Uma ferramenta para recomendação de revisores de código para apoiar a colaboração em Desenvolvimento Distribuído de Software

Vinicius Schettino

Federal University of Juiz de Fora - Brazil

Abstract

1. INTRODUÇÃO

O *code review* é considerado uma das principais técnicas para diminuição de defeitos de software [1]. Nela, o autor de uma alteração na base de código de um projeto submete tal conteúdo ao crivo de um conjunto de pares técnicos, que irão revisar sua estrutura com base em um lista de regras e convenções previamente definida. Diferentes aspectos relacionados ao autor, ao revisor e ao processo de revisão em si estão diretamente relacionados à eficiência da prática. Autores relatam a diminuição da incidência de *anti-patterns* [2] de acordo com o nível de participação dos envolvidos e cobertura do código revisado [3, 4, 5]. Reputação [6, 7] e experiência [8] do revisor também parecem impactar nos efeitos do *code review*

Intrinsecamente colaborativa, a atividade de *code review* é exercida com suporte de ferramentas computacionais específicas [9], principalmente no desenvolvimento distribuído. Dentro de workflows de trabalho descentralizados [10], a prática funciona como um *gateway* de qualidade que busca garantir

que apenas alterações aderentes aos padrões de qualidade do projeto serão incorporados à codebase principal. Esta etapa do desenvolvimento se torna uma oportunidade para disseminação de conhecimento, embate de ideias e discussão de melhores práticas entre profissionais de experiência e visões diferentes. Para tanto, percebe-se a necessidade de suporte computacional para essas atividades colaborativas.

Tais aspectos configuram o Desenvolvimento Distribuído de Software (DDS), onde equipes de desenvolvimento se encontram espalhadas por organizações e espaços geográficos distintos. Este novo ramo da Engenharia de Software vem modificando a relação entre empresas e sistemas, principalmente em relação às estratégias de negócios [11]. As próprias relações de negócios fomentam a distribuição das equipes, procurando diminuição dos custos e a incorporação de mão de obra qualificada que pode estar em qualquer lugar do planeta.

Neste contexto, porém, os os desafios à colaboração co-localizada são potencializados e as soluções tradicionais não são suficientes para fomentar esta aspecto das atividades distribuídas [12]. Casey [13] mostra que, com a distribuição geográfica dos times, diversos outros desafios, antes considerados colaterais ou resolvidos, emergem de forma a ameaçar a colaboração entre os membros da equipe: barreiras culturais, temporais e geográficas; reengenharia dos processos de desenvolvimento; resistência em compartilhar informações e conhecimento com os pares distribuídos; entre outros desafios.

Estes desafios do Desenvolvimento Distribuído de Software afetam o *code review* de duas formas distintas. Primeiro, o processo de revisão pode se tornar lento e ineficiente quando a colaboração é afetada, devido aos baixos

níveis de participação e cobertura. O mesmo vale para a disseminação do conhecimento, que fica prejudicada. Outro desafio que se consolida é a escolha do revisor adequado para aquele *patch*. Com um vasto número de opções e pouca informação disponível sobre seus aspectos técnicos e gerenciais (e.g. tempo disponível) já que não há contato co-localizado entre eles, a natureza distribuída deste tipo de desenvolvimento dificulta o processo de escolha do revisor, impactando negativamente a eficiência do processo.

Uma possível solução, visando amparar a colaboração e evitando o *overhead* da escolha do revisor, seria manter grupos bem testados e experientes exercendo as atividades de revisão. Ou ainda, fixar, dentro de cada equipe de desenvolvimento, quem são os responsáveis por revisão e pela submissão dos *patches*, evitando a diversificação das relações de trabalho.

