PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
INSTITUTO DE CIÊNCIAS EXATAS E INFORMÁTICA
UNIDADE EDUCACIONAL PRAÇA DA LIBERDADE
Bacharelado em Engenharia de Software

Anderson Barbosa Coutinho
Henrique Alberone Nunes Alves Ramos
Mateus Santos Fonseca
Yan Max Rodrigues Sette Pinheiro

Trabalho Interdisciplinar de Software VI

Belo Horizonte
2020

1. **GQM (Goal, Question, Metric):**

GOAL: Verificar a efetividade de testes unitários em relação à bugs nos repositórios Javascript.

QUESTIONS:

Questão 1: Qual a relação entre a quantidade de testes unitários e a quantidade de bugs?

Questão 2: Qual a relação entre o coverage do código e a quantidade de bugs?

METRICS:

Métrica 1: $\frac{\textit{Quantidade de bug issues}}{\textit{Linhas de código de teste unitário}}$

Métrica 2: $\frac{\textit{Quantidade de bug issues}}{\textit{Coverage do código}}$

1.1. Tabela GQM:

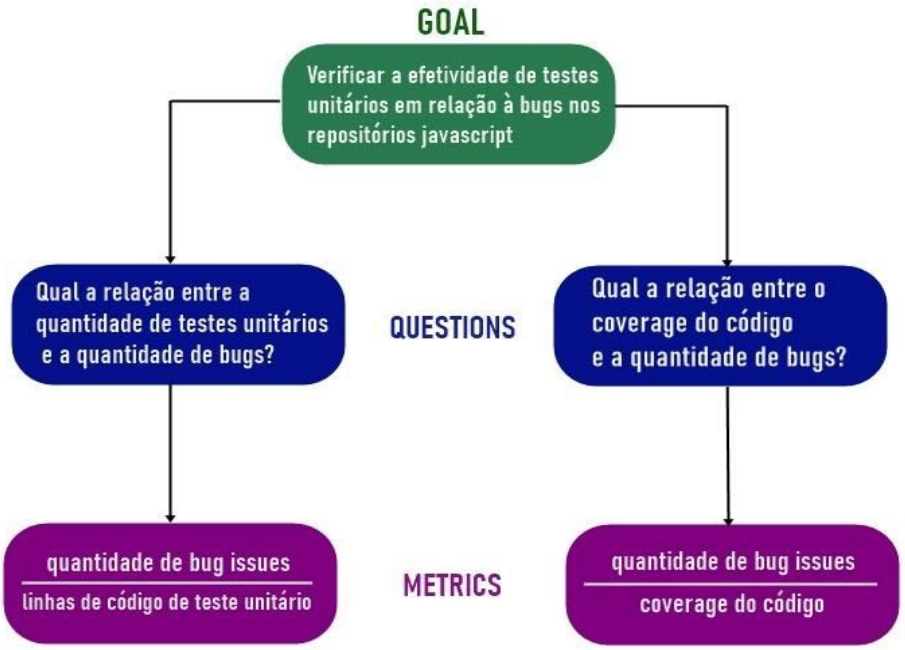

Figura 1. Tabela GQM do projeto.

2. Metodologia

Este trabalho consiste em 2 etapas, sendo a primeira concluída através da execução de scripts python para obter dados que servem de entrada para a segunda etapa, que consiste em análise manual dos repositórios e dos dados obtidos. Abaixo temos um exemplo inicial de como será o fluxo da execução do projeto:

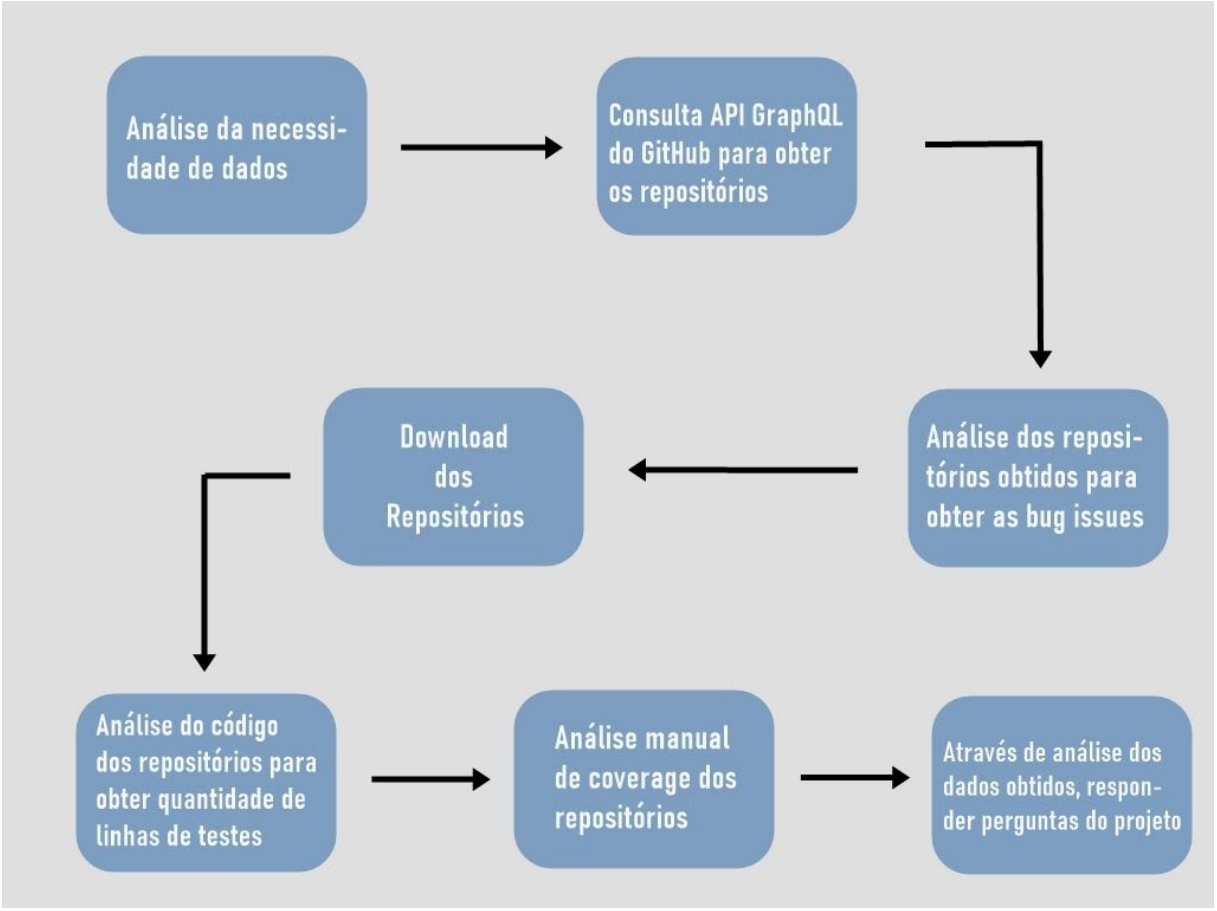

Figura 2. Overview da metodologia do projeto.

Na primeira etapa, tendo em vista que o Goal desta pesquisa depende de uma análise de repositórios GitHub cuja linguagem primária é JavaScript, faremos, através da execução de um script em Python, uma consulta a API GraphQL do GitHub que nos retornará os 100 repositórios JavaScript que mais possuem estrelas e suas respectivas issues. Para cada repositório, será realizada uma análise de todas suas issues com a finalidade de filtrar as que possuem BUG como label, gerando portanto, ao final, a quantidade total de bug issues do repositório (apenas issues que tiverem BUG como label serão consideradas, tendo em vista que a API do GitHub não é capaz de retornar o tipo da issue).

Em sequência, um outro script será desenvolvido, utilizando a biblioteca GitPython, cuja funcionalidade será baixar todos os repositórios retornados pelo primeiro script.

Finalmente, um terceiro script irá, através da análise do código fonte de cada repositórios, realizar a contagem de linhas de código de teste unitário. Esta contagem será baseada em sufixos comuns que códigos de testes possuem. Em sequência, este terceiro script irá exportar para um arquivo .csv as informações de linhas de código de testes obtidas de cada repositório analisado, juntamente com outras informações necessárias obtidas anteriormente.

