Estudo preliminar sobre os impactos do desenvolvimento orientado a testes no processo de revisão de código

Altieres de Matos[1], Larissa Bonifácio Roder[1], Luciane Baldo Nicolodi[1,2], Reginaldo Ré[3] e Marco Aurélio Graciotto Silva[1,3]

[1]Programa de Pós-Graduação em Informática – PPGI
Universidade Tecnológica Federal do Paraná (UTFPR)
Cornélio Procópio – PR – Brasil
{altitdb,larissaroder,baldoluciane}@gmail.com

[2]Programa de Pós-Graduação em Ciência da Computação – PCC
Universidade Estadual de Maringá (UEM)
Maringá – PR – Brasil

[3]Departamento Acadêmico de Computação
Universidade Tecnológica Federal do Paraná (UTFPR)
Campo Mourão – PR – Brasil
{reginaldo,magsilva}@utfpr.edu.br

Abstract. *Even using agile methods and practices like TDD and code review, many problems are encountered during coding. Code review and TDD focus on the quality of software, but their relationship is not sufficiently understood. The objective of this study was to investigate problems encountered by the code review process in software developed using TDD. A survey was conducted with software developers and architects of a medium-sized Brazilian company, seeking to characterize problems found by code reviews for code developed using TDD. The results characterize the main problems with respect to frequency, severity, time, effort, and techniques that could help to address such problems. With this study, it was possible to list important problems in code developed using TDD, presenting opportunities for TDD improvement to avoid such problems and for code review techniques to address them properly.*

Resumo. *Mesmo utilizando métodos e práticas ágeis como TDD e revisão de código, muitos problemas são encontrados durante a codificação. Revisão de código e TDD focam na qualidade do código fonte do software, mas a relação entre essas técnicas não é suficientemente compreendida. O objetivo desse estudo foi investigar problemas encontrados pelo processo de revisão de códigos em software desenvolvido com TDD. Foi realizado um survey com desenvolvedores e arquitetos de uma indústria de software nacional de médio porte, buscando a caracterização dos problemas encontrados por revisões de códigos desenvolvidos com TDD. Os resultados identificaram e caracterizaram os principais problemas quanto à frequência, criticidade, tempo, esforço e técnicas ou ferramentas que podem auxiliar no tratamento de tais problemas. Com a condução desse estudo foi possível elencar problemas importantes em códigos desenvolvidos com TDD, apresentando oportunidades para aprimoramento do TDD para evitar tais problemas e de técnicas de revisão de código para tratá-los adequadamente.*

1. Introdução

O desenvolvimento orientado a testes (TDD) é uma técnica iterativa de projeto e desenvolvimento de software organizada em três estágios: (i) definição de casos de teste relativos à funcionalidade, antes mesmo da implementação da funcionalidade em si; (ii) implementação do código fonte necessário para que o caso de teste seja executado com sucesso; e (iii) refatoração do código fonte, melhorando o design da solução sem alterar a funcionalidade que foi implementada [Beck 2001]. Quando um defeito de software é encontrado durante a utilização do TDD, casos de teste de unidade são adicionados ao pacote antes da correção feitas [Beck 2001]. Além disso, os casos de testes elaborados durante o TDD são definidos para satisfazer especificações, definindo uma medida de cobertura e forçando a indicar circunstâncias quanto à escrita do código da aplicação e dos casos de teste [Pachulski Camara e Graciotto Silva 2016].

No entanto, mesmo com a utilização de TDD, existem problemas no desenvolvimento de software [Rafique e Misic 2013]. Por exemplo, problemas relacionados ao uso da orientação a objetos, classes estruturadas de forma inadequada e que não possuem cobertura de testes prejudicam a qualidade do código, onerando o tempo necessário para correção de defeitos [George e Williams 2003]. Todos os problemas citados anteriormente são pontuais e comuns no desenvolvimento de software, sendo geralmente encontrados durante a sessões de revisão de código (Code Review). De fato, revisão de código pode ser complementar ao uso de TDD, permitindo a detecção de características indesejáveis, determinando oportunidades para refatoração do código. Desta forma, com a técnica de TDD focada em testes de unidade, testando pequenas partes, e a técnica de revisão de código centrada na análise tipicamente parte por parte do código escrito, em conjunto essas técnicas podem reduzir a ocorrência dos problemas previamente citados [George e Williams 2003]. Considerando um cenário comum de utilização de revisão de código, posterior a iterações de TDD e imediatamente anterior à integração do código resultante no código final do software em desenvolvimento, revisores de código devem considerar cuidadosamente os impactos dos problemas e mudanças solicitadas, e participar de reuniões com os demais desenvolvedores para estabelecer estratégias para adequar o código e prevenir erros similares nas próximas iterações de TDD e sessões de revisão de código [McIntosh et al. 2014].