Contudo, estudos recentes demonstram que a fixação de grupos e responsabilidades pode não ser benéfica para o processo de desenvolvimento. Scott Page [14] argumenta que a diversidade de experiências, visões e especilidades fazem com que grupos sejam mais eficientes. Já Prikladnicki et al. [15] apontam índicios de que a formação de grupos temporários em detrimento ou em conjunto com permanentes é um fator de eficiência em projetos de software:

> "Although old colleagues bring knowledge of the development process and prior norms from previous teams, new members bring fresh ideas that could promote project performance and creativity. Old colleagues might not do so and might not give new members a chance to implement their ideas."

Essa visão aponta que a formação dinâmica dos grupos de trabalho em

desenvolvimento de software potencializa a disseminação do conhecimento, um dos objetivos primários do *code review* [9].

Existem alguns trabalhos congêneres que demonstram métodos de recomendação de revisores [16, 17, 18]. Esses trabalhos foram estudados e levados em consideração para escrita do presente texto. Também foram revisadas pesquisas que apontam caracterísitcas de revisões, revisores e autores que possivelmente potencializam a colaboração [2, 19, 6]. Tais aspectos são apresentados e discutidos no capítulo 3.

As principais lacunas deixadas pelos trabalhos anteriores estão relacionadas aos objetivos e à avaliação dos métodos propostos, principalmente em DDS. Primeiramente, não há relato de método de recomendação de revisores de código com o objetivo específico de potencializar a colaboração. Por isso, métodos já propostos não utilizam métricas nem variáveis de entrada relacionadas aos aspectos de cooperação, coordenação e comunicação, como por exemplo a abordagem 3C em DDS [20].

Outro ponto observado diz respeito à avaliação dos modelos de avaliação. Os trabalhos encontrados se limitam a comparar seus resultados com métricas relacionadas à proximidade dos mesmos com a indicação manual do revisor. Ou seja, a eficência é tida de acordo com a interseção entre o recomendado automaticamente e por decisão de um especialista, geralmente um desenvolvedor. Este modelo assume que o responsável pela indicação manual tem os subsídios naturais para fazer uma boa escolha. Em DDS isso pode não ser verdade, uma vez que fatores como diferenças culturais, de horário, geográficas e de maturidade podem diminuir a compreensão do indicador e propiciar a escolha inadequada do revisor. Por isso, no contexto apresentado, outras formas

de avaliação podem ser mais apropriadas. Tais discussão são extendidas no capítulo 2.

Expostos os desafios que o Desenvolvimento Distribuído de Software impõe sobre a escolha do revisor de código, a importância da indicação do revisor adequado do ponto de vista de colaboração e a motivação da formação de grupos heterogêneos e dinâmicos, sumariza-se o intuito do presente texto. De acordo com a abordagem QGM (Goal/Question/Metric) proposta por Basili et al. [21], postula-se o objetivo do trabalho como: **Implementar** um método de recomendação de revisores **com o objetivo de** potencializar a colaboração **em relação aos aspectos** de coordenação **do ponto de vista** de revisores e autores **no contexto de** desenvolvimento distribuído de software.

A principal hipótese que norteia o andamento desta proposta, e que será revisitada e discutida nos capítulos derradeiros é:

- O método de recomendação apresentado pode potencializar a colaboração entre revisores e autores.

O uso de ferramentas computacionais para o processo de revisão de código se tornou prática comum nos últimos anos [9]. O GitHub é uma plataforma rica em repositórios de projetos de software. Muitos são de código aberto, disponíveis para mineração. Discussões sobre o *workflow* de trabalho na ferramenta em contraponto à métodos tradicionais de revisão podem ser vistas na seção 2.1. São 24 milhões de usuários, 67 milhões de projetos e 47 milhões de revisões[1], também chamadas de *pull requests* no modelo de desenvolvimento "*pull based*" [22]. Essa abordagem é explorada na seção 2.1.3.

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

Esta característica permitiu a extração e análise automatizadas das informações sobre as revisões em projetos de código aberto, através de APIs disponibilizadas para este fim. Foram extraídas métricas apontadas como relevantes para nossos objetivos pela literatura relacionada. A arquitetura que embasa a extração e análise destes dados com objetivo de recomendação é explicada no capítulo 4.

