Testes unitários são efetivos contra Bugs?

Anderson B. Coutinho[1], Henrique A. N. A. Ramos[1], Humberto T. M. Neto[1]
José L. P. X. Junior[1], Mateus S. Fonseca[1], Yan M. R. S. Pinheiro[1]

[1]Instituto de Ciências Exatas e Informática – PUC Minas – Belo Horizonte, MG

{mateus.fonseca.1092523, anderson.coutinho, hanaramos, yan.max}@sga.pucminas.br

Abstract. *With the concept of unit tests on the rise, the study of its impact on software becomes very important. However, although there are discussions on this topic, there is still a lack of in-depth studies in this area. In this way, the present work offers a study that relates the lines of unit test code and coverage with BUG Issues from the most popular repositories of several GitHub languages, seeking results that prove the effectiveness of unit tests.*

Resumo. *Com o conceito de testes unitários em alta, o estudo do seu impacto nos softwares se torna muito importante. Porém, embora haja discussões a respeito deste tema, ainda há a carência de estudos aprofundados nesta área. Desta forma, o presente trabalho oferece um estudo que relaciona as linhas de código de teste unitário e coverage com BUG Issues dos repositórios mais populares de diversas linguagens do GitHub, buscando resultados que comprovem, justamente, a eficácia destes testes unitários.*

1. Introdução

Testes em software é um conceito comum nos dias de hoje. Seja feito por um time especializado ou pelos próprios desenvolvedores, verificar que o sistema funciona da maneira esperada é importante para que este atinja as expectativas dos usuários finais, sendo o produto final funcional e o mais resiliente possível. Na pirâmide de testes, os unitários estão na base, por serem os de menor granularidade e mais rápidos. Eles são, muitas vezes, feitos antes mesmo de desenvolver a nova funcionalidade, o chamado *Test-driven development*.

Portanto, tendo conhecimento da importância deste tema, foi analisada a necessidade de se verificar a relação entre testes unitários e o surgimento de Bugs em repositórios populares de linguagens diversas, uma vez que, nas pesquisas em bases bibliográficas, não foram encontrados estudos profundos neste específico assunto. Para isto, foram selecionadas linguagens frequentemente utilizadas no GitHub e que, em seus repositórios, possuíam uma quantidade considerável de dados importantes sobre testes unitários.

Para atingir os objetivos desta pesquisa, temos bem definida a a hipótese nula desta pesquisa: Testes unitários impactam nos BUG Issues dos repositórios e, como hipótese alternativa, sua negação: Testes unitários não impactam nos BUG Issues dos repositórios. Estas hipóteses são fundamentais para atingir o Goal dessa pesquisa, que é verificar a efetividade de testes unitários em relação à bugs nos repositórios do GitHub.

Finalmente, tendo como base as hipóteses e o Goal, foram definidas questões que devem ser respondidas através dos resultados finais, sendo a primeira: **Qual a relação**

entre a quantidade de testes unitários e a quantidade de bugs? A resposta dessa pergunta é encontrada através do cálculo da primeira métrica da pesquisa: **Quantidade de BUG Issues/Linhas de código de teste unitário**. Já a segunda pergunta desta pesquisa é: **Qual a relacão entre o coverage do código e a quantidade de bugs?** Esta última pergunta é respondida através do cálculo da segunda métrica: **Quantidade de BUG Issues/Coverage do repositório**.

Figura 1. Goal, Questions, Metrics

2. Planejamento

Concepção: Verificar a efetividade de testes unitários em relação à bugs nos repositórios do GitHub;

Design: Estamos medindo efetividade de testes unitários em relação à bugs, que é um erro ou falha em um programa ou sistema de computador que faz com que produza um resultado incorreto ou inesperado ou se comporte de maneiras não intencionais. Tendo isso em vista, iremos analisar se quanto mais testes unitários o código contiver, menor será o número de bugs;

Preparação: Para desenvolver este projeto, será necessário estudar a linguagem Python bem como a API GraphQL do GitHub, para recuperar as informações necessárias de forma automatizada. Além disso, é importante estudar as badges do GitHub, especialmente a que será utilizada (coverage), visando melhor utilização desse dado;