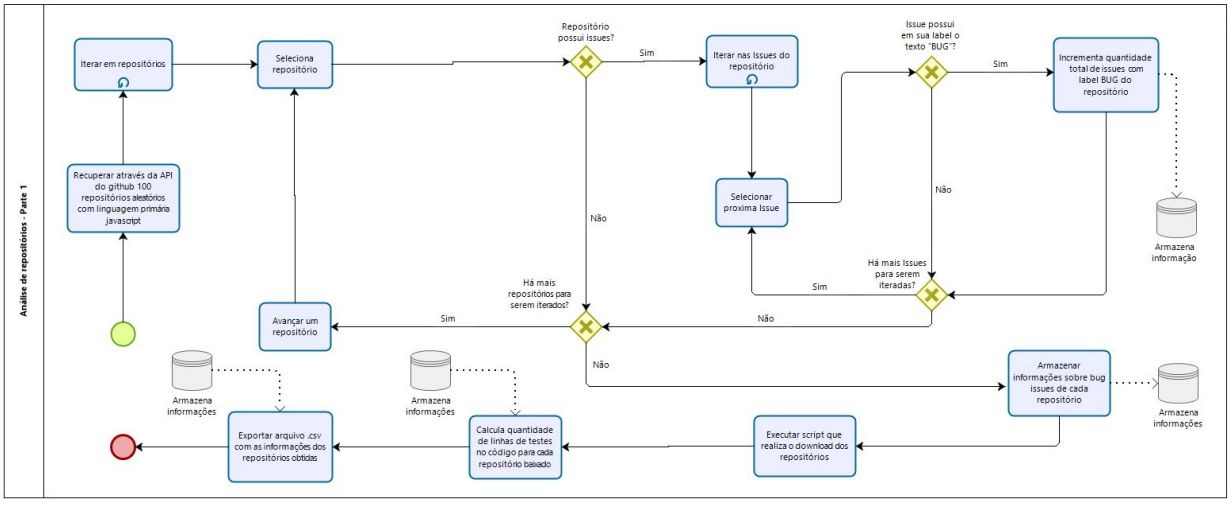

Figura 3. Exemplo do processo realizado pelos scripts 1, 2 e 3 da primeira etapa.

Na segunda etapa, inicialmente, uma análise manual dos repositórios será realizada, através da visualização de todos repositórios retornados pela consulta do primeiro script da primeira etapa. Para cada repositório, será analisado seu README, buscando a badge “coverage”. Para cada repositório que possua esta badge, o valor de coverage contido deverá ser exportado para um arquivo .txt, visando uma posterior concatenação com o arquivo .csv gerado anteriormente.