A avaliação da eficiência do método proposto apresenta particularidades em relação à trabalhos relacionados, devido ao enfoque em colaboração no contexto de DDS. O método de avaliação é devidamente discutido e aplicado no capítulo 5, incluindo a apresentação dos experimentos e a revisitação da hipótese levantada neste capítulo. Por fim, o capítulo 6 é dedicado ao fechamento do trabalho, inclusindo a sugestação de trabalhos futuros e a discussão de ameaças a validade e generalização dos resultados apresentados.

2. PRESSUPOSTOS TEÓRICOS

2.1. Code review

O *code review* é uma prática consolidada e difundida em diversas organizações, contemplando diferentes portes e segmentos de mercado. A técnica constitui da análise técnica de uma mudança a ser submetida à base principal de código (repositório-mestre) por parte de um revisor técnico, tendo como base uma lista de diretrizes e padrões a serem observados. As nuances do processo variam em cada contexto levando em consideração, por exemplo, tolerância a defeitos, modelo de desenvolvimento e os objetivos almejados.

2.1.1. Relevância

O *code review* está associada diretamente à detecção precoce de defeitos em produtos de software [23, 2], sendo reconhecida como uma das principais

técnicas com este fim [1]. Mais especificamente, é relatada maior eficiência quanto aos defeitos não-funcionais, enquanto os defeitos funcionais são menos afetados no processo [24]. Outros autores reportam a diminuição de defeitos através de estudos de caso [25, 5, 4].

2.1.2. Histórico

A atividade de revisão remonta da década de 80 [26], e desde então vem evoluindo para suportar interações mais rápidas e constantes, com uso de ferramentas computacionais e práticas ágeis. O Modern Code Review (MCR) surge em sinergia com os modelos ágeis e distribuídos de desenvolvimento, valorizando mais a comunicação e troca de experiências entre autor e revisor [9].

2.1.3. Pull Based Method

O conceito de *branches* é a base para sistemas de controle de versão descentralizados, como o Git[2] e o Mercurial[3]. Com as *branches* é possível desenvolver paralelamente, submtendo e mesclando as alterações no código em momentos oportunos. Esta característica é interessante para o DDS, uma vez que o isolamento e a atomicidade do trabalho de cada um até o momento de submissão é fundamental para a coordenação dos esforços [27].

Estas tecnologias permitiram o surgimento de um paradigma de desenvolvimento baseado em pulls, ou *pull-based method* [22]. O processo de revisão de código evolui neste novo paradigma, servindo como um *gateway* de qualidade que busca garantir que apenas alterações aderentes aos padrões de qualidade do projeto serão incorporados à codebase principal [28]. A figura 1 ilustra tal

[2]https://git-scm.com/
[3]https://www.mercurial-scm.org/

modelo de trabalho instanciado no GitHub[4], principal expoente que oferece este paradigma. Nele é representado um modelo comum em desenvolvimento OpenSource [29], onde há um *core team* responsável por revisar os *pulls* de seus colegas e da comunidade no geral. Neste modelo, a mudança chega à codebase principal somente se houver o aval de um membro do *core team*.

