UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE EDUCAÇÃO ONLINE
ESPECIALIZAÇÃO EM BIG DATA, DATA SCIENCE E DATA ANALYTICS

Bruno Machado da Silva

APLICAÇÃO DE DATA ANALYTICS EM DADOS RETROSPECTIVOS DE COBERTURAS DE CÓDIGO EM TESTES AUTOMATIZADOS

Porto Alegre

2019

UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE EDUCAÇÃO ONLINE
ESPECIALIZAÇÃO EM BIG DATA, DATA SCIENCE E DATA ANALYTICS

Bruno Machado da Silva

APLICAÇÃO DE DATA ANALYTICS EM DADOS RETROSPECTIVOS DE COBERTURAS DE CÓDIGO EM TESTES AUTOMATIZADOS

Trabalho de Conclusão de Curso apresentado como requisito parcial para a obtenção do título de Especialista em Big Data, Data Science e Data Analytics, pelo curso de Pós-Graduação Lato Sensu em Big data, Data science e Data Analytics da Universidade do Vale do Rio dos Sinos – UNISINOS.

Orientadora: Prof. Dra. Josiane B. Porto

Porto Alegre

2019

Aplicação de Data Analytics em dados retrospectivos de coberturas de código em testes automatizados

Bruno Machado da Silva

[1]Universidade do Vale do Rio dos Sinos (UNISINOS)
Av. Unisinos, 950, Bairro Cristo Rei, São Leopoldo, RS – Brasil

Abstract. *Automated software testing can prevent numerous software defects in the early stages of development. Among the tools that do automated testing there are code coverage libraries, which instrumenting the source code, allowing the developer to know exactly the points that have not yet been executed and highlighting those that need further verification as well. These tools produce a series of data that can be analyzed, creating the possibility to store and analyze the data. So using a non-parametric statistical approach, the study verifies developer profile data and code coverage data produced by libraries, to find the features that best appoint developers that need training in automated software testing. The results show characteristics from low testing code coverage profiles. It's point to possible groups that need more focus on internal trainings.*

Resumo. *Os testes automatizados de software são capazes de, em conjunto, prevenir inúmeros defeitos de software nas fases iniciais de desenvolvimento. Entre as ferramentas que compõem estes tipos de testes, existem as bibliotecas de cobertura de código, que, ao instrumentarem o código-fonte, permitem que o desenvolvedor saiba exatamente os pontos que ainda não foram executados, ou ainda destacando os que carecem de verificações mais profundas. Estas ferramentas produzem uma série de dados que podem ser analisados, tornando possível guardá-los e analisá-los. Então usando uma abordagem de estatística não paramétrica, o estudo verifica os dados cadastrais de desenvolvedores, cruzando com os dados de cobertura de código produzidos por estes, a fim de econtrar as características que melhor apontem os desenvolvedores que precisariam de treinamentos em testes automatizados de software. Os resultados mostraram as características cadastrais dos usuários que possuem menor porcentagem de cobertura de código em testes, indicando assim possíveis grupos de canditados aos treinamentos internos.*

1. Introdução

Atualmente, o uso de softwares com a finalidade de otimizar as tarefas diárias se mostra cada vez mais presente e necessário. Mas nem sempre isso ocorre de maneira que garanta uma boa experiência ao usuário, principalmente quando este se depara com problemas de mal funcionamento.

Para Emanuelsson e Nilsson (2008) os defeitos de software podem dar origem a vários tipos de erros e quase todos os softwares contêm defeitos. Alguns dos defeitos são encontrados com facilidade, enquanto outros nunca são encontrados, normalmente porque surgem raramente ou até mesmo não são percebidos. A grande necessidade do correto funcionamento de softwares tornou latente a discussão em cima de garantias de

qualidade do software, algo que tem sido objeto de debate há muito tempo. A principal preocupação é que muitos atributos de qualidade como confiabilidade, eficiência, facilidade de manutenção e segurança sejam cuidadosamente verificados.

De acordo com Bahamdain (2018) o desenvolvimento de software deve considerar mais do que apenas escrever o código, pois toda organização procura uma arquitetura boa, código confiável e testável, de modo a facilitar também o suporte e a manutenção. Para alcançar isso, a garantia de qualidade de código e a definição de padrões desempenham um papel importante.

A fim de diminuir preocupações com eventuais problemas em programas, é necessário observar para as práticas que possam ajudar a limitar possíveis erros em um software, preferencialmente durante a fase de codificação. Para isso, as ferramentas de automatização de revisão de código e os testes automatizados se mostram úteis.

A verificação de código e os testes possuem o objetivo da qualidade de código em comum, mas são tarefas realizadas de maneiras diferentes. As ferramentas automáticas de verificação de código fonte são uma estratégia útil para automatizar este processo. O teste de software, onde o programa é verificado faz parte das estratégias deste processo. Diante disso é possível chegar à cobertura de código, definida por Sakamoto et al (2010) como o grau em que o código-fonte de um programa foi testado.

O processo de cobertura de código costuma gerar grande volume de dados, estes poderiam ser utilizados para, adicionar valor às informações lançando mão de técnicas de análise de dados, através de estatística descritiva e técnicas de inferência estatística, como o teste U de *Wilcoxon-Mann-Whitney*. Diante deste contexo, este presente estudo de caso busca encontrar respostas para a seguinte questão de pesquisa: quais as características cadastrais possuem maior relação com baixas taxas de cobertura de código. Desta forma informações importantes a respeito de débitos técnicos, entre as diferentes variáveis que compõem os diferentes perfis de programadores e grupos de uma empresa, poderiam levar a um melhor direcionamento dos profissionais com maiores necessidades de treinamento em testes automatizados de software.

O objetivo principal deste artigo foi realizar um estudo de caso dos dados de cobertura de código, a fim de identificar possíveis características cadastrais que estejam relacionadas a menores porcentagens de cobertura de código, através da correlação com diferentes perfis de desenvolvedores em uma empresa estatal no Rio Grande do Sul? Para atingir o objetivo principal deste artigo e responder as questões supracitadas, definiu-se os seguintes objetivos específicos:

- Verificar o perfil dos desenvolvedores de software da empresa, objeto de estudo deste trabalho;
- Levantar os dados de cobertura de código de cada desenvolvedor que realizou pelo menos uma cobertura de código no período que compreende o objeto de estudo, tanto em linguagem C como Java, usando as ferramentas internas de teste da empresa onde o estudo foi realizado;
- Correlacionar os dados cadastrais dos desenvolvedor com os de cobertura de código, a fim de encontrar características que identifiquem grupos de indivíduos com necessidade de realização de treinamentos.

A fim de contribuir na identificação dos desenvolvedores que necessitam um maior

olhar para treinamento, a pesquisa propõe o estudo dos perfis, buscando características em comum entre desenvolvedores de software que tenham lacunas técnicas. Assim, será possível identificar indivíduos que tenham essas características para que sejam o foco dos treinamentos internos. Diferentemente dos estudos relacionados, como de Balogh et al (2015) e Zhang e Lee (2012), o foco do presente trabalho foi o desenvolvedor e suas dificuldades, não as tarefas realizada por este.

Além desta introdução, o restante do artigo está organizado seguindo a seguinte forma: A Seção 2 apresenta o referencial teórico para o desenvolvimento deste artigo; Na seção 3, os aspectos metodológicos adotados nesta pesquisa são apresentados. Logo após, na seção 4, são apresentados e analisados os dados levantados, através de dados retrospectivos de cobertura de código, e pôr fim a seção 5 traz as conclusões deste estudo.

2. Fundamentação Teórica

A fundamentação teórica utilizada na elaboração e execução da pesquisa realizada neste trabalho foi de apresentar os principais conceitos sobre verificação automática de código e teste de software.

2.1. Histórico de revisão de código

Fagan (1976) formalizou um método de revisão de código, onde se passava por módulos de programa, linha por linha, em uma reunião de grupo. Essa forma de revisão de código,chamada de *code review*, foi chamada de inspeção de código. O autor concluiu que as reuniões de *code reviews* podiam encontrar um número significativo de defeitos.

O método proposto por Fagan(1976) para a inspeção define *code review* como uma tarefa composta por várias fases, realizadas em grupo, dentre elas conhecidas como *Planning*, *Overview*, *Preparation*, *Inspection meeting*, *Rework*, *Follow-up*. O intuito desse processo de inspeção criado por Fagan, também chamado de *Fagan Defect Free Process*, tinha como objetivo validar se a saída de um processo estava em conformidade com os critérios de saída especificados.

Este método trouxe benefício para comunidade de desenvolvimento de software, devido a preocupação em definir um processo de validação de qualidade de código. Em um estudo na década de 90, realizado na empresa Hewlett-Packard, foi evidenciado que técnicas baseadas em inspeção de código geraram uma economia de custos na ordem de US$ 21 milhões (GRADY;SLACK,1994).

2.2. Evolução do processo de *code review*

Para Bholanath (2015) o importante é garantir a qualidade de um produto de software tanto reduzindo o número de defeitos ao nível mínimo possível, bem como detectando defeitos o quanto antes no processo de desenvolvimento. Diante dessa necessidade que surgiu há anos atrás, mas que ainda se mostra presente, a adoção da técnica de Fagan (1976) se tornou popular e teve evolução.