Em estudos realizados anteriormente, percebeu-se uma forte ligação entre revisão de código e TDD, pois ambos focam na qualidade do desenvolvimento do código fonte do software [George e Williams 2003, McIntosh et al. 2014]. Observa-se também que a revisão de código é mais eficiente evolutivamente do que funcionalmente e que sua importância é significativa para o usuário final [Mäntylä e Lassenius 2009]. Nesse cenário, a adoção de atividades e critérios de qualidade são importantes para melhorar a eficiência e eficácia do uso em conjunto dessas técnicas. Dessa forma, a motivação deste trabalho é gerar insumos para trabalhos futuros quanto a essa integração, possibilitando a melhoria das técnicas e processos de empresas que trabalham com desenvolvimento de software e que utilizam técnicas de TDD e revisão de código. Logo, o objetivo deste estudo é investigar quais são os maiores problemas encontrados pelo processo de revisão de código quanto ao código de software desenvolvido com TDD, agregando assim nos estudos sobre TDD a partir das principais falhas encontradas pelo revisão de código.

O método de pesquisa utilizado foi o levantamento (*survey*). As questões aborda-

ram a identificação e caracterização quanto à criticidade e esforço dispendido em revisão de código, além de atividades utilizadas para abordar os principais problemas encontrados nos códigos revisados. O público alvo foi composto por desenvolvedores e arquitetos de software que realizam revisões de código em empresas que desenvolvem código usando TDD, sendo assim pessoas chaves em relação à qualidade do desenvolvimento do código fonte.

O restante deste artigo está organizado da seguinte forma. Na Seção 2 são apresentados os principais tipos de problemas identificados em revisões de código, conforme relatados na literatura e pela prática na indústria. Na Seção 3 são descritas as características do *survey*. Na Seção 4 são apresentados e discutidos os resultados, relacionando-os com os tipos de problema apresentados na Seção 2. Trabalhos relacionados são tratados na Seção 5. Na Seção 6 elencam-se as principais ameaças de validade deste estudo. Considerações finais e trabalhos futuros são descritos na Seção 7.

2. Revisão de código

A prática de revisão de código consiste no ato de inspecionar o código fonte em busca de problemas relacionados a qualidade interna do software [Mäntylä e Lassenius 2009]. Tipicamente, durante esse ato, um revisor inspeciona o código fonte de outro membro do time de desenvolvimento, linha a linha [Kalyan et al. 2016]. revisão de código tem sido realizada e pesquisada desde 1970 e, desde então, se tornou uma prática popular no desenvolvimento de software. Com o passar do tempo, a revisão de código vem se adaptando a métodos de desenvolvimento moderno e ágil [Kalyan et al. 2016]. Entretanto, atualmente não existe uma lista de problemas encontrados durante a revisão de código especificamente em cenários em que se emprega TDD. Para fins deste estudo, foi compilada a seguinte lista dos principais tipos de problemas encontrados nesse contexto, considerando a literatura científica e a prática da indústria:

- **P01: Código fonte com baixa cobertura de testes automáticos.** A cobertura de código também é usada como uma medida de qualidade do produto, na qual um caso de teste realiza a verificação e validação do código fonte de produção [Hemmati 2015].
- **P02: Código fonte com code smells.** Geralmente não são defeitos. Eles não são tecnicamente incorretos e podem não impedir o funcionamento do software. Em vez disso, eles indicam fraquezas no software que podem estar atrasando o desenvolvimento ou aumentando o risco de erros no futuro [Tufano et al. 2015].
- **P03: Código fonte com complexidade ciclomática alta.** A complexidade ciclomática é uma métrica de software que afere a quantidade de caminhos de execução independentes a partir de um código fonte [McCabe 1976].
- **P04: Código fonte com defeitos.** O termo defeito refere-se geralmente a algum problema com o software, quer com o seu comportamento externo quer com as suas características internas [Card 1990].
- **P05: Código fonte com design ruim.** O design geralmente é ruim quando seus objetos não são bem definidos, seus métodos ou procedimentos são extensos e complexos, e o código fonte não favorece a manutenibilidade [Fowler 1999].
- **P06: Código fonte com duplicação de código.** Significa que dois ou mais fragmentos de código fonte são idênticos ou muito semelhantes, particularmente em sua estrutura [Toomim e Graham 2004].