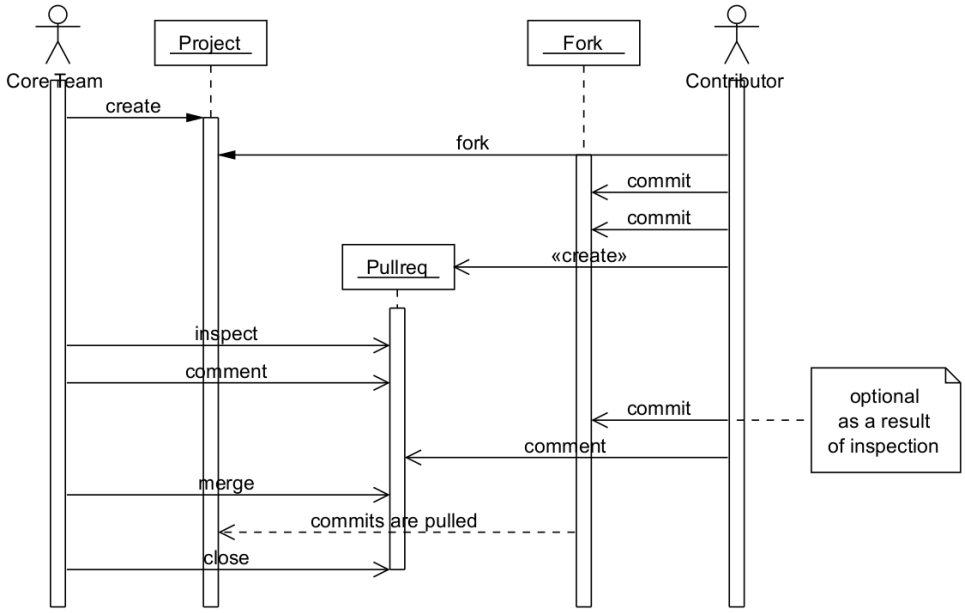

Figure 1: Pull Request Process [22]

Aquele que deseja contribuir cria para si uma cópia do projeto através de um *fork*. Esta ação cria em seu diretório de trabalho um projeto idêntico ao original, mas ao qual ele tem acesso total de submissão e modificação. Nessa cópia, ele executa as modificações desejadas, geralmente em uma *branch* dedicada para tal [10]. Ao terminar, ele solicita a integração da branch do

[4]https://github.com

fork de volta ao projeto original. Essa solitação é chamada de *pull-request*, que será analisada por um desenvolvedor com as devidas permissões. Durante esta revisão, o autor pode gerar novas modificações, geralmente atreladas aos pedidos do revisor. Ao final, a mudança é rejeitada (*closed*) ou aceita *merged*.

Os membros do core também têm suas *branches* revisadas por um processo análogo [29, 7]. A principal diferença é que não há necessidade do fork, já que eles tem as permissões necessárias para criar uma nova *branch* no projeto-alvo.

Dadas as variáveis relevantes do revisor que potencializam a colaboração discutidas na seção 3.2, este capítulo apresenta a ferramenta de recomendação para o desenvolvimento distribuído de software (DDS). Levando em consideração os modelos atuais de revisão de código exploradas na seção 2.1 são propostos os seguintes requisitos funcionais da ferramenta:

- Dada uma revisão de código, a ferramenta deve apresentar sugestões de pares técnicos para o o papel de revisor;
- a lista de revisores deve ser estar ordenada por ordem de relevância da indicação;
- os perfis dos indicados deve estar acessível (via link).

A ferramenta proposta integra com repositórios de código Git, extrair informações de histórico de desenvolvimento e proximidade do *changeset* em análise com a memória do projeto em relação ao canditado a revisor O acesso a esses metadados é efitivado através da biblioteca GitPython[5] que fornece uma interface objeto-relacional para a base de dados interna do Git. Os históricos de revisão encontram-se no Github, e são extraídos através das APIs[6] disponibilizadas para este fim. A figura 2 representa os componentes e suas comunicações.