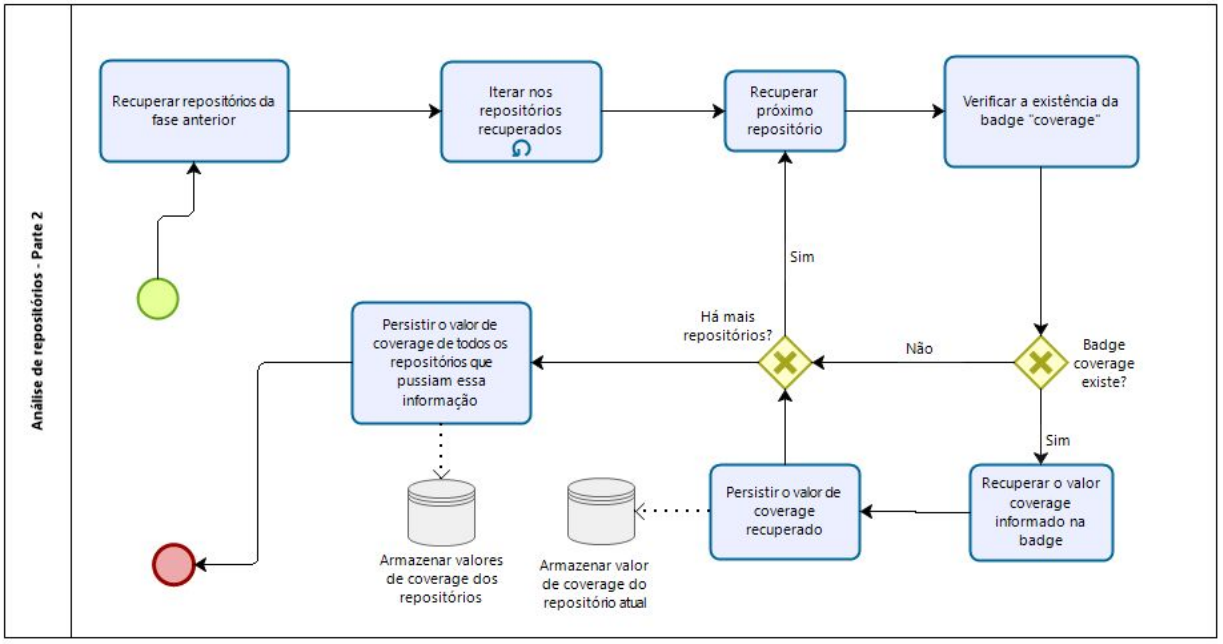

Figura 4. Exemplo do processo de análise manual de repositórios que possuem badge "coverage"

Finalmente, na etapa de análise dos dados obtidos, as métricas identificadas para buscar respostas às questões levantadas na pesquisa serão calculadas e as perguntas respondidas com base nos dados obtidos. Deverão ser analisadas a razão tanto quanto a proporcionalidade de crescimento de uma grandeza em relação à outra (e.g. o número de bugs é inversamente proporcional ao coverage).

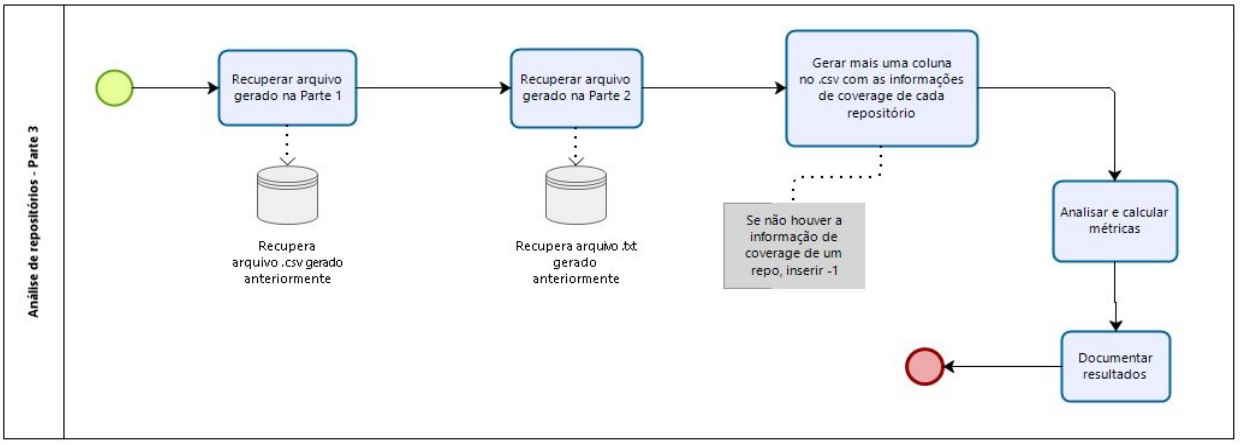

Figura 5. Exemplo do processo realizado na análise dos dados obtidos na segunda etapa

3. Trabalhos relacionados

O artigo **Learning Test-Driven Development by Counting Lines** é resultado de um experimento conduzido por Bas Vodde e Lasse Koskela para verificar a efetividade do desenvolvimento orientado a testes na refatoração do código e seus benefícios gerais, diferindo da pesquisa que será conduzida, pois esta procura estabelecer uma relação numérica entre cobertura de código e o número de *bugs* no código (B. Vodde e L. Koskela, 2007)

Em ***Driving Software Quality: How Test-Driven Development Impacts Software* Quality** (L. Crispin, 2006) a autora procura explicitar os *trade-offs* iniciais da adoção de TDD em um projeto, principalmente, quando não há conhecimento prévio por parte dos colaboradores. No entanto, é um trabalho focado no processo em si, enquanto este trabalho visa explicitar numericamente se há vantagens de adotar este paradigma.