Execução: Executaremos um script Python que fará consultas na API do GitHub, tendo assim as informações dos repositórios e realizando o download deles para que obtenhamos o número de linhas de código de teste, que também é importante dentro do nosso conjunto de dados. A partir dessas informações obtidas, vamos calcular as nossas métricas, assim como a média, mediana e coeficiente de Pearson;

Análise: Primeiramente, vamos revisar todas as medidas tomadas para garantir que sejam válidas e úteis. Então, as medidas serão organizadas em conjuntos de dados que serão examinados como parte do processo de teste de hipóteses. Por fim, esses dados serão analisados com base no coeficiente de Pearson que será utilizado para descobrir se há uma relação linear entre as variáveis;

Disseminação e tomada de decisão: Vamos ter todo o processo documentado, com os dados e resultados obtidos apresentados de forma transparente, para que o experimento possa ser replicado em um cenário semelhante por outros profissionais que queiram confirmar nossas conclusões. Os resultados podem ser utilizados para que os desenvolvedores possam repensar na maneira com que eles lidam com os testes nos seus softwares desenvolvidos, melhorando a manutenibilidade do código.

3. 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 na análise do arquivo .csv gerado. Abaixo temos um exemplo inicial de como foi o fluxo da execução do projeto.

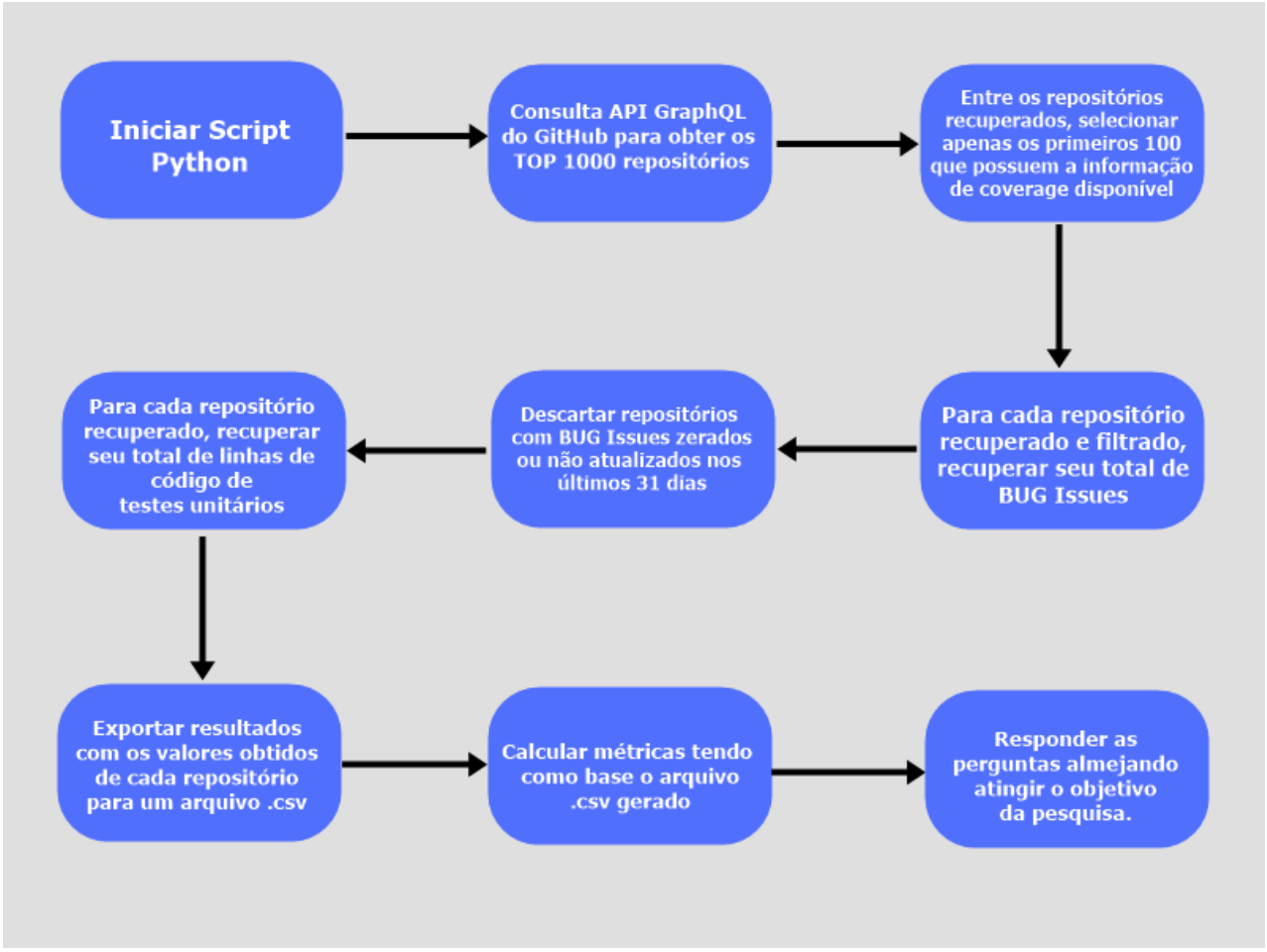

Figura 2. Overview da metodologia do projeto.

Na primeira etapa, tendo em vista que o Goal desta pesquisa depende de uma análise de repositórios GitHub, foram escolhidas *JavaScript*, *Go*, *C#*, *Java* e *Python*. Foram escolhidas em razão de sua popularidade e, por isso, seriam capazes de atender ao mínimo de 10 repositórios retornados levando em conta os filtros desta pesquisa. Esta quantidade mínima de 10 repositórios é necessária para garantir que existem dados suficientes por linguagem para realizar uma análise mais abrangente. Para cada linguagem, foi executado um script em *Python* que retorna os 1000 repositórios mais famosos, em seguida, é consultada a sua porcentagem de cobertura de testes [1], caso não retorne sua cobertura, o repositório é descartado.