- **P07: Código fonte com layout errado.** Está relacionado à organização do código fonte, incluindo o uso de espaços em branco, agrupamento de código, linhas em branco, alinhamento, indentação, entre outros [ISO et al. 2010].
- **P08: Código fonte com testes automáticos errados.** Compreende-se como teste automático errado um caso de teste executado em condições especificas e cujo resultado não é igual àquele definido na especificação do software [Ma et al. 2015].
- **P09: Código fonte fora do padrão da empresa.** Conjunto de requisitos que estabelecem uma abordagem disciplinada e uniforme para assegurar a consistência do código fonte do software, ou seja, um padrão ou forma uniforme para a organização do código [ISO et al. 2010].
- **P10: Código fonte não atende o requisito.** Um requisito é uma condição ou capacidade que deve ser cumprida ou possuída por um sistema, componente do sistema, produto ou serviço para satisfazer um acordo, padrão, especificação ou outros documentos formalmente impostos [ISO et al. 2010].
- **P11: Código fonte sem testes automáticos.** Quando um caso de teste exercita um trecho de código, dizemos que esse trecho de código é coberto de testes. No entanto, existem cenários em que nenhum trecho de código é exercitado por casos de teste [Aniche et al. 2015].

3. Método

Neste estudo utilizamos o método *survey*, que compreende em coletar informações de um grupo de pessoas por amostragem de indivíduos a partir de uma grande população. Uma vez que o método baseia-se na amostragem, é tipicamente realizado primeiro o planejamento e depois a realização do estudo de acordo com o plano. Desta maneira, a realização pode ser dividida em um número de etapas sequenciais [Tichy e Padberg 2007, Linaker et al. 2015].

As seguintes características de problemas encontrados em revisões de código foram consideradas neste *survey*: frequência, criticidade, tempo, esforço e característica. A população avaliada consistiu de desenvolvedores e arquitetos de software que trabalham com desenvolvimento de software e com TDD. A amostragem foi não probabilística, classificada como amostra acidental, restringindo-se a apenas uma empresa. Isso deveu-se ao fato de que nela existia acompanhamento durante as iterações de desenvolvimento (*sprints*) para confirmar que o código era produzido utilizando TDD, permitindo a análise da questão tratada neste artigo.

O questionário foi desenvolvido na plataforma *Google Forms* e foi enviado por e-mail. Foi realizado um estudo prévio sobre TDD e revisão de código, a partir dos quais, e considerando as variáveis previamente estabelecidas, foi preparado o questionário a ser utilizado no survey. Para aferir a qualidade do instrumento construído, foram realizados a evolução do questionário por meio de pesquisador mais experiente e *surveys* pilotos. No primeiro estudo piloto, enviamos o questionário para uma amostra reduzida de pessoas, de empresa distinta daquela alvo do questionário final, com intuito de encontrar possíveis falhas nas perguntas elencadas. Após receber as respostas, os dados foram analisados e algumas perguntas foram ajustadas. Após a evolução do questionário, o mesmo foi enviado novamente para uma nova amostra, diferente da anterior e pertencente a um setor específico da empresa alvo (e que não participou como sujeito do questionário final), com propósito de garantir que o questionário atendia os propósitos esperados. Desta vez, após

análise das respostas e avaliação do questionário, pudemos confirmar que ele estava apto para ser enviado para a amostra da empresa alvo.

O questionário final utilizado na coleta dos dados foi dividido em 7 passos e 9 questões, além do Termo de Consentimento Livre e Esclarecido (TCLE) de forma a prover segurança aos respondentes[1].

4. Resultados

O questionário foi divulgado via e-mail para o grupo de desenvolvedores e arquitetos de empresa de software nacional de médio porte, com cerca de 450 profissionais que atuam no desenvolvimento de software para mercado financeiro, abrangendo 101 pessoas. Após disponibilizado por 2 dias, 32 respostas foram obtidas[2]. Para análise, retirados dados que não estavam de acordo com o esperado, como por exemplo pessoas que não são parte da população almejada.