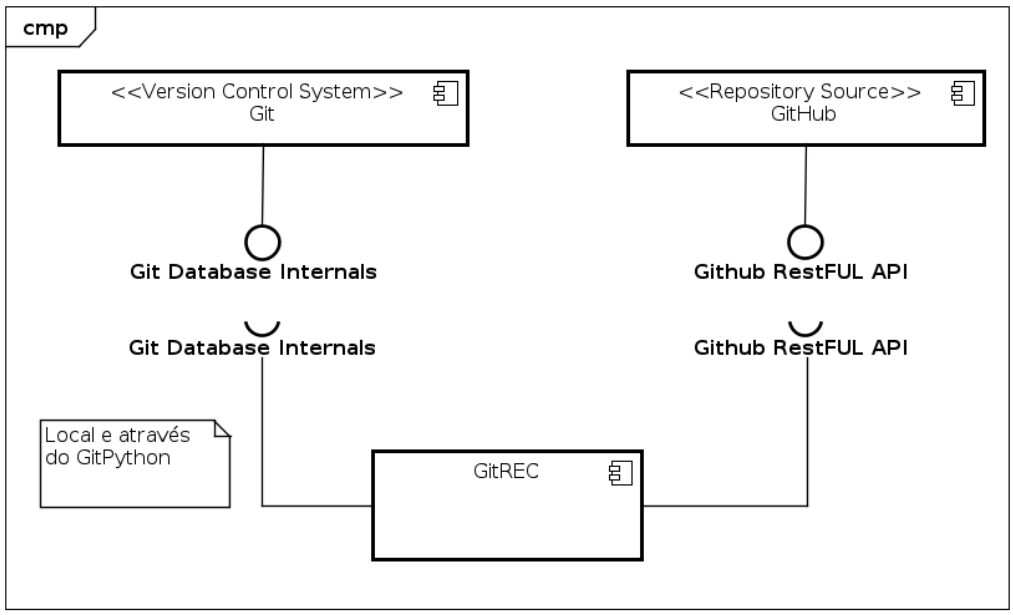

Figure 2: Diagrama de Componentes da ferramenta

O fluxo de operação da ferramenta foi projetado visando compatibilidade com o processo de revisão, especialmente no modelo *pull based* [22] intro-

[5]http://gitpython.readthedocs.io/en/stable/tutorial.html
[6]https://api.github.com

duzido na seção 2.1.3. A figura 3 representa a comunicação entre o autor do *changeset* (responsável pela escolha dos revisores), o GitHub e o utilitário de recomendação.

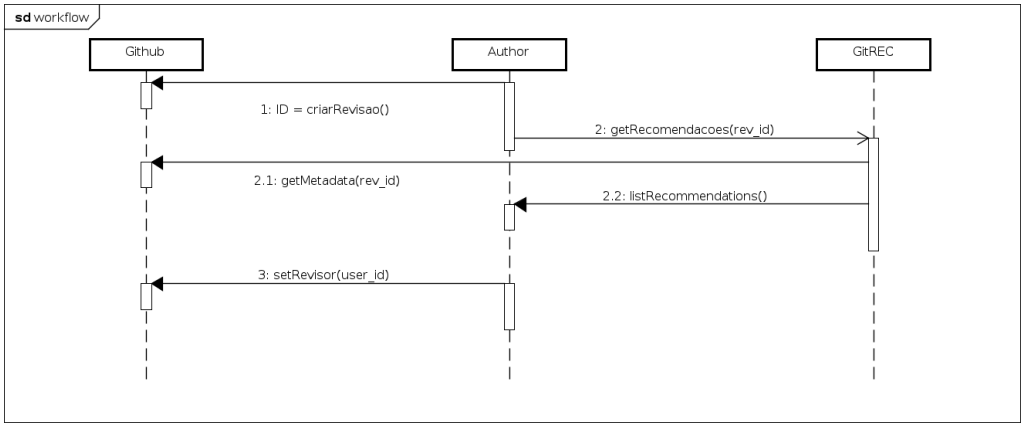

Figure 3: Diagrama de Sequência da ferramenta

O primeiro passo consiste no autor criar a revisão com os dados relevantes, como por exemplo descrição, link com issues solucionadas, dúvidas e possíveis discussões que estejam relacionadas ao código submetido. A partir daí ele já está em posse do identificador único da revisão, que será utilizada para encontrar o revisor adequado. Ao executar o utilitário, este irá acessar os metadados do Github para encontrar outras revisões de contexto semelhante e extrair informações de revisores que estiveram envolvidos nelas. A partir daí a recomendação é feita e disponibilizada para o revisor, que poderá convidar um subconjunto da lista apresentada para o papel de revisor.