Em seguida, para cada um dos repositórios não descartados, uma outra consulta foi realizada a API GraphQL do GitHub, esta retorna suas Issues cujas labels correspondam a BUG. Tendo em vista que as labels das Issues são customizáveis, para este projeto foram considerados como BUG issues aquelas cujas strings das labels sejam iguais a:

bug	***type: bug***	***browser bug***
error	***failure***	***confirmed bug***
type: bug/error	***good first bug***	***confirmed-bug***
bug critical	***bug minor***	***bug moderate***

Tabela 1. *Labels* consideram também maiúsculas e minúsculas.

Nesta consulta, são retornados a quantidade total e a última data de atualização dos *bugs*. Para garantir que são considerados apenas repositórios em que issues ainda são ativamente abertas, foram mantidos apenas aqueles que foram atualizados a pelo menos 31 dias. Foram contabilizados até 100 repositórios.

Posterior a filtragem das BUG Issues, foram recuperadas as informações de linhas de código de testes unitários de cada repositório. Para isso, foi realizado o *download* de cada item através do *GitPython* e, posteriormente, outro *script* realizará a contagem, analizando arquivos cujo nome contenha *test*, *tests* e, no caso de JavaScript: *spec.js* e *karma.js*.

Por fim, foram realizadas análises estatísticas acerca dos repositórios. Para cada um, foi montado uma correlação linear em gráfico para as duas métricas da pesquisa, além de armazenar o valor de intercepto, P e R. É importante ressaltar que foram produzidos dois gráficos para cada linguagem e este processo foi repetido para as 5 selecionadas.

[1] A porcentagem de cobertura de teste dos repositórios é buscada no site `https://coveralls.io/` de forma automática.