As primeiras duas perguntas estão relacionadas à função desempenhada na empresa e à execução de atividades de revisão de código. Dos 32 respondentes, 25 eram desenvolvedores e 7 são arquitetos, conforme apresentado a seguir:

- **Q1. Qual é sua função?** O questionário contou com 25 respostas de desenvolvedores, correspondendo a 78,1% da amostra, e 7 respostas de arquitetos, correspondendo a 21,9%. Embora existisse a opção de informar outras funções, nenhum dos 32 sujeitos afirmou exercer funções distintas.
- **Q2. Você faz revisão de código?** Dos sujeitos que responderam o questionário, 25 afirmaram realizar revisão de código, sendo 7 arquitetos e 18 desenvolvedores, e correspondem a 78,1% da amostra. Apenas 2 sujeitos não realizam revisão de código, composta apenas por desenvolvedores, e correspondem a 6,3% da população. Por fim, 5 sujeitos não quiseram informar ou não sabem se realizam revisão de código, correspondendo a 15,6% da população e sendo composta apenas por desenvolvedores. Esses 7 sujeitos, por estarem fora do perfil da população investigada, foram excluídos do restante do estudo.

Em relação aos problemas encontrados pelas revisões de código, as questões Q3 e Q4 trataram da frequência com que eles eram encontrados e se existiam problemas não mencionados na lista apresentada na Seção 2.

- **Q3. Quais problemas são frequentemente encontrados durante a revisão de código?** Para cada problema elencado na revisão de código, o sujeito respondente poderia escolher, segundo a escala *Likert*, as opções: nunca (N), raramente (R), algumas vezes (AV), frequentemente (F) e sempre (S). Sumarizamos as respostas na Tabela 1, relacionando a quantidade de sujeitos que escolheram determinada opção com o problema relacionado. Após realizar a sumarização das respostas, realizamos o cálculo da Moda (M) para cada problema elencado. Desta forma, os problemas que são encontrados com mais frequência são código fonte com baixa cobertura de testes automáticos (P01) e sem testes automáticos (P11). De outro

[1]O questionário completo está disponível em `https://github.com/altitdb/utfpr/blob/master/eres2018/questions.pdf`.

[2]Os dados coletados estão disponíveis em `https://github.com/altitdb/utfpr/blob/master/eres2018/answers.xlsx`.

	N	R	AV	F	S	M
P01: Com baixa cobertura de testes automáticos	0	3	10	11	1	4
P02: Com *code smells*	0	1	11	11	2	3
P03: Com complexidade ciclomática alta	0	1	14	10	0	3
P04: Com defeitos	1	8	16	0	0	3
P05: Com *design* ruim	0	2	15	6	2	3
P06: Com duplicação de código	0	8	11	3	3	3
P07: Com *layout* errado	2	13	7	3	0	2
P08: Com testes automáticos errados	2	9	11	3	0	3
P09: Fora do padrão da empresa	0	14	8	3	0	2
P10: Não atende o requisito	4	13	8	0	0	2
P11: Sem testes automáticos	0	4	8	13	0	4

Tabela 1. Relação de problemas encontrados em revisões de código e respectiva frequência. As colunas correspondem à: nunca (N), raramente (R), algumas vezes (AV), frequentemente (F), sempre (S) e cálculo da moda (M).

lado, os problemas que possuem menor frequência são código fonte com *layout* errado (P07), fora do padrão da empresa (P09) e que não atende o requisito (P10).

- **Q4. Existem problemas que são frequentemente encontrados durante a revisão de código e não foram citados? Cite esses problemas, descrevendo-o brevemente e informando quão frequentemente eles são encontrados.** Não obtivemos respostas para essa pergunta. Entendemos que as alternativas supriram as escolhas da amostra.

Identificados os problemas mais frequentes, buscou-se a seguir a caracterização deles, em especial quanto à gravidade, esforço necessário para a identificação e para a correção. Na Tabela 2, são relacionados os problemas mais graves (Q5) e os que demandaram mais tempo para identificação (Q6) e para correção (Q7).