A grande quantidade de código criado a cada dia torna os métodos baseados em reuniões, para realização da inspeção, cada vez mais complexas e custosas. Johnson (1998) verificou que as reuniões de revisão de código possuíam um alto custo, pois aumentavam o tempo despendido na fase de desenvolvimento, e que abordagens alternativas poderiam superar este problema sem afetar a eficácia da detecção de defeitos.

Uma das recomendações de Johnson (1998) era investigar a revisão mediada por computador, ou seja, automatizar o processo de revisão de código de forma a liberar os programadores para as tarefas que geram valor de fato para o software, diminuindo o tempo necessário de revisão de código e até mesmo eliminando a revisão manual por completo. A revisão automática de código, segundo o autor, poderia reduzir a sobrecargas dos times, aumentar a acurácia e automatizar a coleta de dados para posterior análise.

Referente ao código-fonte, seria interessante lançar mão da abordagem dinâmica, onde o processo está essencialmente executando o código, o programa e verificando dinamicamente as inconsistências dos resultados fornecidos.

2.3. Custos de erro de software

Raja e Marietta (2012) sugeriram que o custo de retrabalho de software é um dos principais focos no desenvolvimento de software, já que o custo é um parâmetro importante que define o sucesso dos projetos de software. Zahra et al (2014) descreveram o retrabalho como uma "tortura", que pode consumir de 40% a 70% do orçamento total de um projeto. Em geral, o custo dos retrabalhos torna-se uma taxa considerada certa, e paga pelas empresas de software como parte do processo de desenvolvimento de programas.

O controle de qualidade é uma prática que não diminui a quantidade de trabalho para o desenvolvedores. Efe e Demirörs (2013) em um estudo de caso investigaram os impactos de retrabalhos de software nas estimativas de projeto. Como resultado, durante a fase de testes, foram encontradas 152 questões, que resultaram em mais de 133 dias por pessoas para a remoção de todos os defeitos. Com isso, durante a fase de teste do sistema, foi sempre necessário atribuir mais desenvolvedores do que o estimado especialmente para trabalharem na correção de *bugs*.

2.4. Teste de software

O teste de software é um processo de produção de programas onde é garantido o correto funcionamento e a confiabilidade deste para o cliente (KHAN; AMJAD, 2015). De acordo, com Naik e Tripathy (2008) os testes de software são tarefas intensas e altamente trabalhosas, porque geralmente são executadas e suas resultantes analisadas de forma manual. Pressman corrobora com essas informações, definindo teste de software como um processo de execução de um programa ou sistema com intenção de verificar erros.

Atualmente, o "Modelo V"proposto por Paul Rook no final dos anos 80 é considerado um dos mais aceitos para testes de software, um processo de desenvolvimento programas que pode ser retomado como a extensão do modelo em cascata. O Modelo V demonstra as relações entre cada fase do ciclo de vida do desenvolvimento e sua fase de teste associada. Em vez de descer de forma linear, as etapas do processo são dobradas para cima, após a fase de codificação para gerar a forma típica em V.

Segundo os autores Mathur e Malik (2010), o modelo V implanta um método bem estruturado, no qual cada fase pode ser implementada pela documentação detalhada da fase anterior. As atividades de teste, como o projeto de teste, começam no início do projeto bem antes da codificação e, portanto, economizam uma grande quantidade de tempo do projeto. O objetivo do Modelo V é melhorar a eficiência e a eficácia do desenvolvimento de software e refletir a relação entre atividades de teste e atividades de desenvolvimento. Os autores descrevem as diversas fases do modelo em V como sendo:

- Teste unitário: é o teste que se concentra em cada componente individualmente, garantido que estas pequenas partes funcionem corretamente, como uma unidade. São utilizadas intensamente técnicas de teste de caixa branca, exercitando caminhos específicos na estrutura de controle de um módulo para garantir cobertura completa e máxima detecção de erros.
- Teste de integração: é o tipo de teste que aborda a integração de componentes para formar um pacote de software completo. Utiliza técnicas de teste de caixa preta para resolver problemas relacionados a verificação da construção de programas.
- Teste de sistema: são os testes realizados em um sistema completo e integrado para avaliar conformidade do sistema com os requisitos especificados. Neste tipo de teste, não é requisitado nenhum conhecimento do design interno do código fonte ou lógica.
- Teste de aceitação: neste teste é feita verificação se o produto desenvolvido atende aos requisitos específicos do cliente. Geralmente o cliente faz esse tipo de teste em um produto quando este é desenvolvido externamente.

2.4.1. Testes automáticos de software

Os testes automáticos surgiram, para ajudar na tarefa de testes, que são ferramentas que aumentam a eficiência e efetividade da fase de teste de software. Segundo autores, o grande benefício dos testes automáticos de software é prover para a organização uma rica biblioteca de testes reusáveis, que facilitam a execução consistente de um conjunto deles. Isto facilita e cria algo que seria difícil utilizando-se testes manuais, devido a dificuldade destes em reproduzir as mesmas condições de forma repetida, quando erros são encontrados. No teste automático isso é simplificado, já que as pré-condições iniciais do sistema são configurados automaticamente, tornando fácil a reprodução de resultados de testes (NAIK; TRIPATHY, 2008).

O teste de unitário (ou teste de módulos), de acordo com Myers (2004) é um processo de teste de subprogramas, sub-rotinas ou procedimentos individuais em um programa. Neste tipo de teste, o foco são os blocos menores do programa, ao invés do todo. Logo, a automatização dos testes unitários se mostra extremamente útil na tarefa de programação, pois atua muito próximos à tarefa de codificação do programa.

Existem três motivações para se realizar um teste unitário automático. A primeiro, é uma maneira de gerenciar os elementos combinados de teste, já que a atenção é focada inicialmente em unidades menores do programa. Segunda, ele facilita a tarefa de depuração (o processo de identificar e corrigir um erro descoberto), pois, quando um erro é encontrado, sabe-se que ele existe em um ponto muito específico. Por fim, o teste unitário automático introduz paralelismo no processo de teste do programa, possibilitando o teste vários módulos simultaneamente (MYERS, 2004).

No trabalho de Felbinger et al (2018), foi definido testes unitários automáticos convencionais como métodos dentro de uma classe de teste. Esses métodos de teste são sem parâmetros. Cada teste unitário convencional tem por função explorar apenas um aspecto particular do comportamento da unidade sob teste. Estes testes automáticos contêm verificações, que são representadas como asserções, que comparam o comportamento observado com os resultados esperados como comportamento do programa.

Testes unitários automáticos consistem, normalmente, na interação de um ou mais objetos, trazendo estes para um estado inicial específico, para que se utilize a alimentação através do lançamento de um número de mensagens. Finalmente, após conservar os objetos em um determinado desejado, são verificadas as alterações que estes sofreram, validando os impactos das alterações nos objetos sob análise, ou ainda verificando o impacto no ambiente, como por exemplo arquivos de log, arquivos de sistema, imagens e etc.

2.4.2. Cobertura de código-fonte

Embora o processo de teste seja a tarefa de executar um programa com a intenção de encontrar erros (Myers, 2004), entender se um produto está suficientemente testado ou não, segundo Antinyan et al (2018), é uma tarefa bastante complexa. Para essa tarefa, surge a cobertura de código.

Para tarefa de mensurar o teste de software, a cobertura de teste ou cobertura de código se mostra uma medida importante, que segundo Sakamoto et al (2010) refere-se ao grau em que o código-fonte de um programa foi testado. Para Abdallah et al (2018) a cobertura de código é considerada uma maneira de garantir que os testes estejam testando o código, de fato, pois quando os testadores executam testes, eles estão apenas verificando os resultados obtidos, comparando estes com os esperados.

Segundo Tengeri et al (2016) a cobertura de código surgiu a partir do chamado teste de caixa branca (geralmente também chamado de teste baseado em estrutura). De acordo com os autores, esta é uma técnica de teste dinâmico que depende da cobertura de código para verificar sistematicamente a quantidade de testes necessários para atingir uma meta de completude, muito embora não esteja claro se as próprias métricas de cobertura podem prever a efetividade do teste. Nos testes de caixa branca, ferramentas de medição de cobertura de código são usadas por serem capazes de relatar diferentes tipos de taxas de cobertura, em termos de porcentagem, e também criar relatórios detalhados sobre os elementos de programa abrangidos e não cobertos.

No momento do teste de software, a medição de cobertura se mostra útil de várias maneiras, como melhorando o processo de teste, fornecendo informações ao usuário sobre o status do processo de verificação e ajudando a encontrar áreas que não foram cobertas. Então, para Jalote(2002), lançar mão da análise de cobertura de código para procurar áreas de um programa não percorrido por um conjunto de casos de teste, é criar uma medida quantitativa do teste e, por consequência, uma medida indireta da qualidade de código.

Segundo Antinyan et al (2018), existem várias medidas de cobertura de teste de unidade que são usadas para quantificar a suficiência de teste. Para os autores, as medidas mais populares para medir cobertura de código são cobertura de declarações, cobertura de decisões e cobertura de funções.

De acordo com Antinyan et al (2018), a cobertura de declaração é a porcentagem de declarações em um arquivo que foram disparadas durante a execução dos testes automáticos. A cobertura de decisão se preocupa com a porcentagem atingida durante uma execução de teste, para cada bloco de decisão de um determinado arquivo em inspeção. Já a cobertura de função, ou cobertura de método, é a porcentagem de cada uma das funções em um arquivo que foram disparadas durante uma execução de teste.

