armazenam referências a arquivos e commits em pull requests; e pull_request_review e pull_request_review_comment armazenam as informações sobre revisões de código.

Os dados do sistema de controle de versão são relativamente complexos, devido à diversidade dos dados armazenados. A coleção principal é o commit, que contém os metadados gerais sobre os commits, por exemplo, o autor, o committer, o hash de revisão, a mensagem do commit e o carimbo de data/hora. Além disso, commit também contém

9https://docs.mongodb.com/drivers/
10https://smartshark2.informatik.uni-goettingen.de/documentation/

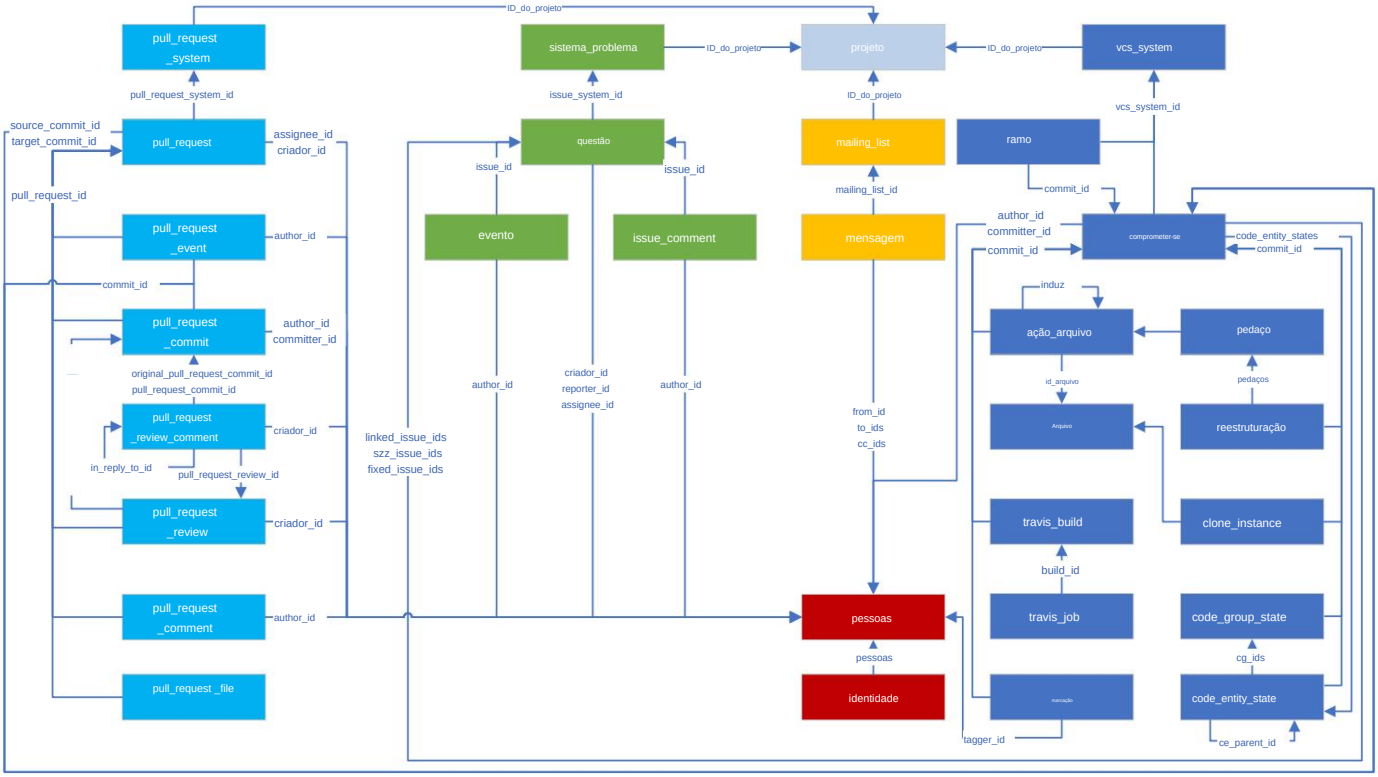

Fig. 2. Visão geral do esquema do banco de dados e os relacionamentos entre as coleções.

dados computados, por exemplo, rótulos como correção de bugs ou links para problemas. O file_action agrupa todas as alterações feitas em um arquivo em um commit, hunk contém as alterações reais, incluindo os diffs.