Os requisitos não funcionais estão relacionados às caracterísitcas técnicas da ferramenta que suportam os requisitos funcionais, como por exemplo disponibilidade, portabilidade e manutenibilidade. Dentre tais aspectos, pode-se destacar como mais relevantes:

- Interoperabilidade
  - A ferramenta proposta deve-se comunicar o repositório de código fonte e à ferramenta de apoio a revisão de forma automática.
- Manutenibilidade
  - Existem diversas ferramentas de mercado para revisão, além de ferramentas de repositório de código fonte. Por isso a ferramenta deve ser modelada para permitir evoluções futuras que permitam a agregação de novas integrações que sejam importantes em outros domínios, como por exemplo software proprietário.

Para manter o padrão Git e possibilitar a integração futura com diversas IDEs de desenvolvimento, a ferramenta desenvolvida possui interface via linha de comando. através do utilitário `gitrec` é possível acessar todas as funcionalidades previstas. A sintaxe do comando é `gitrec [PULL REQUEST ID]` onde o utilizador fornece o identificador númerico único da revisão para a qual deseja a indicação do revisor. Outros parâmetos são:

```
--top=n | -t # Indica n potenciais revisores. O padrao e 3.
--exclude=username | -u # Exclui um usuario da busca por
   revisores.
--version | -v # Mostra a versao do utilitario.
--help | -h # Imprime esta lista de parametros.
--verbose | -vv[v] # Modo de debug, imprime todos os logs.
```

O utilitário deve ser executado dentro da pasta do projeto. São utilizadas as credenciais de clone/merge do projeto para acesso à API do Github, enquanto o histórico de evolução do código é acessado localmente através da base do Git.

5. AVALIAÇÃO DA SOLUÇÃO

5.1. Processo de Avaliação
5.2. Apresentação dos resultados
5.3. Discussão dos resultados

6. CONCLUSÃO

6.1. Ameaças
6.2. Trabalhos futuros
6.3. Considerações finais

[1] B. Boehm, V. R. Basili, Software defect reduction top 10 list, Computer 34 (1) (2001) 135–137. doi:10.1109/2.962984.

[2] C. F. Kemerer, M. C. Paulk, The impact of design and code reviews on software quality: An empirical study based on psp data, IEEE Transactions on Software Engineering 35 (4) (2009) 534–550. doi:10.1109/TSE.2009.27.

[3] A. Meneely, A. Tejeda, B. Spates, S. Trudeau, D. Neuberger, K. Whitlock, C. Ketant, K. Davis, An empirical investigation of socio-technical code review metrics and security vulnerabilities, 2014, pp. 37–44. doi:10.1145/2661685.2661687.

[4] R. Morales, S. McIntosh, F. Khomh, Do code review practices impact design quality? a case study of the qt, vtk, and itk projects, 2015 IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering (SANER) 00 (2015) 171–180.

[5] G. Bavota, B. Russo, Four eyes are better than two: On the impact of code reviews on software quality, 2015, pp. 81–90. doi:10.1109/ICSM.2015.7332454.

[6] O. Baysal, O. Kononenko, R. Holmes, M. Godfrey, The influence of non-technical factors on code review, 2013, pp. 122–131. doi:10.1109/WCRE.2013.6671287.

[7] A. Bosu, J. Carver, Impact of developer reputation on code review outcomes in oss projects: An empirical investigation, 2014. doi:10.1145/2652524.2652544.

[8] O. Kononenko, O. Baysal, L. Guerrouj, Y. Cao, M. Godfrey, Investigating code review quality: Do people and participation matter?, 2015, pp. 111–120. doi:10.1109/ICSM.2015.7332457.

[9] A. Bacchelli, C. Bird, Expectations, outcomes, and challenges of modern code review, in: Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, IEEE Press, 2013, pp. 712–721.

[10] G. Gousios, M.-A. Storey, A. Bacchelli, Work practices and challenges in pull-based development: The contributor's perspective, in: Software Engineering (ICSE), 2016 IEEE/ACM 38th International Conference on, IEEE, 2016, pp. 285–296.