2.4.3. Técnicas para cobertura de código

Para tarefa de realizar a cobertura de código durante a fase de testes uma ferramenta de adequada é altamente desejável para atender a todos esses objetivos. Consegue-se isto lançando mão de algumas ferramentas existentes para principais linguagens de programação, como .NET e Java.

Diversas ferramentas e técnicas estão disponíveis para serem utilizadas na realização da cobertura de código fonte. Segundo Tengeri et al (2016) existem duas abordagens principais de instrumentação diferentes: uma a nível de código-fonte e outra a nível de bytecode, como demonstra a Figura 1. Os autores descrevem a instrumentação de código-fonte como feita ao se inserir sondas no código, antes da realização do build e da execução dos testes. O segundo método consiste em instrumentar a versão compilada do sistema, ou seja, o bytecode.

Na instrumentação de bytecode, existem duas ramificações de abordagens existentes. Na primeira, sondas que ajudam a detectar execução de código são inseridas logo após a compilação, efetivamente produzindo versões modificadas dos arquivos de bytecode. Na segunda abordagem, a instrumentação ocorre durante o tempo de execução, ao carregar uma classe para disparo. Essas duas diferentes abordagens para instrumentação de bytecode podem ser referidas como instrumentação de bytecode *off-line* e *on-line*, respectivamente (JaCoCo, 2019).

Devido a seus vários benefícios técnicos, como não requer o código-fonte, possuir maior liberdade para posicionar as sondas com precisão e liberdade para manipular os código, a instrumentação de bytecode é bastante predominante. No entanto, se os resultados precisarem ser mapeados de volta para o código-fonte, isso pode levar a imprecisões devido às diferenças entre as duas representações do programa, um cuidado que deve ser levado em conta na hora do usuário verificar a cobertura dos testes.

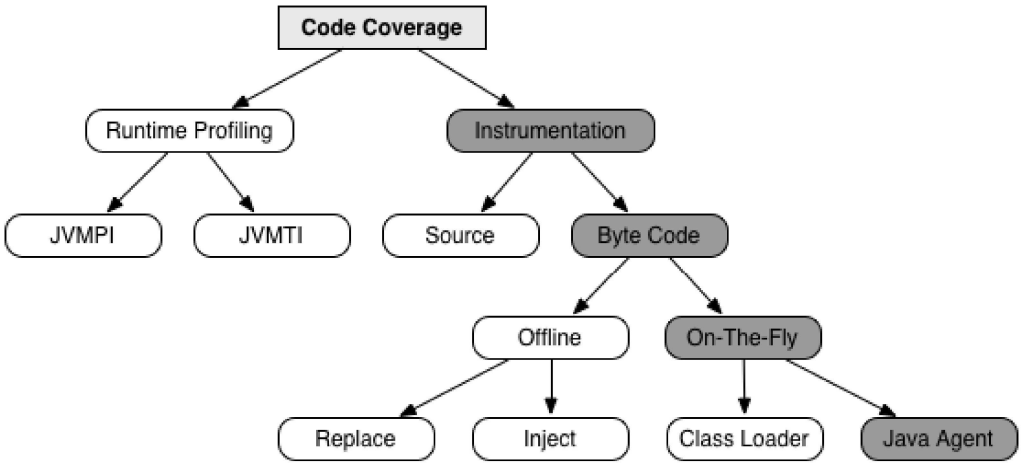

Figura 1. Fluxos de cobertura de código possíveis para as diferentes abordagens, na linguagem Java

Fonte: JaCoCo, 2019.

O funcionamento da instrumentação por bytecode pode ser representada por um grafo de fluxo de controle de um método, que pode ter edges. Cada limite, conhecido como edge, conecta uma instrução de origem com uma instrução de destino. No exemplo abaixo vemos um bytecode, onde o fluxo de controle possível podem ser representado por um gráfico. Os nós são instruções a nível de bytecode e os limites (edges) do gráfico representam o fluxo de controle possível entre as instruções. A instrumentação então ocorre nos pontos em que há edges no código, como demonstra a Figura 2.

2.4.4. Ferramentas de mercado para cobertura

Existem diversas ferramentas no mercado para coleta de cobertura de código durante o tempo de teste. A empresa onde o presente estudo foi realizado opta pelo uso de JaCoCo, para linguagem Java, e OpenCover.

A ferramenta JaCoCo (2019) é uma biblioteca gratuita de cobertura de código Java desenvolvida pela equipe do EclEmma. Ela realiza a abordagem de instrumentação de bytecode em tempo de execução, conhecida como instrumentação online. Suas medições podem ser facilmente integradas em testes, pois o Jacoco tem o poder de realizar instrumentação de bytecode tanto *online* como *offline*.

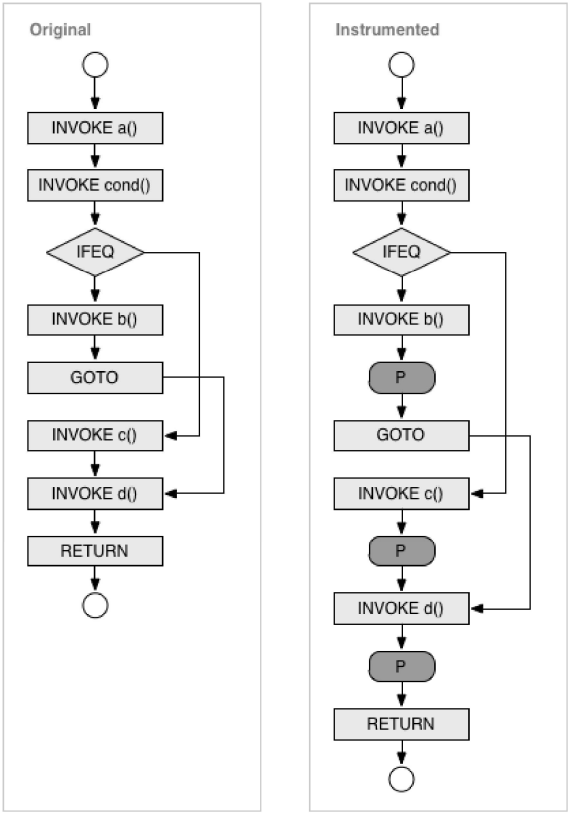

Figura 2. Exemplo de fluxo de instrumentação via bytecode.
Fonte: JaCoCo (2019)

Na instrumentação *online*, também conhecida como *on-the-fly*, as informações de cobertura devem ser coletadas em tempo de execução. O processo de instrumentação

acontece então em tempo real, durante o carregamento da classe usando os chamados Java *agents*. A instrumentação direta com uso do Java *agent* pode ser incluída na Java Virtual Machine, onde os programas são executados, sem qualquer modificação no projeto que está sendo testado, bastando modificar as propriedade de execução do ambiente.

O JaCoCo é uma das poucas ferramentas do mercado que realiza instrumentação *online* e possui uma comunidade de desenvolvedores ativa. Logo, devido a necessidade de não haver modificação no *pipeline* de criação e execução de testes automáticos da empresa, o JaCoCo foi a ferramenta escolhida para instrumentação *online*. Desta forma, não é necessário que o desenvolvedor altere as propriedades de um projeto seu que esteja sob teste, não tendo com ele o encargo de preparar e modificar o ambiente.

Para plataforma .NET, há a ferramenta OpenCover (2019). A ferramenta é atualmente a única de código aberto ativamente desenvolvida e mantida para a plataforma .NET. Assim como o JaCoCo, o Open Cover instrumenta o código durante o tempo de execução, não sendo necessária nenhuma modificação pelo desenvolvedor.

A ferramenta escolhida para instrumentação de codigo .NET foi o OpenCover, por ser uma das poucas opções para cobertura de código em .NET, com uma comunidade fortemente ativa. De modo geral, há dificuldade para encontrar ferramentas que realizem instrumentação em uma linguagem proprietária.

2.4.5. Trabalhos relacionados

Na Tabela 1 estão listados os artigos relacionados com o assunto desta pesquisa, enumerando os seus principais pontos. Estes artigos tiver grande importância para criação da pesquisa, servindo de base para o estudo.

No artigo escrito por Balogh et al (2015), é apresentado um método de medição de esforço de desenvolvedores de software, aplicado em um projeto de desenvolvimento contínuo com 17 desenvolvedores por um período de 7 meses. O método de medição de produtividade dos autores é dependente de dados de desenvolvimento, incluindo várias ações do desenvolvedor no editor de código-fonte, modificações de arquivos e registros de tempo. Para acumular informações importantes do projeto, os rastreamentos detalhados são registrados na interface gráfica onde os códigos eram modificados. O trabalho dos autores se difere do presente trabalho ao se preocupar com ações específicas relacionadas ao processo de codificação e vincular ao esforço de realização dos projeto, ao invés de usar os dados para encontrar perfis com dificuldades no processo, como é o caso do trabalho presente.

O estudo realizado por Zhang e Lee (2012) propôs um método para recomendar desenvolvedores para correção de bugs, a partir de coletas de relatórios de erros de repositórios de bugs, para sugerir desenvolvedores que para resolução de novos bugs. Primeiro, quando um novo relatório de bug era submetido ao repositório, o método procurava relatórios de bugs similares, empregando o algoritmo de recuperação baseado em clusterização. Diferente do presente trabalho, a preocupação não estudar as características dos desenvolvedores, mas sim clusteriza-los conforme o similaridade de tarefas anteriormente desenvolvidas.