As informações gerais sobre o histórico são completadas pelas coleções de ramificações e tags. O code_group_state e code_entity_state contêm os resultados da análise estática que executamos no repositório em cada commit. Grupos de código são, por exemplo, pacotes, entidades de código, são, por exemplo, arquivos, classes e métodos. Removemos entidades de código duplicadas, por exemplo, arquivos em que as medidas não foram alteradas de um commit para o próximo.

Desta forma, podemos reduzir o volume de dados em mais de 11 Terabyte. Para ainda permitir a identificação dos estados da entidade de código no momento de uma confirmação, a coleção de confirmação contém uma lista das entidades de código corretas. Embora as entidades de código também contenham um link para o commit para o qual foram medidas, esse link deve ser evitado, porque os usuários podem inadvertidamente supor que poderiam encontrar todas as entidades de código para um commit específico dessa maneira, o que não é o caso. A coleção clone_instance armazena dados sobre clones de código. As refatorações detectadas automaticamente são armazenadas na coleção de refatoração. A coleção travis_build contém as informações gerais sobre a compilação, por exemplo, carimbos de data e hora e o status da compilação, e a coleção travis_job contém os logs dos trabalhos de compilação individuais.

A coleção de pessoas não está associada a nenhuma fonte de dados específica. Em vez disso, mapeamos todos os metadados que contêm contas, nomes ou endereços de e-mail para essa coleção e armazenamos o nome, o endereço de e-mail e o nome de usuário. A identidade coleção contém uma lista de pessoas, que muito provavelmente pertencem à mesma identidade. Usamos nosso próprio algoritmo de mesclagem de identidade, que está disponível online.11

E. Estratégia de Amostragem e Representatividade

Os dados contêm apenas projetos da Apache Software Foundation que têm Java como linguagem principal. Todos os projetos têm entre 1.000 e 20.000 commits, ou seja, os dados não contêm projetos muito pequenos ou muito grandes. O motivo da exclusão de projetos muito grandes é o volume de dados e o tempo de processamento para a análise estática de cada commit.

Embora a amostra não seja sorteada aleatoriamente, ela deve ser representativa para projetos Java bem mantidos que possuem altos padrões para seus processos de desenvolvimento, especialmente no que diz respeito ao rastreamento de problemas. Além disso, os projetos abrangem diferentes tipos de aplicativos, incluindo sistemas de compilação (ant-ivy), aplicativos da Web (por exemplo, jspwiki), estruturas de banco de dados (por exemplo, calcite), ferramentas de processamento de big data (por exemplo, kylin) e bibliotecas de uso geral (comuns).

F. Lista de Projetos

Coletamos os dados que descrevemos acima para os projetos a seguir. Dados validados manualmente estão disponíveis para os projetos em itálico. Os dados do Travis CI estão disponíveis para todos os projetos em negrito.

11https://github.com/smartshark/identitySHARK (uma publicação científica sobre nosso algoritmo ainda não está disponível)

activemq, ant-ivy, archiva, **bigtop, calcite, cayenne, commons-bcel, commons-beanutils, commons-codec, commons-collections, commons-compress, commons configuration, commons-dbcp, commons-digester, commons imaging, commons- io, commons-jcs, commons-jexl, commons-lang, commons-math, commons-net, commons rdf,** commons-scxml, **commons-validator, commons-vfs, curador,** cxf-fediz, deltaspike, derby, directory-fortress -core, directory-kerby, directory-studio, eagle, falcon, **fineract, flume, freemarker,** giraph, gora, helix, **httpcomponents client, httpcomponents-core,** jackrabbit, jena, jspwiki, kafka, **knox, kylin,** lens, **mahout, manifoldcf,** maven, mina sshd, **nifi,** nutch, oozie, openjpa, openwebbeans, **opennlp, parquet-mr, pdfbox,** phoenix, pig, **ranger,** roller, **samza,** santuario-java, **storm, streams, struts,** systemml, **tez,** tika , wss4j, xerces2-j, **xmlgraphics-batik, zeppelin,** zookeeper.