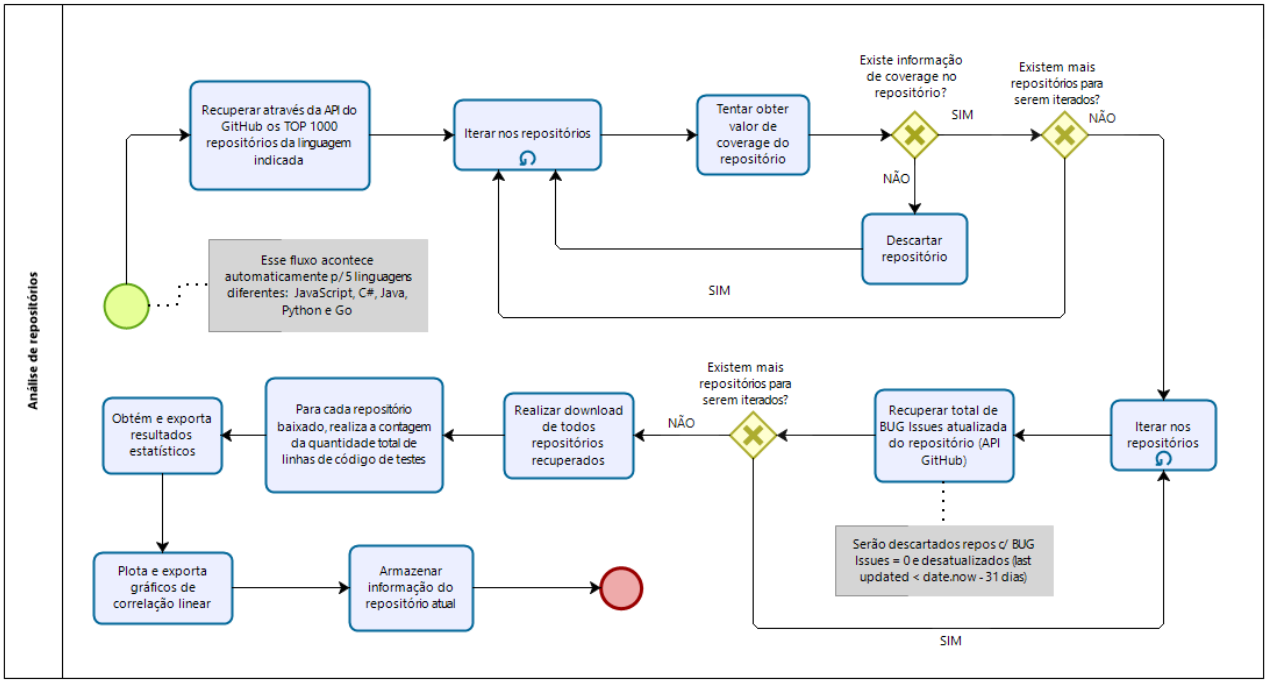

Figura 3. Exemplo unificado do processo realizado pelos scripts que compõem a solução.

4. Resultados

Todos os dados contidos nesta seção foram obtidos em 23 de Maio de 2020, podendo sofrer alterações em períodos futuros.

4.1. C#

Nome	Autor	Cobertura	Bug Issues	Último	LOC de teste	URL
Avalonia	AvaloniaUI	70%	647	19/05/20	68183	https://github.com/AvaloniaUI/Avalonia
choco	chocolatey	20%	436	19/05/20	1282	https://github.com/chocolatey/choco
ReactiveUI	reactiveui	69%	189	23/05/20	14574	https://github.com/reactiveui/ReactiveUI
duplicati	duplicati	91%	1070	18/05/20	5848	https://github.com/duplicati/duplicati
Ocelot	ThreeMammals	88%	85	22/05/20	12796	https://github.com/ThreeMammals/Ocelot
NLog	NLog	45%	371	18/05/20	7582	https://github.com/NLog/NLog
Lean	QuantConnect	89%	476	20/05/20	127941	https://github.com/QuantConnect/Lean
cake	cake-build	74%	201	19/05/20	78100	https://github.com/cake-build/cake
Wexflow	aelassas	68%	17	23/05/20	802	https://github.com/aelassas/Wexflow
AsyncEx	StephenCleary	65%	4	17/05/20	3756	https://github.com/StephenCleary/AsyncEx
markdig	lunet-io	94%	60	30/04/20	4275	https://github.com/lunet-io/markdig
SimpleInjector	simpleinjector	96%	93	22/05/20	45332	https://github.com/simpleinjector/SimpleInjector
piranha.core	PiranhaCMS	75%	147	22/05/20	6586	https://github.com/PiranhaCMS/piranha.core
Saml2	Sustainsys	74%	133	19/05/20	23291	https://github.com/Sustainsys/Saml2