	Q5	Q6	Q7
P01: Com baixa cobertura de testes automáticos	9	5	5
P02: Com *code smells*	14	6	5
P03: Com complexidade ciclomática alta	11	6	15
P04: Com defeitos	14	13	6
P05: Com *design* ruim	10	5	16
P06: Com duplicação de código	8	5	5
P07: Com *layout* errado	3	3	4
P08: Com testes automáticos errados	7	13	9
P09: Fora do padrão da empresa	1	3	1
P10: Não atende o requisito	11	12	12
P11: Sem testes automáticos	13	2	7

Tabela 2. Respostas das questões Q5, Q6 e Q7.

- **Q5. Quais são os problemas mais graves encontrados durante a revisão de código?** Com os resultados obtidos podemos observar que os problemas mais críticos são código fonte com *code smells* (P02) e com defeito (P04). O problema menos crítico observado é o código fonte fora do padrão da empresa (P09).

- **Q6. Quais os problemas levam mais tempo para serem encontrados na revisão de código?** De acordo com os resultados, os problemas que levam mais tempo para serem encontrados são código fonte com defeitos (P04) e com testes automáticos errados (P08). Dentre os problemas, o último a aparecer é o código fonte sem testes automáticos (P11).
- **Q7. Quais itens reportados na revisão de código demandam mais esforço para correção?** Os resultados demonstraram que o problema que demanda mais esforço para correção é o código fonte com *design* ruim (P05). De todos os problemas relacionados, o código fonte fora do padrão da empresa (P09) apareceu em último.

Finalmente, foram investigados fatores relacionados às respostas anteriores que envolvam a aplicação de técnicas e ferramentas de revisão de código, além de outros elementos considerados pertinentes à pesquisa quanto à amostra da população considerada:

- **Q8. Quais técnicas, práticas, *plugins* ou ferramentas você utiliza para reduzir problemas na revisão de código?** As ferramentas que tiveram destaques foram *SonarQube* e *SonarLint*, citadas por 7 sujeitos cada. Além disso, outra ferramenta mencionada foi o Comparador de Arquivos, citado por 3 sujeitos. A experiência e ferramenta Stash foram citadas 2 vezes por cada sujeito. Foram citadas apenas 1 vez as seguintes técnicas, práticas, *plugins* e ferramentas: Ferramentas próprias (não foram detalhadas pelos respondentes), *Test Driven Development*, *TestCoverage*, Ferramenta de Versionamento (*WinMerge*), *Pair Programming*, *Bit-Bucket Code Review*, Bom Senso e *Checklist*.
- **Q9. Caso você tenha alguma sugestão ou queira complementar suas respostas com algo que o questionário não abordou fique a vontade para descrever neste espaço.** Para esta pergunta, os respondentes levantaram alguns pontos que podem ser adicionados a pesquisa. São eles: (i) Nível de conhecimento das técnicas, práticas, *plugins* e/ou ferramentas utilizadas no desenvolvimento de software e (ii) Realizar classificação direcionada a código novo e antigo, com intenção de descobrir os problemas mais próximos para cada realidade.

5. Trabalhos Relacionados

Embora não sejam relatados na literatura estudos que relacionam diretamente atividades de revisão de código desenvolvido com TDD, pode-se identificar trabalhos que tratam desses elementos individualmente e que permitam relação com este trabalho: problemas encontrados em revisões de código no contexto de métodos ágeis e problemas em TDD.

Quanto à revisão de código e como ela pode ser melhorada, Bernhart *et al.* (2010) apresentaram um método baseado na evolução do uso de revisão de código junto com uma ferramenta de revisão de código. Usando uma abordagem contínua para revisão de código, a sobrecarga de revisão pode ser reduzida e a eficácia e a aplicabilidade em ambientes ágeis podem ser melhoradas [Bernhart et al. 2010].

Considerando outra prática típica de métodos ágeis, a programação em pares, Swamidurai *et al*. (2014) realizaram um estudo sobre o uso conjunto dessa com técnicas mais leves de revisão de código, mais especificamente a revisão por pares. Eles demonstraram a maior eficácia da revisão por pares em comparação com a programação

em pares no contexto do TDD. Em suma, eles forneceram evidências de que programas de qualidade igual podem ser produzidos com menor custo usando revisão por pares [Swamidurai et al. 2014].