As alterações indutoras de bugs não estão disponíveis para maven, porque o projeto usa vários rastreadores de problemas, que atualmente não podemos manipular.[12]

III. EXEMPLOS DE USO

Os dados do SmartSHARK são versáteis e permitem diferentes tipos de pesquisas. No passado, nos concentramos principalmente na análise de bugs, bem como na análise longitudinal de tendências dentro do histórico de desenvolvimento. Abaixo, listamos alguns exemplos de artigos que usaram (um subconjunto) desse conjunto de dados. Observe que alguns dos artigos abaixo ainda estão em análise e ainda não foram publicados em suas versões finais.

- Avaliamos a qualidade dos dados de previsão de defeitos com foco em SZZ e validação manual [6]. O manuscrito descreve como validamos manualmente os links entre commits e problemas, bem como os tipos de problemas e como usamos o SmartSHARK para criar dados de previsão de defeitos no nível da versão.
- Avaliamos as tendências dos avisos de análise estática do PMD e o uso de regras personalizadas para o PMD, bem como o impacto na densidade de defeitos [8].
- Avaliamos o impacto de métricas estáticas de código-fonte e avisos de análise estática do PMD na previsão de defeitos just-in-time [9].
- Usamos os dados de tipo de problema validados manualmente para treinar e avaliar modelos de previsão de tipo de problema [10].
- Fornecemos os dados para a modelagem do comportamento do desenvolvedor através de Hidden Markov Models (HMMs) [11].
- Analisamos o emaranhamento nos commits de correção de bugs, bem como a capacidade dos pesquisadores de identificar manualmente o emaranhamento [12].
- Realizamos uma avaliação inicial de um modelo de custo para predição de defeitos [13].

4. POSSÍVEIS PERGUNTAS DE PESQUISA

A força de nossos dados é a capacidade de raciocinar sobre dados de diferentes fontes de informação. Perguntas sobre as diferenças entre as discussões nas listas de discussão e nos rastreadores de problemas podem ser respondidas sem extrair dados de várias fontes. Além disso, os resultados da análise estática e a rotulagem das alterações permitem pesquisar a relação, por exemplo, entre refatorações e dívida técnica auto-admitida ou correções de bugs. A disponibilidade de dados validados manualmente permite avaliar a validade das heurísticas, bem como o desenvolvimento de melhorias das heurísticas, por exemplo, para rotulagem de correções de bugs. Além disso, embora já tenhamos estabelecido muitos links entre as fontes de dados, há mais oportunidades que podem ser consideradas, por exemplo, os links entre a lista de discussão e os commits, ou a lista de discussão e os problemas relatados. Da mesma forma, os links entre pull requests e bugs podem ser explorados, por exemplo, para entender por que os bugs pós-lançamento não foram detectados durante a revisão do código.

V. LIMITAÇÕES

A maior limitação dos dados do SmartSHARK é o número de projetos para os quais os dados estão disponíveis. A razão para isso é o grande esforço computacional necessário para a análise estática do código Java de cada commit. Isso não apenas limita a validade externa devido ao tamanho da amostra, mas também devido ao foco em Java como linguagem de programação. No futuro, planejamos superar essa limitação estendendo o banco de dados com um grande conjunto de projetos, para os quais omitimos a análise estática e, assim, podemos aumentar o número de projetos. Embora isso não dê suporte às mesmas perguntas de pesquisa, há muitas perguntas interessantes que podem ser respondidas sem uma análise estática do código-fonte para cada confirmação.