[11] J. L. N. Audy, R. Prikladnicki, Desenvolvimento distribuído de software, Elsevier, 2007.

[12] A. M. Nicolaci-da Costa, M. Pimentel, Sistemas colaborativos para uma nova sociedade e um novo ser humano, Sistemas colaborativos. PIMENTEL, M.; FUKS, H.(Orgs.). Rio de Janeiro: Elsevier.

[13] V. Casey, Virtual software team project management, Journal of the Brazilian Computer Society 16 (2) (2010) 83–96.

[14] S. E. Page, The difference: How the power of diversity creates better groups, firms, schools, and societies, Princeton University Press, 2008.

[15] R. Prikladnicki, M. G. Perin, S. Marczak, A. C. S. Dutra, The best software development teams might be temporary, IEEE Software 34 (2) (2017) 22–25. doi:10.1109/MS.2017.50.

[16] Y. Yu, H. Wang, G. Yin, C. X. Ling, Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration, in: Software Engineering Conference (APSEC), 2014 21st Asia-Pacific, Vol. 1, IEEE, 2014, pp. 335–342.

[17] X. Xia, D. Lo, X. b. Wang, X. Yang, Who should review this change?: Putting text and file location analyses together for more accurate recommendations, in: IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER 2016), 2015, pp. 261–270. doi:10.1109/ICSM.2015.7332472.

[18] J. Jiang, Y. Yang, J. He, X. Blanc, L. Zhang, Who should comment on this pull request? analyzing attributes for more accurate commenter recommendation in pull-based development, Information and Software Technology 84 (2017) 48–62.

[19] C. Bird, T. Carnahan, M. Greiler, Lessons learned from building and deploying a code review analytics platform, Vol. 2015, 2015, pp. 191–201. doi:10.1109/MSR.2015.25.

[20] H. Fuks, A. B. Raposo, M. A. Gerosa, C. J. P. Lucena, Do modelo

de colaboração 3c à engenharia de groupware, Simpósio Brasileiro de Sistemas Multimídia e Web–Webmidia (2003) 0–8.

[21] V. R. Basili, D. M. Weiss, A methodology for collecting valid software engineering data, IEEE Trans. Softw. Eng SE-10, no. 6 (1984) 728–738.

[22] G. Gousios, M. Pinzger, A. v. Deursen, An exploratory study of the pull-based software development model, in: Proceedings of the 36th International Conference on Software Engineering, ACM, 2014, pp. 345–355.

[23] V. J. Schettino, M. A. P. Araújo, Implantação da prática de code review em um modelo de desenvolvimento de software: um estudo de caso.

[24] M. Beller, A. Bacchelli, A. Zaidman, E. Juergens, Modern code reviews in open-source projects: Which problems do they fix?, 2014, pp. 202–211. doi:10.1145/2597073.2597082.

[25] S. McIntosh, Y. Kamei, B. Adams, A. E. Hassan, The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects, 2014, pp. 192–201. doi:10.1145/2597073.2597076.

[26] M. E. Fagan, Design and code inspections to reduce errors in program development, IBM Syst. J. 15 (3) (1976) 182–211. doi:10.1147/sj.153.0182.

[27] E. Barr, C. Bird, P. Rigby, A. Hindle, D. German, P. Devanbu, Cohesive and isolated development with branches, Fundamental Approaches to Software Engineering (2012) 316–331.

[28] G. Gousios, A. Zaidman, M.-A. Storey, A. Van Deursen, Work practices and challenges in pull-based development: the integrator's perspective, in: Proceedings of the 37th International Conference on Software Engineering-Volume 1, IEEE Press, 2015, pp. 358–368.

[29] O. Baysal, O. Kononenko, R. Holmes, M. W. Godfrey, The secret life of patches: A firefox case study, in: 2012 19th Working Conference on Reverse Engineering, 2012, pp. 447–455. doi:10.1109/WCRE.2012.54.

Artigo Mapeamento Sistemático

Colocar aqui o artigo.