Tabela 2. Repositórios *C#* analisados

Para o conjunto de todos repositórios na linguagem C# analisados, chegamos nos seguintes valores estatísticos:

Para X sendo LOCs de Teste e Y sendo BUG Issues.

- Inclinação: 0.0016157447748580767
- Intercepto: 234.43855791965137

- Erro padrão: 0.002208686337422108
- Valor P: 0.4784916994757268
- Valor R: 0.2066207374367496

Interpretando esses valores a partir do Coeficiente de Pearson, temos que o valor R obtido é igual a 0.2066207374367496. Ou seja a correlação é desprezível.

Para X sendo Coverage e Y sendo BUG Issues

- Inclinação: -0.5484794834409498
- Intercepto: 320.52515101020623
- Erro padrão: 4.176913794552045
- Valor P: 0.8977042251486562
- Valor R: -0.03787934453847704

Interpretando esses valores a partir do Coeficiente de Pearson, temos que o valor R obtido é igual a -0.03787934453847704. Ou seja a correlação é desprezível.

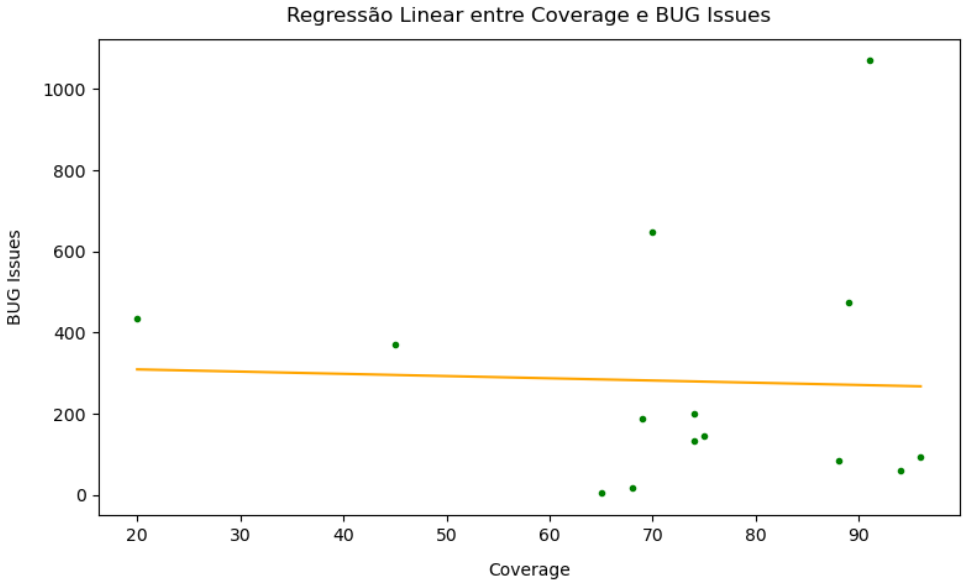

Figura 4. Gráfico de correlação entre cobertura e *bugs* dos repositórios C#

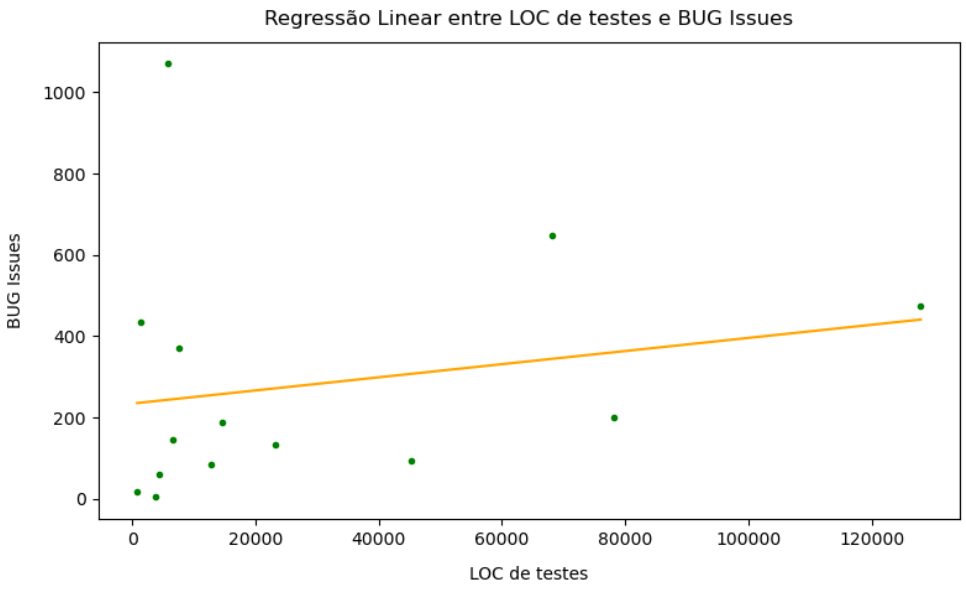

Figura 5. Gráfico de correlação entre LOC de testes e *bugs* dos repositórios C#

Na Figura 4 e Figura 5, vemos a correlação entre BUG Issues e Coverage dos repositórios e correlação entre BUG Issues e LOC de teste, respectivamente, da linguagem C#, com uma correlação não linear.

4.2. Go

Nome	Autor	Cobertura	Bug Issues	Último	LOC de teste	URL
awesome-go	avelino	96%	8	18/05/20	151	https://github.com/avelino/awesome-go
gin	gin-gonic	21%	50	20/05/20	5832	https://github.com/gin-gonic/gin
go-ethereum	ethereum	44%	146	13/05/20	75539	https://github.com/ethereum/go-ethereum
nsq	nsqio	61%	84	27/04/20	6293	https://github.com/nsqio/nsq
echo	labstack	82%	109	02/05/20	5806	https://github.com/labstack/echo
helm	helm	66%	504	22/05/20	22424	https://github.com/helm/helm
go-micro	micro	58%	10	07/05/20	11426	https://github.com/micro/go-micro
grpc-go	grpc	92%	238	21/05/20	57352	https://github.com/grpc/grpc-go
jaeger	jaegertracing	100%	70	19/05/20	34557	https://github.com/jaegertracing/jaeger