No âmbito de TDD, Fucci *et al.* (2014) comentam sobre a limitação de estudos empíricos sobre TDD e se desenvolvedores estão seguindo o ciclo *test-code-refactor*. De acordo com eles, nenhuma das pesquisas até aquele momento incluíam a conformidade do processo de TDD como parte fundamental da análise. Desta forma, eless analisaram o impacto da conformidade no processo nos efeitos alegados pelo TDD na qualidade externa, produtividade dos desenvolvedores e qualidade de testes. Com base em uma pequena amostra, os autores demonstram preocupações sobre como TDD é interpretado, além de questionarem a viabilidade econômica da adoção estrita do TDD [Fucci et al. 2014].

Em continuidade ao trabalho anterior, Fucci *et al.* (2015) aprimoraram um modelo para execução do TDD. Esse modelo tinha como objetivo realçar dois fatores adicionais: habilidades de programação e testes de unidade. Desta forma, seria possível incluir tais habilidades em um modelo que representasse a relação que o esforço da atividade teste tem com a produtividade do desenvolvedor e a qualidade externa. Os dados coletados em ambiente acadêmico foram utilizados para avaliar a relação entre esforço de teste, qualidade externa e produtividade [Fucci et al. 2015].

6. Ameaças a validade

Em relação à validade interna, o instrumento de medida foi avaliado por um especialista e por dois estudos pilotos, com sujeitos compatíveis com a população investigada. Também foi possível identificar, pelas questões Q1 e Q2, sujeitos que responderam o questionário, mas que não eram da população de interesse. As respostas relativas a esses sujeitos foram removidas. Embora a participação fosse anônima, foi requerido que cada sujeito respondesse o questionário após autenticação com uma conta no serviço *Gmail*, garantindo assim que cada sujeito respondesse apenas um questionário. Entretanto, embora diversas ameaças quanto ao instrumento tenham sido abordadas, constatou-se uma limitação quanto ao tamanho da amostra e também a a diferença do tamanho das amostras de desenvolvedores e arquitetos, restringindo a comparação entre os tipos de papéis.

Em relação à ameaças de validade externa, observa-se que o estudo engloba apenas uma empresa de engenharia de software de médio porte. Desta forma, não é possível generalizar os resultados obtidos. Não obstante, novos estudos podem ser realizados, contemplando outras empresas, de diferentes tipo, portes e regiões do Brasil e do mundo.

Em relação à validade da conclusão, avaliou-se a confiabilidade das respostas. Em relação à questão Q3, calculou-se o *Cronbach's Alpha*, resultando no valor de $\alpha = 0{,}7606$, considerando toda a amostra respondente. Avaliando-se somente os sujeitos desenvolvedores, obteve-se o valor $\alpha = 0{,}7637$. Considerando-se os arquitetos de software, obteve-se o valor $\alpha = 0{,}7346$. Com base nesses resultados obteve-se consistência interna aceitável.

7. Conclusão

O objetivo desse estudo foi investigar problemas que são gerados durante o desenvolvimento de software utilizando TDD e que são encontrados durante o processo de revisão de código. Primeiramente, foram levantados os problemas considerados na revisão de

código por desenvolvedores e arquitetos. Um *survey* foi realizado para uma amostra de 101 pessoas que participam do processo de desenvolvimento de software de uma empresa nacional de médio porte, com foco no mercado financeiro, com 32 respostas.

A partir da análise dos resultados obtidos, foram identificados que os problemas mais frequentes de revisão de código são código fonte com baixa cobertura de testes automáticos e código fonte sem testes automáticos. Os problemas mais críticos são *code smells* e código fonte com defeito. Em relação aos problemas que levam mais tempo para serem encontrados na revisão de código, destacam-se o código fonte com defeitos e o código fonte com testes automáticos errados. Para problemas que afetam o esforço para correção, identificou-se o código fonte com *design* ruim.

Por fim, foi possível analisar as variações de percepção de problemas relacionados a revisão de código de acordo com a característica do sujeito, que neste estudo contemplou apenas dois grupos, sendo eles desenvolvedores e arquitetos. Com o resultado deste trabalho, foi possível elencar problemas importantes em códigos desenvolvidos com TDD, apresentando oportunidades para aprimoramento do TDD para evitar tais problemas e de técnicas de revisão de código para tratá-los adequadamente. Encorajamos também outros pesquisadores a replicarem este estudo, provendo maior confiabilidade e gerando mais conhecimento para a comunidade.