No artigo desenvolvido por Terrel et al (2016) foi verificado que mulheres possuem maior aceitação de seus commits e pull-requests no GitHub, devido à qualidade de seus códigos, havendo evidências que possuem uma melhor produção destes. Este artigo foi de extrema importância para correlação do atributo sexo com o percentual de cobertura de código.

Tabela 1. Compilação de trabalhos relacionados

Referência	Técnicas de análise	Pontos importantes sobre o artigo
Balogh et al (2015)	Estudo de caso quantitativo, com dados retrospectivos de tarefas realizadas por desenvolvedores de software.	O método de medição de produtividade dos autores leva em conta os dados de desenvolvimento, incluindo várias ações do desenvolvedor no editor de código-fonte, modificações de arquivos e registros de tempo. Os autores lançaram mão de gravações de dados de todas as ações para facilitar uma posterior avaliação. O trabalho foca na dificuldade das tarefas realizadas por estes.
Zhang e Lee (2012)	Estudo de caso com análise quantitativa, lançando mão de algoritmos de *machine learning* para criação de um modelo.	Lançaram mão de um sistema de recomendação, sugerindo desenvolvedores de acordo com histórico de de correção de *bugs*, categorizando os desenvolvedores de acordo com tipo de *bug* resolvido.
Terrel et al (2016)	Estudo de caso com análise quantitativa, utilizando dados de repositório público para aplicar estatística descritiva.	Verificaram a diferença de aceitação de *pull-requests* feito por mulheres e homens no GitHub, chegando à conclusão que há maior aceitação por solicitações feitas por mulheres.

3. Metodologia de Pesquisa

Esta seção apresenta a metodologia usada para o desenvolvimento desta pesquisa, seu delineamento, a população e amostra utilizadas para coletar os dados, como a coleta de dados foi executada, a análise destes dados, bem como as etapas da pesquisa.

3.1. Delineamento da Pesquisa

O presente trabalho pode ser classificado como um estudo quantitativo, com dados retrospectivos, de caráter descritivo. Para atender aos objetivos deste trabalho foi realizado um estudo em duas etapas, onde em um primeiro momento foi realizado um levantamento bibliográfico, com o objetivo de abordar os conceitos de teste de software e cobertura de código fonte, identificando qualidade de código. De acordo com Gerhardt e Silveira (2009), a pesquisa quantitativa se centra na objetividade, considera que a realidade só pode ser compreendida com base na análise de dados brutos, recolhidos com o auxílio de instrumentos padronizados e neutros.

O objetivo descritivo do estudo de caso se dá em virtude de ter como objetivo buscar mais informações as características dos profissionais desenvolvedores de software dentro da empresa estudada, Banco do Estado do Rio Grande do Sul, a fim de identificar as características cadastrais que melhor representam o grupo de indivíduos que produzem baixos percentuais de cobertura de código na etapa de testes.

Para Triviños (1987), a pesquisa descritiva exige do investigador uma série de informações sobre o que deseja pesquisar, com a pretensão de descrever os fatos e fenômenos de uma realidade específica.

3.2. Objeto de estudo

Segundo Fonseca (2002), por ser um estudo retrospectivo o pesquisador não interfere sobre o foco do estudo, para que possa ser revelado como ele o percebe. Então, para manter a pesquisa com caráter neutro, o fluxo de trabalho dos desenvolvedores não foi alterado, sendo apenas realizada uma coleta retrospectiva dos dados através de consulta de banco de dados, Oracle e DB2.

A população ao qual o estudo foi desenvolvido consiste de profissionais que atuam na área de desenvolvimento de software, dentro da empresa Banco do Estado do Rio Grande do Sul. A amostra, composta de ocorrências de cobertura de código produzidas por 155 desenvolvedores únicos, foi retirada do banco de dados de ocorrências de cobertura de código, geradas a partir do fluxo de trabalho do desenvolvedor, ao usar as ferramentas de teste de C e Java da instituição.

A amostra é formada por 2534 ocorrências de cobertura de código, produzidas ao longo de um período de 1 ano, entre maio de 2018 e maio de 2019, sendo coletada através da consulta de banco de dados. Além desses dados de cobertura, foram coletados os dados cadastrais dos desenvolvedores que geraram cada uma dessas ocorrências. Houve necessidade de considerar apenas as amostras com cobertura de código maior que 20%, visto que estas geralmente representam sistemas ou componentes que possuem dificuldades para configuração de ambiente para teste. Como este não é o objeto de estudo do presente trabalho, não foram contabilizadas.

O dataset do trabalho foi composto por variáveis de porcentagem de cobertura de código, entre elas nome do arquivo, quantidade de linhas cobertas, quantidade de linhas não cobertas e código do usuário, e variáveis dos dados cadastrais dos desenvolvedores, entre elas sexo, data de nascimento, nível-função dentro da empresa e data de admissão.

3.3. Técnicas de Coleta de dados

Segundo os autores Gerhardt e Silveira (2009), a coleta de dados compreende o conjunto de operações por meio das quais o modelo de análise é confrontado aos dados coletados. Nesta etapa, o importante não é somente coletar informações, mas também obter informações de forma que se aplique posteriormente os tratamentos necessários teste das hipóteses do estudo.

Então, a coleta de dados foi realizada documental, como descrevem os autores Lakatos e Marconi (2003), onde os dados foram recolhidos no momento da ocorrência do fato ou fenômeno. Segundo os autores, a característica da pesquisa documental é que a fonte de coleta de dados está restrita a documentos, escritos ou não, constituindo o que se denomina de fontes primárias.

Ao longo do estudo, foram coletados os dados de cobertura de código, provindos de ferramentas de cobertura de código de arquivos Java e C, e dados cadastrais de desenvolvedores de software, tendo sido ambos coletados via querys de banco de dados na empresa. Após a consulta com filtro de data, os dados foram armazenados e mantidos em documentos do tipo CSV para consulta posterior na ferramenta de análise Jupyter (2019).

Os dados de cobertura de código foram coletados durante o ciclo de desenvolvimento da empresa, quando o desenvolvedor realiza construção da lógica de negócio dos programas e realiza os testes unitários a partir de ferramentas de testes. O ciclo de desenvolvimento na empresa ocorre em duas principais fases, sendo elas o desenvolvimento do código de regras de negócio e o teste automático desta. A etapa de codificação compreende a etapa onde o desenvolvedor transforma a regra de negócio em código. Na etapa seguinte, de teste automático de código, o desenvolvedor utiliza as ferramentas internas para criação de testes unitários para classe desenvolvida, para que a partir deles sejam geradas evidências de coberturas de código, assegurando um mínimo de percentual de execução da classe, garantindo uma mínima qualidade de código e gerando maior transparência no processo de desenvolvimento. As ferramentas internas lançam mão de *frameworks* de cobertura de código, como o JaCoCo (para linguagem Java) e o OpenCover (para linguagem C), geram dados referentes aos percentuais de coberturas cobertos pelos testes do desenvolvedor. Essa gravação é acionada pelo desenvolvedor, fazendo com que novas ocorrências de cobertura de código sejam enviadas ao banco de dados, vinculando a classe testada com o total de cobertura atingido.

3.4. Etapas da pesquisa

A primeira etapa da pesquisa se preocupou em realizar a coleta dos dados para posterior análise. Nela, foram feitas consultas em bancos de dados, para retirar as informações de porcentagem de código e dados cadastrais dos desenvolvedores. Foram feitas filtragens para garantir que os dados presentes estivessem na faixa de datas entre 1º de maio de 2018 e 1º de maio de 2019. Com os dados em mãos, foram feitos dois arquivos CSV, para cada uma das querys, para posterior tratamento e junção dentro das análises.

Na primeira parte da coleta, foram selecionadas as coberturas de código. Em cada ocorrência presente na tabela relativa à cobertura, é guardado o código do desenvolvedor que gerou aquela ocorrência. Então, a partir das ocorrências de cobertura foi possível coletar todos os códigos de desenvolvedores que foram necessários coletar também os dados cadastrais. Na segunda parte do trabalho foi realizada a análise exploratória e técnicas estatísticas para verificações de hipóteses de relações entre as diferentes variáveis que compunham o dataset. Para manipulação e visualização dos dados, foi utilizada a linguagem Python, juntamente com Jupyter (2019) e as bibliotecas Pandas (2019), Mathplotlib (2019), Seaborn (2019), Statsmodels (2019) e Scipy (2019).

Os dados obtidos foram analisados via estatística descritiva, analisando dados como a média, desvio padrão, variância, mediana e intervalo interquartílico. Segundo os autores Martins e Fonseca (2012), a estatística descritiva é um conjunto de técnicas que possuem como objetivo descrever, analisar e interpretar os dados de um determinado objeto de estudo.