fyne	fyne-io	66%	131	22/05/20	11353	https://github.com/fyne-io/fyne
hydra	ory	52%	171	04/05/20	12213	https://github.com/ory/hydra
casbin	casbin	75%	37	20/05/20	3792	https://github.com/casbin/casbin
bleve	blevesearch	66%	79	23/05/20	25000	https://github.com/blevesearch/bleve
weave	weaveworks	66%	665	22/05/20	8695	https://github.com/weaveworks/weave
goreleaser	goreleaser	84%	184	22/05/20	14208	https://github.com/goreleaser/goreleaser
validator	go-playground	100%	22	29/04/20	1099	https://github.com/go-playground/validator
go-swagger	go-swagger	72%	375	13/05/20	40172	https://github.com/go-swagger/go-swagger
rpcx	smallnest	19%	32	19/05/20	879	https://github.com/smallnest/rpcx
lnd	lightningnetwork	63%	163	14/05/20	131597	https://github.com/lightningnetwork/lnd
gqlgen	99designs	66%	78	07/05/20	7478	https://github.com/99designs/gqlgen
git-bug	MichaelMure	3%	23	22/05/20	4442	https://github.com/MichaelMure/git-bug
govalidator	asaskevich	92%	12	12/05/20	789	https://github.com/asaskevich/govalidator
quic-go	lucas-clemente	59%	386	23/05/20	34759	https://github.com/lucas-clemente/quic-go
bigcache	allegro	92%	12	09/05/20	2404	https://github.com/allegro/bigcache
sqlflow	sql-machine-learning	63%	115	22/05/20	11105	https://github.com/sql-machine-learning/sqlflow
cadence	uber	69%	107	12/05/20	106191	https://github.com/uber/cadence
go-socket.io	googollee	52%	60	23/05/20	592	https://github.com/googollee/go-socket.io
go-astilectron	asticode	79%	14	19/05/20	1343	https://github.com/asticode/go-astilectron
statping	statping	48%	38	22/05/20	3937	https://github.com/statping/statping
argo-cd	argoproj	27%	684	23/05/20	21668	https://github.com/argoproj/argo-cd
nats.go	nats-io	92%	11	13/05/20	13822	https://github.com/nats-io/nats.go
yaegi	containous	72%	240	21/05/20	1892	https://github.com/containous/yaegi
emitter	emitter-io	90%	10	17/05/20	7809	https://github.com/emitter-io/emitter
cameradar	Ullaakut	100%	33	06/05/20	2009	https://github.com/Ullaakut/cameradar
mercure	dunglas	95%	4	08/05/20	2979	https://github.com/dunglas/mercure
oathkeeper	ory	65%	20	07/05/20	8775	https://github.com/ory/oathkeeper