No artigo **What Makes Testing Work: Nine Case Studies of Software Development Teams** nove estudos de caso de times de desenvolvimento para tentar entender quais práticas são mais efetivas para testar um software. Os métodos utilizados pelas empresas são: método tradicional não-especificado, método tradicional especificado, testes unitários escritos ao final do desenvolvimento, testes unitários escritos após o código em um ciclo iterativo de desenvolvimento. Para Calcular o valor dos processos de teste, foi calculada a relação entre testes bem-sucedidos e testes que falharam. A cobertura foi calculada a cada fim de semana do projeto. A principal conclusão ao fim do estudo foi que somente adotar XP, testes ou ambos, não é suficiente para garantir uma alta qualidade do produto final. O artigo é relevante à nossa pesquisa pois questiona a eficiência de XP e testes utilizados de maneira leviana e, além disso, toca na questão de cobertura, que é calculado a cada fim de semana do projeto.

O artigo **A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last?** (D. Fucci, 2017) tem como objetivo entender como os princípios básicos de TDD afetam a qualidade final de um

Software a partir de quatro “dimensões”: **granularidade**, **uniformidade**, ***test-first*** e **refatoração**. O estudo foi conduzido com dados coletados em quatro *workshops* sobre testes unitários e desenvolvimento orientado a testes, foram dadas cinco tarefas para que fossem desenvolvidas e, ao final das oficinas, os participantes deveriam responder duas perguntas à respeito da qualidade externa e produtividade dos desenvolvedores. Este artigo é fundamental pois questiona a importância dos princípios do TDD e seu ciclo.

O estudo **Test-Driven Development-Still a Promising Approach?** (S. Kollanus, 2010) está focado nas experiências relatadas sobre TDD. Foi realizada uma revisão sistemática da literatura para analisar as evidências empíricas atuais. A principal questão de pesquisa para a pesquisa foi: Existe evidência empírica sobre os benefícios sugeridos do TDD? Houveram muitos resultados contraditórios nos dados da revisão, que é um problema de validação. A questão principal com isso é a qualidade dos relatórios de pesquisa. Eles geralmente fornecem informações limitadas sobre o cenário da pesquisa. Em experimentos atípicos, há um grupo usando TDD e um grupo de controle que segue o "método de desenvolvimento tradicional". Os estudos de caso geralmente descrevem experiências de implementação do TDD em uma equipe de desenvolvimento. Tanto o processo TDD quanto o processo não TDD, em geral, são definidos muito brevemente (ou nem são definidos). Esperamos que ao analisar os repositórios, possamos trazer dados mais próximos do real, com informações mais completas e apresentando as métricas necessárias para responder se o número de testes impacta na qualidade do código.

O artigo **Implications of test-driven development: a pilot study** (R. Kaufmann, D. Janzen, 2003) tem como pauta os testes de software antes do desenvolvimento. Essa prática é recomendada por diversas metodologias de desenvolvimento ágil, devido a confiabilidade e melhoria na produtividade do programador. Para realizar o experimento foram selecionados dois grupos de estudantes, sendo um grupo realizando os testes antes do desenvolvimento e o outro depois do desenvolvimento, utilizando a linguagem de programação Java para desenvolver aplicativos gráficos para jogos. Ao final do experimento, foi observado que o grupo que realizou os testes anteriormente foi mais produtivo e o código

produzido pelos grupos tinham complexidade semelhante. Este artigo se mostra fundamental, pois possui relação direta com as implicações que utilização de TDD têm em relação à qualidade do software e, consequentemente, quantidade de bugs e complexidade.

Já o artigo "**Quality of Testing in Test Driven Development" (**A. Cauevic, D. Sundmark, S. Punnekkat, 2012**)** apresenta resultados de um experimento especificamente projetado para avaliar a qualidade dos casos de teste criados pelos desenvolvedores que usaram as tradicionais abordagens test-first (teste em primeiro) e test-last (teste em último). Em média, a qualidade dos testes no desenvolvimento orientado a testes foi quase o mesmo que a qualidade dos testes usando test-last. Entretanto, análises detalhadas de casos de teste, criadas pelo grupo de desenvolvimento orientado a teste, revelou que 29% dos casos de teste eram casos de teste "negativos" (com base em requisitos não especificados) mas contribuindo com até 65% para o índice de qualidade geral dos testes. No que tange ao test-first citado anteriormente, temos também o artigo "**On the effectiveness of the test-first approach to programming"** que apresenta uma investigação formal sobre os pontos fortes e fracos da topografia da abordagem de test-first. O principal resultado foi que os programadores test-first escrevem mais testes por unidade de esforço de programação. Por sua vez, um número mais alto de testes de programadores leva a níveis de produtividade proporcionalmente mais altos. Assim, através de um efeito em cadeia, o test-first parece melhorar a produtividade. Estas pesquisas, embora específicas da área de teste, são importantes para nosso projeto, tendo em vista que há uma análise da qualidade de testes e, consequentemente, qualidade do código.