Analisados os dados do objeto de estudo, foi realizada comparação das porcentagens de cobertura de código em relação ao sexo, à idade, à nível do cargo e ao tempo

de empresa dos desenvolvedores de software. para verificação das hipóteses estudadas, foi necessário lançar mão de testes não paramétricos. Segundo Martins e Fonseca (2012), os testes não paramétricos não exigem suposições com relação à distribuição do objeto de estudo, podendo ser aplicada tanto em dados que disponham de ordem ou até mesmo sejam nominais. Quando observadas as distribuições estudadas ao longo do trabalho, não foi possível assegurar de forma satisfatória a normalidade nas distribuições das porcentagens de cobertura de código. Logo, de forma a garantir adequadas inferências estatísticas, foram utilizadas técnicas de estatística não-paramétrica, através dos testes de *Wilcoxon-Mann-Whitney*, de *Kruskal-Wallis* e Qui-Quadrado.

Dentre os testes não paramétricos, o Teste U de *Wilcoxon-Mann-Whitney*, de acordo com Conover (1998) é um teste não paramétrico baseado em ranking, utilizado para todos os tipos de tamanho de amostragens. Esse teste, de acordo com o autor, é usado quando temos amostras independentes e queremos comparar sempre duas-a-duas as variáveis. Assim, ele é composto pela hipótese nula, que a as duas amostras provêm de uma única população, e a hipótese alternativa, que as duas amostras são de populações diferentes. O teste geralmente é utilizado quando amostra é pequena ou a variável numérica não apresenta uma variação normal satisfatória. Além disso, pode ser utilizado quando não há homogeneidade das variâncias.

O teste U de *Wilcoxon-Mann-Whitney* foi usado para comparação de percentual de cobertura nas hipóteses de influência do sexo e do estado civil na porcentagem. As amostras satisfazem as condições de serem amostras aleatórias de suas respectivas populações. Para sexo, as ocorrências utilizadas são amostras aleatórias de desenvolvedores femininos e masculinos, enquanto as ocorrências de solteiros e casados são amostras aleatórias de desenvolvedores dentro de dos respectivos estados civis. Além da independência dentro de cada amostra, há independência mútua entre as duas amostras.

Para Gibbons (2003), o teste de *Kruskal-Wallis* é a extensão natural do teste de Wilcoxon para verificação de duas amostras independentes, para a situação de k amostras mutuamente independentes de populações contínuas. A hipótese nula é que as populações k são as mesmas, então o teste de *Kruskal-Wallis* é usado para verificar se há alguma diferença nas medianas dos grupos comparados. Este teste foi usado especialmente para comparar os diferentes grupos de faixa etária, os diferentes níveis de cargo dentro da empresa e as diferentes faixas de tempo de empresa.

Para todas as análise que incluíram o teste , foi realizado teste post-hoc para determinar quais níveis da variável independente diferem um do outro. Para Zar (2010), pode-se afirmar que o teste de Dunn é apropriado para grupos com números desiguais de observações.

Outro teste usado ao longo do estudo para verificações das frequências da distribuição de categorização da cobertura de código foi o teste de Qui-quadrado, que de acordo com Viali (2008) é utilizado quando os dados da pesquisa se apresentam sob forma de freqüências em categorias discretas. Para o autor, o teste determina a significância de diferenças entre dois grupos independentes, e conseqüentemente, as freqüências relativas com que os componentes do grupo se enquadram nas diversas categorias. Para lançar mão também deste teste, as porcentagens de cobertura foram categorizadas de acordo com as normas da empresa. Coberturas acima de 80% são rotuladas como ‘OK’, abaixo

de 80% e maior que 70% são consideradas 'ALERTA', e menores que 70% são rotuladas como 'ERRO'. Isso permite a utilização de teste estatístico do Qui-quadrado como suporte a verificação das hipóteses. Essa rotulação permitiu que fosse aplicado o testes da frequência das distribuições entre as diferentes variáveis que compõem o dataset.

Por último, como suporte às conclusões de cada uma das verificações, foi usado o algoritmo Apriori, apresentado por Agrawal e Srikant (1994). Segundo os autores, o algoritmo Apriori, pode ser usado para descobrir de forma eficiente e rápida a relação de associação entre os conjuntos de itens em um dataset. Desta forma o algoritmo pode apresentar suporte para as hipóteses ao determinar quais são os mais frequentes combinações dentro do dataset.

4. Análise de Resultados

Após o levantamento de todos os dados necessários, foi realizada análise dos dados. Os resultados foram avaliados com uma definição de nível de significância, ou seja, quanto admitimos errar nas conclusões estatísticas, de 5%. Todos os intervalos de confiança construídos ao longo do trabalho, foram construídos com 95% de confiança estatística. Foram utilizando testes e técnicas estatísticas não paramétricas, pois as condições para a utilização de técnicas e testes paramétricos, como a normalidade e homocedasticidade não foram encontradas neste conjunto de dados.

4.1. Verificando o perfil dos desenvolvedores de software

A primeira variável utilizada para verificações de perfis foi a distribuição das coberturas de códigos. Os dados relativos às coberturas de código via testes unitários se mostraram dentro dos parâmetros de adequação da empresa, que considera que um componente deve ser testado em no mínimo 70% para que esteja em conformidade com as regras de boas práticas estabelecidas internamente. Pela amostra obtida é possível observar, conforme demonstra a Figura 3, um esforço dos desenvolvedores e comprometimento em atingir a cobertura ideal de 80%, estipulada pela empresa, pois mais de 50% das amostras estão acima do percentual de 80% de cobertura de código. Deve ser levado em conta que as exigências de cobertura de código em testes automatizado ainda são recentes na empresa.

Aplicando técnicas de binning na variável de cobertura de código fica ainda mais claro a inclinação da cobertura em direção à taxa de 100%. Por ser uma exigência da empresa, os desenvolvedores acabam tendendo a entregar uma taxa de cobertura minimamente alta, porém é possível verificar que ocorreram taxas de entrega acima do estabelecido como adequado, com uma maior frequência nas taxas entre 95% e 100%.

Verificando a amostra coletada, de um total de 2534 ocorrências, constatamos que a maioria dos desenvolvedores possuíam idades entre 22 e 60 anos, apresentando média de idade de 36 anos, como mostra distribuição abaixo. A partir desses dados, as idades foram categorizadas em três faixas, sendo elas "20 anos", "30 anos" e "40 anos ou mais".

Com relação ao tempo de empresa, os desenvolvedores apresentaram maior concentração ao redor de 10 e 5 anos de empresa. O tempo mínimo de empresa se deu em 1 ano, enquanto o tempo máximo ficou em 30 anos. Com relação ao sexo, ao todo somaram 12 desenvolvedoras do sexo feminino, enquanto a maioria dos desenvolvedores, 143, pertenciam ao sexo masculino. Com relação ao estado civil (variável EST_CIVIL), ao todo foram contabilizados 118 solteiros e 37 casados. Verificou-se então que a grande

maioria dos desenvolvedores são homens, ocorrendo maior frequência na faixa etária de 30 anos.

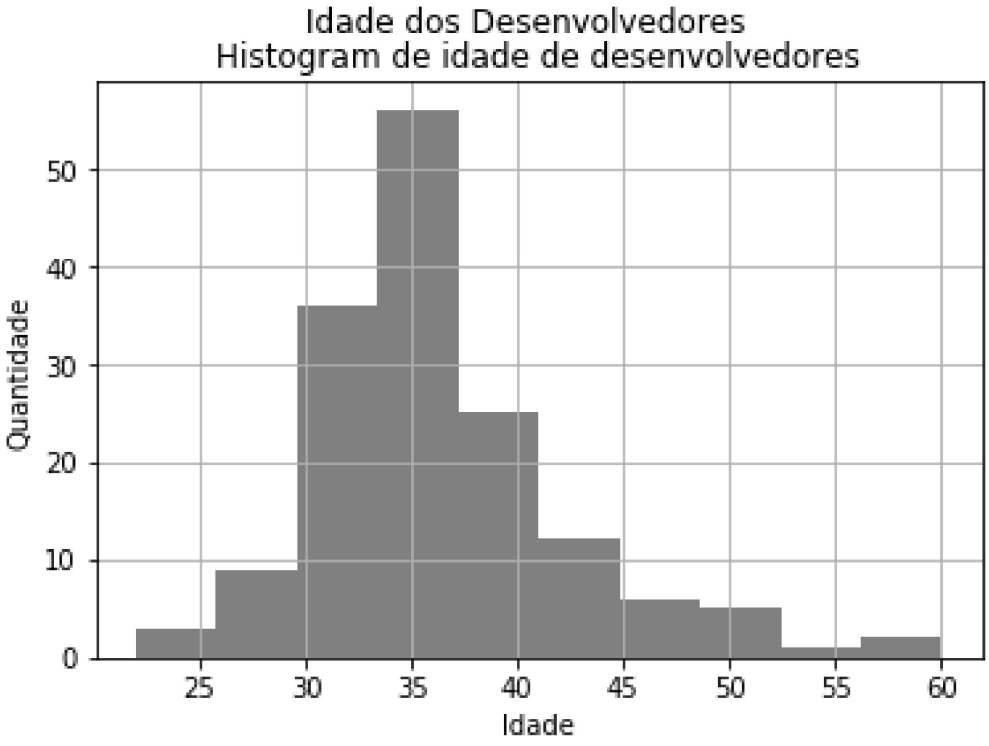

Figura 3. Histograma de idade dos desenvolvedores verificados no estudo.
Fonte: Elaborada pelo autor.