VI. CONCLUSÃO

O conjunto de dados SmartSHARK fornece uma rica fonte de dados que nos permite explorar questões de pesquisa que exigem dados de diferentes fontes e/ou dados longitudinais ao longo do tempo.
Como todos os dados são armazenados em um único banco de dados, os resultados são fáceis de reproduzir. Os dados ainda estão crescendo e as versões futuras estenderão ainda mais os dados com mais projetos e dados adicionais fontes.

REFERÊNCIAS

[1] F. Trautsch, S. Herbold, P. Makedonski e J. Grabowski, “Abordando problemas com replicabilidade e validade de estudos de mineração de repositórios por meio de uma plataforma de dados inteligente”, Empirical Software Engineering, agosto de 2017.

[2] A. Trautsch, F. Trautsch, S. Herbold, B. Ledel e J. Grabowski, “O ecossistema smartshark para mineração de repositório de software”, em Proc. do Internacional 2020. Conf. Softw. Eng. - Faixa de Demonstrações, 2020.

[3] B. Fluri, M. Wursch, M. Pinzger e H. Gall, "Change destilação: Tree differencing for fine-grained source code change extract", IEEE Transactions on Software Engineering, vol. 33, nº. 11, pp. 725-743, 2007.

[4] D. Silva e MT Valente, “Refdiff: Detecting refactorings in version histories”, em 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), maio de 2017, pp. 269–279.

[5] N. Tsantalis, M. Mansouri, LM Eshkevari, D. Mazinanian e D. Dig, “Detecção de refatoração precisa e eficiente no histórico de commits”, em Proceedings of the 40th International Conference on Software Engineering, ser. ICSE '18. Nova York, NY, EUA: ACM, 2018, pp. 483–494. [Online]. Disponível: http://doi.acm.org/10.1145/3180155.3180206 [6] S. Herbold, A. Trautsch e F. Trautsch, “Questões com szz: Uma avaliação empírica do estado da prática da coleta de dados de previsão de defeitos, "2019.

12A versão 2.2 deste conjunto de dados com mais projetos e esses dados indutores ausentes está prevista para dezembro de 2022. Esta pré-impressão será atualizada com a lista final de projetos disponíveis para o desafio.

[7] S. Herbold, A. Trautsch, B. Ledel, A. Aghamohammadi, TA Ghaleb, KK Chahal, T. Bossenmaier, B. Nagaria, P. Makedonski, MN Ahmadabadi, K. Szabados, H. Spieker, M. Madeja, N. Hoy, V. Lenar duzzi, S. Wang, G. Rodríguez-Perez, R. Colomo-Palacios, R. Verdecchia, P. Singh , Y. Qin, D. Chakroborti, W. Davis, V. Walunj, H. Wu, D. Marcílio, O. Alam, A. Aldaeej, I. Amit, B. Turhan, S. Eismann, A.-K. Wickert, I. Malavolta, M. Sulir, F. Fard, AZ Henley, S. Kourtzanidis, E. Tuzun, C. Treude, SM Shamasbi, I. Pashchenko, M. Wyrich, J. Davis, A. Serebrenik, E. Albrecht, EU Aktas, D. Struber e J. Erbel, “Validação manual em larga escala de confirmações de correção de bugs: uma análise granular de emaranhamento”, 2020.

[8] A. Trautsch, S. Herbold e J. Grabowski, "Um Estudo Longitudinal da Evolução do Aviso de Análise Estática e os Efeitos do PMD na Qualidade do Software em Projetos de Código Aberto Apache," Engenharia Empírica de Software, 2020. [Online]. Disponível: https: //doi.org/10.1007/s10664-020-09880-1 [9] ——, “Métricas de código-fonte estático e avisos de análise estática para previsão de defeitos just-in-time com granulação mais fina,” IEEE Transactions on Software Engineering, 2020. Softw. Manutenção e Evolução, 2020.

[10] S. Herbold, A. Trautsch e F. Trautsch, “Sobre a viabilidade da previsão automatizada do tipo de emissão”, https://arxiv.org/abs/2003.05357, 2020.