Outros dois artigos importantes acerca de TDD devem ser citados, o primeiro: **Effects of Developer Experience on Learning and Applying Unit Test-Driven Development** (R. Latorre, 2014) é resultado de um experimento com **30 programadores** para avaliar a dificuldade de aprendizado de Frameworks orientados à teste. Este se relaciona à esta pesquisa pois a dificuldade de aprendizado pode ser um fator crucial ao número de bugs, mesmo adotando corretamente o TDD. Ainda no tópico de TDD, temos o segundo artigo: **A structured**

experiment of test-driven development (B. George e L. Williams, 2004), que fizeram um experimento com 8 programadores e 3 empresas nas quais os programadores foram aleatoriamente designados para um dos dois grupos: TDD e controle. Como resultado, observaram que os pares do grupo de TDD obtiveram aproximadamente 18% mais sucesso nos casos de teste em relação aos pares do grupo de controle. A relevância deste artigo se dá na análise quantitativa da qualidade do código (no que tange aos casos de teste) entre os grupos citados. Finalmente, o terceiro: **Does test-driven development really improve software design quality?** (D. Janzen, H. Saiedian, 2008) é um estudo de caso, rotulado como ICS, que examinou quinze projetos de software concluídos em um grupo de desenvolvimento por mais de cinco anos. Os quinze projetos incluíram os cinco projetos test-first e test-last da indústria quase-experimentos. Em todos os estudos, exceto o último, os programadores de test-first escreveram testes que cobriram uma maior porcentagem de código. O que demonstra, mais uma vez, resultados a respeito de test-first.

O artigo **The Impact of Agile Software Development Process on the Quality of Software Product** (P. Jain, A. Sharma and L. Ahuja, 2018) trata da importância dos métodos ágeis para evitar falhas de software que são levadas ao mercado. A razão para a falha pode estar ligada ao processo de engenharia de software por trás desses produtos. A qualidade é um aspecto importante para a produção, principalmente para discutir o impacto do processo de desenvolvimento de software ágil na qualidade do produto, definindo o mapeamento entre o processo de desenvolvimento de software ágil e vários atributos de qualidade. Dessa forma, a qualidade de software tem um importante papel enquanto é realizado o projeto. Vários fatores precisam ser satisfeitos para se ter um desempenho satisfatório. O desenvolvimento consiste em cinco fases que devem acompanhar padrões de qualidade durante o desenvolvimento. Através de vários estudos, foi demonstrado que todos os atributos de qualidade são importantes na perspectiva de ter um produto de qualidade. O artigo apresenta uma importância para o processo de desenvolvimento de software em geral para um produto de qualidade.

Atualmente, a metodologia ágil é bastante incorporada para melhorar as entregas e ampliar os seus fatores de sucesso que levam à satisfação do cliente. No entanto, algumas organizações ainda possuem dúvidas sobre a qualidade do software que é realizado com o desenvolvimento ágil, conforme é discutido no artigo **Current State of the Research in Agile Quality Development** (P. Jain, 2016). Dessa forma, é necessário garantir a qualidade dessas metodologias de maneira quantificável. Apesar das discussões dos modelos ágeis, chega-se à conclusão que o fator de qualidade precisa ser melhor integrado ao desenvolvimento ágil. É preciso novos estudos que incluam os fatores de sucesso durante a modelagem da qualidade para essa metodologia, mais uma vez tornando-se necessário o estudo relacionado aos testes unitários.