4.2. Correlacionando os dados cadastrais dos desenvolvedor com os dados de cobertura de código

4.2.1. Variável Sexo

De acordo com estatísticas publicadas pelo US Departament of Labor (2019) apenas 25 por cento das funções que envolvam matemáticas e computadores são ocupadas por mulheres. Em 2019, o mesmo instituto apresenta as estatística para a área de programação, revelando que apenas 21 por cento das posições no mercado de trabalho de programadores são ocupadas por mulheres. Essa estatística melhora na área de programação *web*, ficando em 32 por cento. No objeto deste estudo, é possível verificar que o mesmo acontece, pois aproximadamente 10% representam indivíduos do sexo feminino.

Pelo gráfico apresentado na Figura 4, e pela construção de uma tabela de contingência, foi constatado que o sexo feminino possui uma concentração maior de ocorrências em conformidade com os padrões definidos pela instituição, ou seja, com porcentagem de código acima de 80%. Quanto ao grupo do sexo masculino, foi constatado uma maior dispersão dos dados, quando comparado ao grupo de amostras do sexo feminino. O teste U de *Wilcoxon-Mann-Whitney* demonstra que há diferença na distribuição das porcentagens, quando levamos em conta o sexo, sendo a distribuição do sexo feminino melhor, quando comparada com sexo masculino.

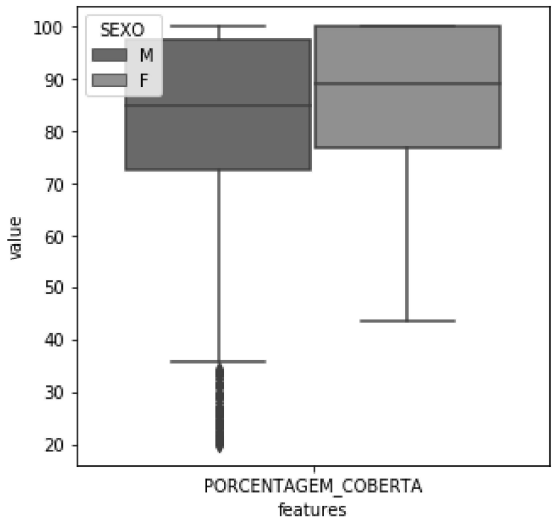

Figura 4. Gŕafico de *boxplot* da diferença de cobertura entre os sexos.
Fonte: Elaborada pelo autor.

Além do testes U de *Wilcoxon-Mann-Whitney*, foi aplicado o teste do Qui-Quadrado, para verificação da frequência dos dados entre as categorias de conformidade, através da construção da tabela de contingência. A verificação do Qui-quadrado, ao nível de significância de 5%, corroborou o algoritmo Apriori que demonstra que 15,7% das ocorrências do total de amostras foram relacionadas a homens que possuíam código-fonte fora de conformidade com a cobertura de código mínima, pela definição da empresa. Portanto existem evidências de diferença entre as distribuições de porcentagem de cobertura de testes entre desenvolvedores do sexo masculino e feminino.

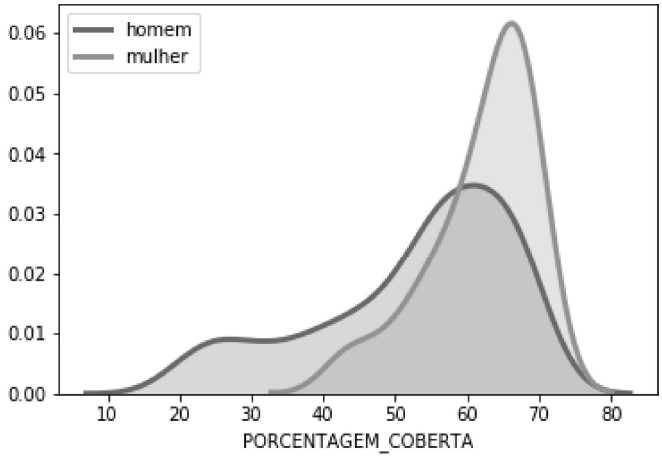

Figura 5. Histograma de densidade das distribuições do percentual de cobertura, quando comparados os sexos
Fonte: Elaborada pelo autor.

Além da verificação da diferença entre as distribuições e frequências, usando o sexo como variável explicativa, foi feito um filtro para comparar a distribuição apenas da faixa abaixo de 70% de cobertura, ou seja, para verificar se há diferença significativa

dentro da faixa considerada abaixo do ideal de cobertura de código definida pela empresa. Verificando o gráfico de densidade da distribuição, apresentado pela Figura 5, foi observada uma relativa diferença entre as médias nessa faixa.

Assim como pelo teste U de *Wilcoxon-Mann-Whitney* devemos rejeitar a hipótese nula, de que a distribuição das porcentagens de cobertura são iguais entre os sexos, pelo teste de Qui-Quadrado também há indícios de diferenças nas frequências entre coberturas categorizadas como ERRO e ALERTA. Então, indícios levam a crer que o sexo feminino tem um maior cuidado ao criar testes para que atinjam maior cobertura de código possível. Essa conclusão acaba corroborando com o estudo sobre qualidade de código, realizado por Terrell et al (2016), em relação a *pull-requests* efetuados por mulheres no GitHub, onde as mulheres tem média de aceitação em suas modificações de código do que homens. Isso leva a crer que realmente há um maior cuidado para produção de boas codificações, com nível de detalhamento alto, inclusive no quesito testes.

Os indícios que as mulheres são, em média, mais acertivas que os homens ao criar cobertura de código, corroboraram com a revisão bibliográfica. Na revisão da literatura sobre psicologia e sociologia realizada por Lemkau (1979), a autora descobriu que mulheres em ocupações dominadas por homens tendem a ser altamente competitivas, fazendo com que a performace seja acima da média de homens. Isso corroboraria com os achados do presente estudo, visto que o número de mulheres no objeto de estudo, e no mercado de tabalho da área de codificação de software, é muito baixo em relação ao número de homens, havendo assim uma possível pressão para produção ser de qualidade.

4.2.2. Faixa Etária

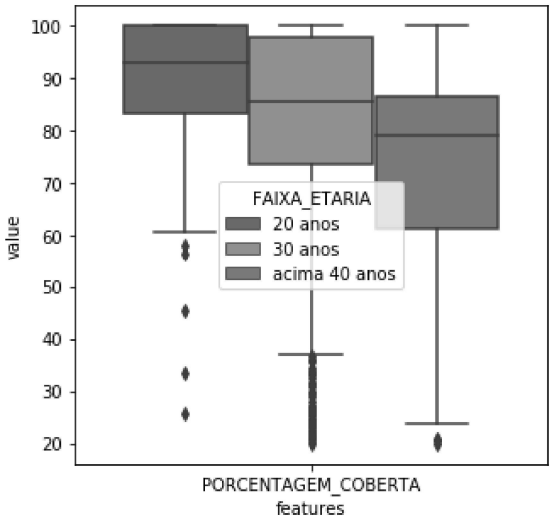

Figura 6. Gráfico *boxplot* das distribuições do percentual de cobertura, quando comparadas as diferentes faixas etárias.

Fonte: Elaborada pelo autor.

Observando as distribuições de cobertura de código, apresentada pela Figura 6, que levou em conta apenas a separação em grupos através da idade, vemos que a média

das distribuições possuem uma ligeira diferença entre os grupos que compõem, com uma tendência de diminuição da média de percentual de cobertura conforme a idade do grupo aumenta. Foi possível verificar que o grupo com faixa etária em 20 anos possui uma alta concentração entre 80 e 100% de cobertura, mediana de 93%. Além disso, 80% das amostras estão dentro da faixa considerada cobertura ideal. Já o grupo de 40 anos ou mais está disperso ao longo da extensão de coberturas possíveis, possuindo mediana de 78% e apenas 48% está dentro da faixa considerada ideal para a cobertura de código.

Pelo teste de *Kruskal-Wallis*, ao nível de significância de 5%, rejeitou-se a hipótese nula de que não há diferença significativa entre as médias das distribuições de porcentagem, quando comparadas as diferentes faixas etárias, pois haviam indícios que a média das distribuições eram diferentes. É interessante notar que ao utilizar o ANOVA pode ser verificado que 30% da variação da porcentagem de cobertura pôde ser explicada através utilizando apenas a variável IDADE do desenvolvedor.

Além desta verificação geral, foi feita a mesma verificação isolando-se os sexos. Foi notado que no sexo feminino, não houve diferença significativa entre nenhuma das faixas etárias, enquanto que no sexo masculino todas as faixa diferiram, como se pode observar no *heatmap* com as significância do teste de Dunn, apresentado na Figura 7. É interessante notar a divergência que há entre os achados do presente trabalho e a literatura, como no trabalho de Dostie (2011). Segundo o autor, foi encontrado declínio na produtividade em trabalhadores do sexo masculino, de áreas gerais, apenas quando estes atingiam 55 anos ou mais. De acordo com ele, a produtividade parece ser menor que seus salários, enquanto o inverso parece verdadeiro para homens mais jovens. No entanto, segundo o autor, mesmo nesses casos os diferenciais de produtividade são imprecisos demais para tirar conclusões acertivas. O mesmo ocorre no presente trabalho, pois poderiam haver outros fatores implícitos que tornem a produção de percentual de cobertura maior para jovens, como por exemplo o fato de receberem treinamentos.