[11] V. Herbold, “Dinâmica do desenvolvedor de mineração para simulação baseada em agente da evolução do software”, Dissertação, Universidade de Goettingen, Alemanha, 2019. [Online]. Disponível: http://hdl.handle.net/21.11130/00-1735-0000-0003-C15C-C [12] S. Herbold, A. Trautsch e B. Ledel, “Large-scale manual validation of bugfixing changes ”, março de 2020. [Online]. Disponível: osf.io/acnwk [13] S. Herbold, “Sobre os custos e lucros da previsão de defeitos de software”, IEEE Transactions on Software Engineering, no. 01, pp. 1–1, dezembro de 2019.

[14] Y. Zhao, H. Leung, Y. Yang, Y. Zhou e B. Xu, “Para uma compreensão dos tipos de mudança no código de correção de bugs”, Information and Software Technology, vol. 86, pp. 37 – 53, 2017. [Online]. Disponível: http://www.sciencedirect.com/science/article/pii/S0950584917301313 [15] J. Sliwerski, T. Zimmermann e A. Zeller, “Quando as mudanças induzem correções?” in Proceedings of the 2005 International Workshop on Mining Software Repositories, ser. MSR '05. Nova York, NY, EUA: ACM, 2005, pp. 1–5. [Conectados]. Disponível: http://doi.acm.org/10.1145/1082983. 1083147

[16] D. Spadini, M. Aniche e A. Bacchelli, “PyDriller: framework Python para mineração de repositórios de software”, em Anais da 26ª Reunião Conjunta da ACM de 2018 na Conferência Europeia de Engenharia de Software e Simpósio sobre os Fundamentos da Engenharia de Software - ESEC/FSE 2018. Nova York, Nova York, EUA: ACM Press, 2018, pp. 908–911. [Conectados]. Disponível: http://dl.acm.org/citation.cfm?doid= 3236024.3264598 [17] K. Herzig, S. Just e A. Zeller, “Não é um bug, é um recurso: como a classificação incorreta afeta a previsão de erros ”, em Anais da Conferência Internacional de Engenharia de Software de 2013, ser. ICSE '13. Piscataway, NJ, EUA: IEEE Press, 2013, pp. 392–401. [Conectados].

Disponível: http://dl.acm.org/citation.cfm?id=2486788.2486840 [18] RPL Buse e WR Weimer, “Learning a metric for code readability”, IEEE Trans. Softw. Eng., v. 36, nº. 4, pp. 546-558, jul. 2010. [On-line]. Disponível: http://dx.doi.org/10.1109/TSE.2009.70 [19] S. Scalabrino, M. Linares-Vasquez, R. Oliveto e D. Poshyvanyk, “Um modelo abrangente para legibilidade de código”, Journal of Software: Evolução e Processo, vol. 30, não. 6, pág. e1958, 2018.

[20] G. Gousios, “The ghtorrent dataset and tool suite,” em Proceedings of the 10th Working Conference on Mining Software Repositories, ser. MSR '13. Piscataway, NJ, EUA: IEEE Press, 2013, pp. 233–236.

[21] F. Trautsch, S. Herbold e J. Grabowski, “As definições de teste de unidade e integração ainda são válidas para projetos java modernos? um estudo empírico sobre projetos de código aberto”, Journal of Systems and Software, vol. 159, pág. 110421, 2020. [Online]. Disponível : http://www.sciencedirect.com/science/article/pii/S0164121219301955

[22] A. Trautsch, “Efeitos de ferramentas de análise estática automatizada: uma visão multidimensional na evolução da qualidade”, em 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE Companion), maio de 2019, pp. 184– 185.

[23] F. Trautsch, S. Herbold, P. Makedonski e J. Grabowski, “Abordando problemas com validade externa de estudos de mineração de repositórios por meio de uma plataforma de dados inteligente”, em Proceedings of the 13th International Conference on Mining Software Repositories, ser . MSR '16. Nova York, NY, EUA: Association for Computing Machinery, 2016, p. 97-108. [Conectados]. Disponível: https://doi.org/10.1145/2901739.2901753