O artigo de **An Empirical Study of Test Cases in Software Testing** (N. Shete and A. Jadhav, 2014) enfoca a importância dos casos de teste e seu papel no teste de software usado nas indústrias de TI. De acordo com os autores, sem o caso de teste, o teste não seria possível. Um software possui um ciclo de vida, e os testes são uma fase importante desse ciclo, pois cumprem os requisitos do cliente. Dessa forma, pode-se concluir que a maioria das empresas utilizam casos de teste que oferecem qualidade ao software. Os casos de testes eficazes reduzem efeitos negativos no software. Tendo este fato como base, e levando em consideração o estudo conduzido pelo autores deste artigo, fica evidente a necessidade de se analisar a relevância e efetividade dos testes em relação aos bugs de um sistema.

Portanto, os testes tornaram-se uma prática amplamente difundida entre os praticantes, como é debatido no artigo **An Empirical Study of Bugs in Test Code** (A. Vahabzadeh, A. Milani, A. Mesbah 2015). Nessa ocasião, casos de testes são gravados para verificar se o código funciona conforme o esperado. Os testes de bugs podem ser divididos em bugs em produção e bugs de teste. O estudo destaca que os bugs de teste não recebem tanta atenção quanto aos de produção. Além disso, é revelado que o FindBugs, uma ferramenta popular de detecção de bugs, não é eficaz na detecção de bugs de teste. Nesse contexto, torna-se necessário

analisar tais correlações entre bugs de teste e métricas de software para identificá-los.

4. Referências

B. Vodde and L. Koskela, ”**Learning Test-Driven Development by Counting Lines**”, in IEEE Software, vol. 24, no. 3, pp. 74-79, May-June 2007

L. Crispin, ”***Driving Software Quality: How Test-Driven Development Impacts Software* Quality**”, in IEEE Software, vol. 23, no. 6, pp. 70-71, Nov.-Dec. 2006.

Roberto Latorre, **”Effects of Developer Experience on Learning and Applying Unit Test-Driven Development**”, Software Engineering IEEE Transactions on, vol. 40, no. 4, pp. 381-395, 2014.

Thomson, Christopher & Holcombe, M. & Simons, Anthony. ”**What Makes Testing Work: Nine Case Studies of Software Development Teams**”, Testing: Academic and Industrial Conference - Practice and Research Techniques 2009. TAIC PART ’09., pp. 167-175, 2009.

Fucci, D., Erdogmus, H., Turhan, B., Oivo, M., & Juristo, N. (2017). “**A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last?**” *IEEE Transactions on Software Engineering*, *43*(7), 597–614. https://doi.org/10.1109/TSE.2016.2616877

B. George, L. Williams, ”**A structured experiment of test-driven development**”, Information and Software Technology, vol. 46, no. 5, pp. 337-342, 2004.

A. Cauevic, D. Sundmark, S. Punnekkat, ”**Quality of Testing in Test Driven Development**”, presented at the Quality of Information and Communications Technology, 2012.

H. Erdogmus, M. Morisio, M. Torchiano, ”**On the effectiveness of the test-first approach to programming**”, Software Engineering IEEE Transactions, vol. 31, pp. 226-237, March 2005.

D. Janzen, H. Saiedian, ”**Does test-driven development really improve software design quality?**”, IEEE Software, vol. 25, pp. 77-84, 2008.

S. Kollanus, ”**Test-Driven Development-Still a Promising Approach?**”, presented at the Quality of Information and Communications Technology, 2010.

R. Kaufmann, D. Janzen, ”**Implications of test-driven development: a pilot study**”, OOPSLA ‘03: Companion of the 18th annual ACM SIG-PLAN conference on Object-oriented programming systems languages and applications, pp. 298-299, 2003

P. Jain, A. Sharma and L. Ahuja, ”**The Impact of Agile Software Development Process on the Quality of Software Product**”, 2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), Noida, India, 2018, pp. 812-815.

A. Vahabzadeh, A. Milani, A. Mesbah, ”**An Empirical Study of Bugs in Test Code**”, IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 101-110, 2015.

P. Jain, L. Ahuja, A. Sharma, ”**Current State of the Research in Agile Quality Development**”, Proceedings of the International Conference on Computing for Sustainable Global Development, pp. 1877-1879, 2016.

N. Shete and A. Jadhav, ”**An empirical study of test cases in software testing**”, International Conference on Information Communication and Embedded Systems (ICICES2014), Chennai, 2014, pp. 1-5.