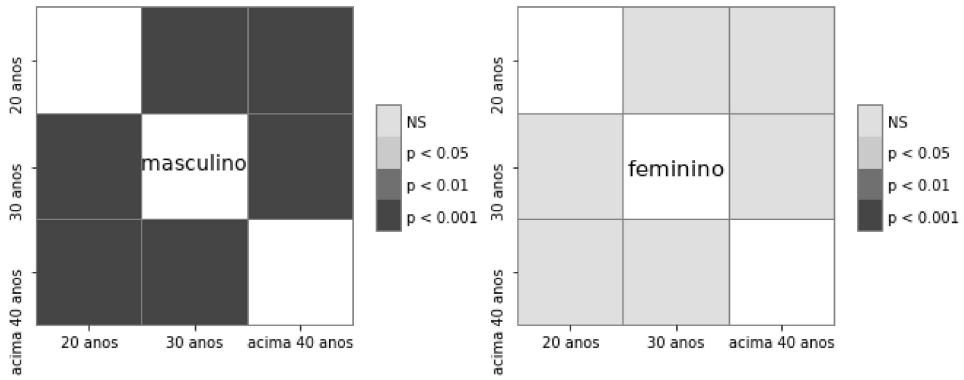

Figura 7. Gráfico *heatmap* do nível de significancia par a par, do teste de Dunn.
Fonte: Elaborada pelo autor.

4.2.3. Tempo de empresa

Para estudo do tempo de empresa, é necessário que seja contextualizado o processo de testes automatizados que ocorre na empresa objeto de estudo. O processo de criação de testes

automáticos e a realização de treinamentos longos começaram a ocorrer há aproximadamente um ano, tendo um enfoque especial nos novos colaboradores. Esse foco decorre devido a alta frequência recente, dentro desta janela de tempo do estudo, que novas turmas foram chamadas via concurso público. Diante disto, um viés de maior cobertura de código em novos colaboradores poderia ser explicado por estes receberem um programa de treinamento mais robusto, quando comparados com colaboradores mais antigos.

Verificando os dados, reparamos que são bem definidos os grupos de desenvolvedores que possuem alta porcentagem de cobertura, quando estes categorizados por ano. Podemos perceber que os desenvolvedores que possuem até 1 ano completo de empresa estão entre os que possuem maior média e maior concentração dos dados ao redor de 80% e 100%.

O testes de *Kruskal-Wallis* e Dunn, em conjunto, demonstram através de seus resultados que os únicos grupos que diferem em média dos outros, é exatamente o grupo de desenvolvedores que possui 1 ano de empresa, bem como o grupo que possui 6 anos de empresa. Para os demais grupos, não houve diferença significativa. A diferença de distribuição do grupo que possui até 1 ano de empresa fica evidente na Figura 8, destacando-se pela densidade muito acima dos demais na área entre 90 e 100

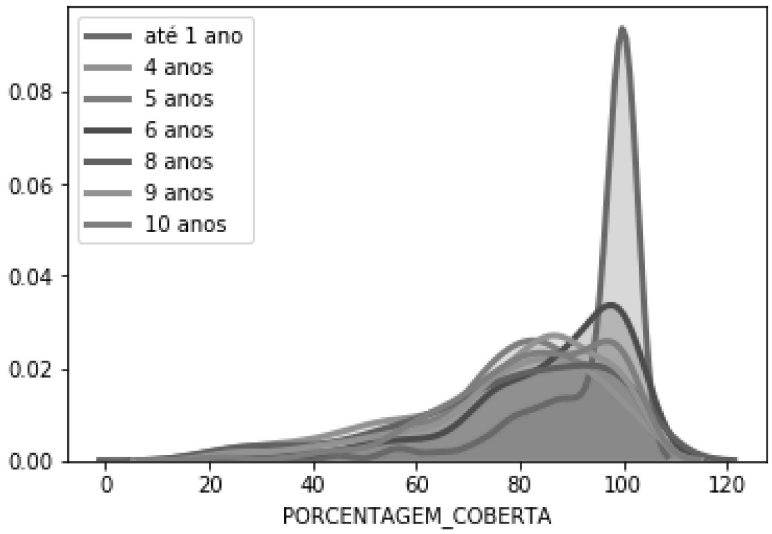

Figura 8. Historgrama de densidade das distribuições do percentual de cobertura, quando verificados os tempos de empresa.

Fonte: Elaborada pelo autor.

Isso demonstra que pode haver uma correlação causal entre o novo processo de capacitação dos funcionários, que inclui um curso de testes, e a alta cobertura de código produzida pelos desenvolvedores que passaram por ele.

Com relação à diferença entre as médias de cobertura, entre o grupo que possui 6 anos e os demais grupos, pode haver uma indicação que houve uma maior participação dos indivíduos deste nos cursos que ocorreram de forma eventual, ao longo dos últimos dois anos. Logo, seria adequado a realização de estudo onde houvesse variável indicativa de participaçãoem cursos de capacitação em testes automatizados. Desta forma, uma

melhor inferência sobre a influência dos resultados práticos dos treinamentos poderia ser verificada.

Os autores Ng e Feldmanb (2013) sugerem em seus estudos que, enquanto permanecer no mesmo emprego por um longo período de tempo pode diminuir a criatividade dos funcionários, uma vez que eles têm poucos estímulos em seus ambientes de trabalho, uma maior permanência no emprego pode aprimorar as habilidades dos funcionários para facilitar e implementar mudanças de maneira mais eficaz. Além disso, os autores que o tédio é uma das principais razões pelas quais os trabalhadores mais antigos não têm maior produtividade do que os trabalhadores que recém entraram. Eles sugerem que pequenas mudanças no contexto do trabalho, sem mudar o conteúdo, podem fazer sentido. Isso corrobora o achado do grupo com nível de função 1 (recém chegados à empresa), como o grupo com maior percentual de cobertura.

4.2.4. Função

A variável que representa o percentual coberto possui uma uma correlação negativa com a variável de nível de função, de aproximadamente 0,18. A grande maioria das evidências de cobertura de código foram criadas dentre os níveis de função que vão de 1 até 4. Possivelmente, devido o fato que conforme o nível atribuído ao usuário aumenta, menos contato com código este possui. Diante disto, foi realizado filtro para comparação apenas entre funções de menor grau (entre 1º e 4º grau de função).

Foi observado na plotagem da densidade da distribuição do percentual de cobertura, dentre os diferentes níveis de função, diferenças entre as distribuições de percentual de cobertura. Há uma tendência de maior dispersão dos dados quanto maior o nível da função. Possivelmente isso se deve ao fato que novos funcionários, ainda na função inicial, recebem o treinamento de execução de testes de software e cobertura de código ao longo dos primeiros meses de treinamento inicial na empresa. Já os funcionários mais antigos estão recebendo o treinamento formal sob demanda e em grupos menores, podendo assim apresentar maiores dificuldades na execução da tarefa.

Na utilização do teste estatístico *Kruskal-Wallis* foi verificado a existência de evidências de diferença entre as distribuições de cobertura, quando comparadas as funções que mais produzem dados de cobertura de código. Isto ocorre tanto quando são comparados todos os grupos, como quando são comparados em pares. Utilizando juntamente o teste de Qui-Quadrado, para teste de frequência das coberturas geradas por estes grupos, entre ERRO, ALERTA e OK, rejeitamos a hipótese nula de igualdade entre as distribuições. Isto nos leva a crer que a função possui relevância para determinar os grupos que precisam treinamentos dentro da empresa. Fato, que a função 1 e a função 3 se destacam pela mediana alta de porcentagem de código coberta em testes, inclusive não tendo diferença significativa entre elas, pelo teste U de *Wilcoxon-Mann-Whitney*.

No caso de cargo, assim como ocorreu em tempo de empresa, seria adequado entender quais cargos já tiveram algum treinamento em testes, visto que não há uma linearidade na correlação entre o nível da função do desenvolvedor e a porcentagem coberta. Possivelmente pode ter havido um foco maior dos treinamentos nos desenvolvedores de turmas que ingressaram há mais de 5 anos, de forma não intencional. Porém, é sabida-

mente conhecido que todos os desenvolvedores com menos de 1 ano de empresa receberam treinamento em testes automatizados.

De acordo com Dostie (2011) o grau de salário, equivalente ao nível da função para o presente estudo, não possui relação linear com a produtividade. Isso corrobora com a dificuldade encontrada em determinar uma correlação entre o nível de função e o percentual de cobertura de código atingido por cada um destes. Neste caso, a criação uma subdivisão dos grupos, através do previo conhecimento dos indivíduos que já passaram por algum tipo de treinamento, seria essencial para uma melhor comparação entre os grupos.

5. Considerações finais

O objetivo desse estudo de caso compreendeu entender quais as características cadastrais de desenvolvedores de software possuem maior relação com baixas taxas de cobertura de código. Para tal, foram aplicadas técnicas de inferência estatística a fim de verificar as características com maior relação com os percentuais de cobertura de código em testes automatizados. Foi verificado que os dados cadastrais dos desenvolvedores tendem a possuir um grau de relação com o percentual de cobertura de código, em especial a variável sexo e idade.

Como conclusão do trabalho, entende-se que os desenvolvedores do sexo masculino tendem a ter menor percentual de cobertura de código, enquanto mulheres tendem a ter uma menor variação de cobertura. Além disso, os grupos com maior idade e mais tempo na empresa compõem o grupo de programadores que necessitam um maior treinamento, devido à evidente diferença das distribuições e frequências, quando comparados com grupos de menor idade e menos tempo de empresa. Essas diferenças de frequência e distribuição foram observadas no uso de inferência estatística usando diferentes testes para verificação da unidade de análise, onde foram detectadas essas discrepâncias.