Referências

Aniche, M. F., Oliva, G. A., e Gerosa, M. A. (2015). Why statically estimate code coverage is so hard? a report of lessons learned. In: *29th Brazilian Symposium on Software Engineering*, p. 185–190. IEEE.

Beck, K. (2001). Aim, fire. *IEEE Software*, 18(5):87–89.

Bernhart, M., Mauczka, A., e Grechenig, T. (2010). Adopting code reviews for agile software development. In: *2010 Agile Conference*, p. 44–47. IEEE.

Card, D. N. (1990). Software quality engineering. *Information and Software Technology*, 32(1):3–10.

Fowler, M. (1999). *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, Boston, MA, USA.

Fucci, D., Turhan, B., e Oivo, M. (2014). Impact of process conformance on the effects of test-driven development. In: *8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, p. 10:1–10:10, New York, NY, USA. ACM.

Fucci, D., Turhan, B., e Oivo, M. (2015). On the effects of programming and testing skills on external quality and productivity in a test-driven development context. In: *19th International Conference on Evaluation and Assessment in Software Engineering*, p. 25:1–25:6, New York, NY, USA. ACM.

George, B. e Williams, L. (2003). An initial investigation of test driven development in industry. In: *18th ACM Symposium on Applied Computing*, p. 1135–1139, New York, NY, USA. ACM.

Hemmati, H. (2015). How effective are code coverage criteria? In: *2015 IEEE International Conference on Software Quality, Reliability and Security (QRS 2015)*, p. 151–156. IEEE.

ISO, IEC, e IEEE (2010). Iso/iec/ieee 24765:2010 – systems and software engineering – vocabulary.

Kalyan, A., Chiam, M., Sun, J., e Manoharan, S. (2016). A collaborative code review platform for GitHub. In: *2016 21st International Conference on Engineering of Complex Computer Systems (ICECCS)*, p. 191–196. IEEE.

Kollanus, S. (2011). Critical issues on test-driven development. In: *12th International Conference on Product-Focused Software Process Improvement*, volume 6759 of *Lecture Notes in Computer Science*, p. 322–336.

Linaker, J., Sulaman, S. M., Maiani de Mello, R., e Martin, H. (2015). Guidelines for conducting surveys in software engineering. techreport, Lund University, Sweden.

Ma, L., Zhang, C., Yu, B., e Sato, H. (2015). An empirical study on effects of code visibility on code coverage of software testing. In: *10th International Workshop on Automation of Software Test (AST 2015)*, p. 80–84. IEEE.

McCabe, T. J. (1976). A complexity measure. *Transactions on Software Engineering*, 2(4):308–320.

McIntosh, S., Kamei, Y., Adams, B., e Hassan, A. E. (2014). The impact of code review coverage and code review participation on software quality: A case study of the Qt, VTK, and ITK projects. In: *11th Working Conference on Mining Software Repositories*, p. 192–201, New York, NY, USA. ACM.

Mäntylä, M. V. e Lassenius, C. (2009). What types of defects are really discovered in code reviews? *Transactions on Software Engineering*, 35(3):430–448.

Pachulski Camara, B. H. e Graciotto Silva, M. A. (2016). A strategy to combine test-driven development and test criteria to improve learning of programming skills. In: *47th ACM Technical Symposium on Computing Science Education*, p. 443–448, New York, NY, USA. ACM.

Rafique, Y. e Misic, V. B. (2013). The effects of test-driven development on external quality and productivity: A meta-analysis. *Transactions on Software Engineering*, 39(6):835–856.

Swamidurai, R., Dennis, B., e Kannan, U. (2014). Investigating the impact of peer code review and pair programming on test-driven development. In: *IEEE SOUTHEASTCON 2014*, p. 1–5. IEEE.

Tichy, W. F. e Padberg, F. (2007). Empirical methods in software engineering research. In: *29th International Conference on Software Engineering (ICSE'07 Companion)*, p. 163–164. IEEE.

Toomim, M., B. A. e Graham, S. L. (2004). Managing duplicated code with linked editing. In: *20th IEEE Symposium on Visual Languages and Human Centric Computing*, p. 173–180. IEEE.

Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Penta, M. D., Lucia, A. D., e Poshyvanyk, D. (2015). When and why your code starts to smell bad. In: *37th International Conference on Software Engineering*, volume 1, p. 403–414, Piscataway, NJ, USA. IEEE.