Neste estudo de caso, foi considerado que os desenvolvedores homens, da faixa etária de 30, de forma geral, necessitam de um maior treinamento, devido alto número de amostras com esse perfil, além da diferença nas médias de porcentagem quando comparados com os demais grupos estudados. Acredita-se que uma maior foco em treinamentos neste grupo, pode ajudar suprir as dificuldades em produzir melhores testes. Além disso, foi constatado que as mulheres tendem a ter um maior cuidado na produção de testes e códigos. Assim sendo, os resultados dessa pesquisa colabora para melhor conhecimento dos grupos que deveriam ser foco dos treinamentos, em especial os desenvolvedores do sexo masculino e da faixa etária de 30 anos.

Além da evidente diferença de distribuição das amostras, quando verificadas as variáveis idade e sexo, foi verificado que o treinamento tem sido especialmente eficiente para as turmas com menos tempo de empresa, visto que estas apresentam as maiores médias de cobertura de código, em relação aos demais grupos. Embora as evidências não apontem para importância do tempo de empresa para o percentual de cobertura de código, foi concluído que existem evidências iniciais que demonstram a importância de treinamentos em testes, nos treinamentos de capacitação de novos colaboradores, pois estão representadas nos percentuais de cobertura de código, especialmente nos colaboradores que esta situação é sabidamente conhecida. Isso, aliado à uma motivação extra dos trabalhadores mais novos, corroboraria também com o estudo de Ng e Feldmanb (2013)

sobre o tédio dos trabalhadores mais antigos.

Ao longo do estudo, houveram dificuldades e limitações práticas e teóricas. No que tange à parte teórica, existiu grande dificuldade em encontrar maior quantidade de bibliografias relacionadas ao trabalho, que levem em consideração as dificuldades do indivíduo em si, ao invés das complexidade das tarefas. Na parte prática, a maior limitação se deu pela falta de classificação de indivíduos que já haviam participado de algum treinamento de testes. Para um trabalho futuro, seria adequado que houvesse classificação dos treinamentos, de acordo com duração e tipo, para que no momento da análise dos grupos de indivíduos fosse conhecido o tipo de treinamento que cada um recebeu. Logo, seria adequado ter um estudo específico do impacto da efetividade dos treinamentos nas coberturas de código. Dessa forma, para trabalhos futuros sugere-se:

1. Verificar quais indivíduos receberam treinamento, identificando qual tipo de treinamento ocorreu, e comparar com os diferentes grupos de treinamento.
2. Comparar as amostras antes e depois de treinamentos, fazendo testes pareados para identificar a importância dos treinamentos em médias de cobertura de código.

Com o resultado deste trabalho, será possível aplicar identificar os grupos que mais necessitam treinamentos, permitindo otimização do processo de treinamento em testes de software para a empres, o que oportuniza sugerir a futura replicação desse estudo, no contexto de outras empresas intensivas em desenvolvimento de software.

Referências

Abdallah, S.A., Moawad,R. Fawzy, E.E. (2018) An optimization approach for automated unit test generation tools using multi-objective evolutionary algorithms, Future Computing and Informatics Journal, Volume 3, Issue 2,2018, Pages 178-190, ISSN 2314-7288

Agrawal, R., Srikant, R. (1994) *Fast algorithms for mining association rules in large databases.* Research Report RJ 9839 IBM Almaden Research Center, San Jose, CA, June .

Antinyan, V. , Derehag, J., Sandberg, A., Staron, M. (2018) *Mythical Unit Test Coverage*. IEEE Software, vol.35, no.3, pp. 73-79.

Bahamdain, S. (2015) Distributed software development in an offshore outsourcing project: A case study of source code evolution and quality.

Beller, M., Bholanath, R., McIntosh, S., Zaidman, A.(2016) *Analyzing the State of Static Analysis: A Large-Scale Evaluation in Open Source Software* IEEE 23rd *International Conference on Software Analysis, Evolution, and Reengineering* (SANER), Suita, 2016, pp. 470-481.

Balogh,G., Antal, G., Beszédes,Á., Vidács,L., Gyimóthy,T., Végh,Á. Z. (2015) *Identifying wasted effort in the field via developer interaction data*, *IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Bremen, 2015, pp. 391-400.

Dostie, B. W. (2011), *Productivity and Aging*. *The Economist* (0013-063X), [s. l.], v. 159, n. 2, p. 139–158, 2011.

Dunn, O.J. (1961) *Multiple comparisons among means*. JASA, 56: 54-64.

Emanuelsson, P., Nilsson, U. (2008). *A Comparative Study of Industrial Static Analysis Tools*. Electronic Notes in Theoretical Computer Science. 217.

Efe, P., Demirörs,O. (2013) *Applying EVM in a Software Company: Benefits and Difficulties*, 39th *Euromicro Conference on Software Engineering and Advanced Applications* Santander 2013, pp. 333-340.

Fagan, M.(1976) Design and code inspections to reduce errors in program development. IBM Systems Journal, 1976.

Fagan, M. E.. (2001) "Advances in Software Inspections". Pioneers and Their Contributions to Software Engineering. pp. 335–360. 2001.

Felbinger, H., Wotawa,F., Nica,M. (2018) *Adapting Unit Tests by Generating Combinatorial Test Data*. IEEE *International Conference on Software Testing*. Verification and Validation Workshops (ICSTW), Vasteras, 2018, pp. 352-355.

Fonseca, J. J. S. (2002) Metodologia da pesquisa científica. Fortaleza: UEC, 2002.

Grady, R. B., Slack, T. V . (1994) *Key lessons in achieving widespread inspection use in IEEE Software*, vol. 11, no. 4, pp. 46-57, July 1994.

Glenford J. Myers, Art of Software Testing, John Wiley Sons, Inc., New York, NY, 2004

Jalote,P. (2002) *An Inte grated Appr oac h to Softwar e Engineering*. Addison-W esley.

Johnson, P.M. (2008). *Reengineering Inspection*. Communications of the ACM.

Khan, R., Amjad, M. (2015) *Automatic test case generation for unit software testing using genetic algorithm and mutation analysis.* IEEE UP *Section Conference on Electrical Computer and Electronics* (UPCON), Allahabad, 2015, pp. 1-5

Lemkau, JP. (1979). *Personality and background characteristics of women in male-dominated occupations: a review*. Psychology of Women Quarterly 4(2):221-240

Mathur,S., Malik,S. (2010). *Advancements in the V-Model*. Int. J. Comput. Appl., vol. 1, no. 12, pp. 29–34, 2010.

Myers,G.J., Sandler,C., Badgett, T. (2011) The Art of Software Testing John Wiley Sons

Ng,T.W.H., Feldman, D.C. (2013) *Does longer job tenure help or hinder job performance?, Journal of Vocational Behavior*, Volume 83, Issue 3, 2013, Pages 305-314.

Naik, K.; Tripathy, P. (2008) *Software Testing and Quality Assurance: Theory and Practice*. Hoboken, N.J.: Wiley-Spektrum, 2008. Disponível em: ¡https://search.ebscohost.com/login.aspx?direct=truedb=nlebkAN=238501lang=pt-brsite=eds-live¿. Acesso em: 3 jun. 2019.

Paul, E. R. (1986). *Controlling software projects*, IEEE *Software Engineering Journal*, 1(1), 1986, pp. 7-16.

Raja, U., Tretter, M. J. (2012) *Defining and Evaluating a Measure of Open Source*

Project Survivability, IEEE *Transactions on Software Engineering*, vol. 38, no. 1, pp. 163-174, Jan.-Feb. 2012.

Sakamoto, K., Washizaki, H., Fukazawa,Y. (2010) "Open Code Coverage Framework: A Consistent and Flexible Framework for Measuring Test Coverage Supporting Multiple Programming Languages,"2010 10th International Conference on Quality Software, Zhangjiajie, 2010, pp. 262-269.

Triviños, A. N. S. (1987) Introdução à pesquisa em ciências sociais: a pesquisa qualitativa em educação. São Paulo: Atlas, .

Terrell, J. Kofink, A., Middleton, J., Rainear, C., Murphy-Hill, E., Parnin, C. (2016). *Gender bias in open source: Pull request acceptance of women versus men.*

Tengeri, D., Horváth,F., Beszédes,Á., Gergely,T., Gyimóthy,T.(2016) *Negative Effects of Bytecode Instrumentation on Java Source Code Coverage. IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering* (SANER), Suita, 2016, pp. 225-235.

Zahra,S., Nazir, A., Khalid, A., Raana, A., Majeed, M.N. (2014) *Performing Inquisitive Study of PM Traits Desirable for Project Progress*. International Journal of Modern Education and Computer Science. vol. 6, no. 2, pp. 41.

Zar, J.H. (2010). Biostatistical Analysis, 5th ed. Pearson Prentice Hall: Upper Saddle River, NJ. United States Department of Labor, Bureau of Labor Statistics.

Zhang, T., Lee, B. (2012) *How to Recommend Appropriate Developers for Bug Fixing? IEEE 36th Annual Computer Software and Applications Conference*, Izmir, 2012, pp. 170-175.