Tabela 3. Repositórios *Go* analisados

Para o conjunto de todos repositórios na linguagem Go analisados, chegamos nos seguintes valores estatísticos:

Para X sendo LOCs de Teste e Y sendo BUG Issues.

- Inclinação: 0.0012130004038066334
- Intercepto: 113.20656531058617
- Erro padrão: 0.0009992690542046869
- Valor P: 0.23315558436204337
- Valor R: 0.20381040139113313

Interpretando esses valores a partir do Coeficiente de Pearson, temos que o valor R obtido é igual a 0.20381040139113313. Ou seja a correlação é desprezível.

Para X sendo Coverage e Y sendo BUG Issues

- Inclinação: -1.53641596394797
- Intercepto: 241.23916288279673
- Erro padrão: 1.2312748994742286
- Valor P: 0.2206209171379465
- Valor R: -0.2092622130809256

Interpretando esses valores a partir do Coeficiente de Pearson, temos que o valor R obtido é igual a -0.2092622130809256. Ou seja a correlação é desprezível.

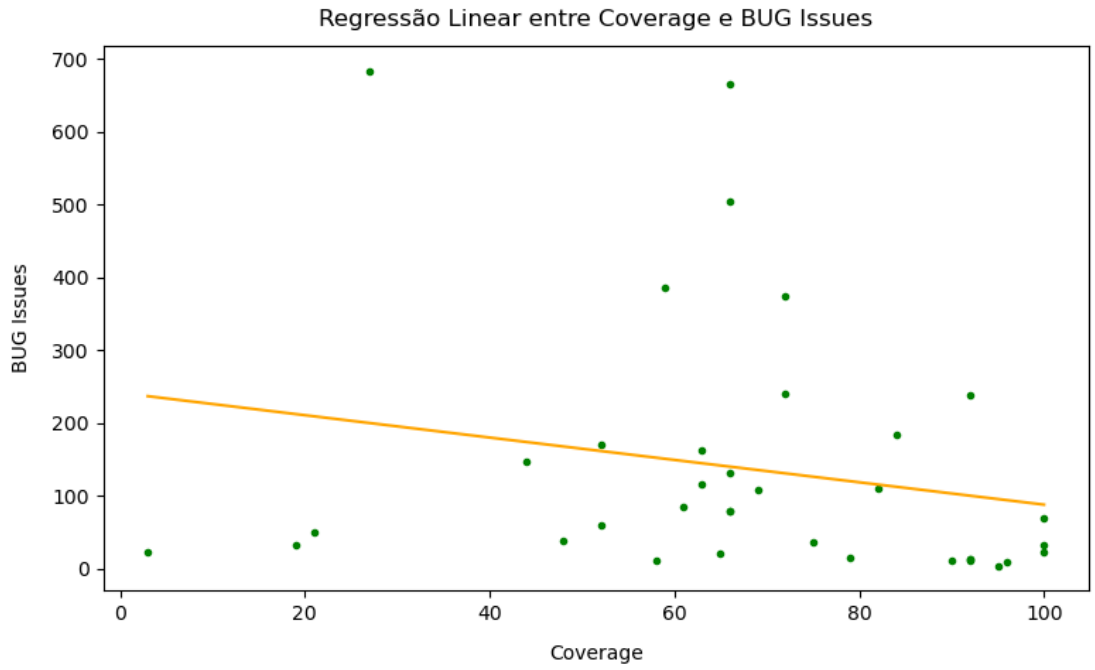

Figura 6. Gráfico de correlação entre cobertura e *bugs* dos repositórios Go

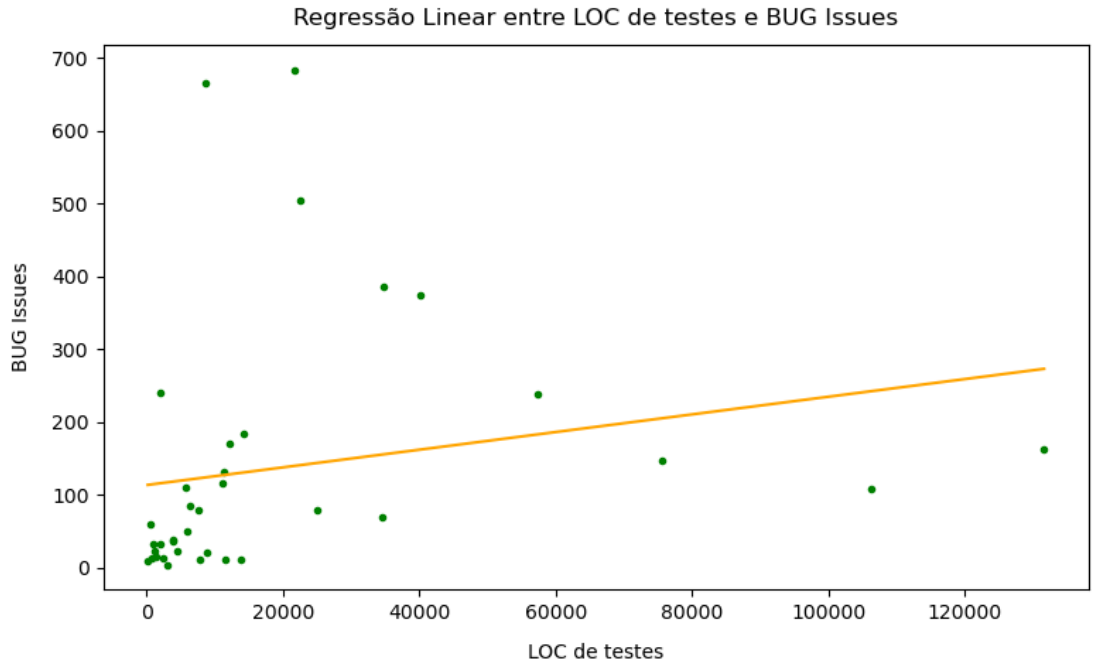

Figura 7. Gráfico de correlação entre LOC de testes e *bugs* dos repositórios Go

Na Figura 6 e Figura 7, vemos a correlação entre BUG Issues e Coverage dos

repositórios e correlação entre BUG Issues e LOC de teste, respectivamente, da linguagem Go, com uma correlação não linear.

4.3. Java

Nome	Autor	Cobertura	Bug Issues Último	LOC de teste	URL	
java-design-patterns	iluwatar	83%	71	19/05/20	29324	https://github.com/iluwatar/java-design-patterns
glide	bumptech	75%	409	11/05/20	37125	https://github.com/bumptech/glide
jadx	skylot	73%	173	22/05/20	27487	https://github.com/skylot/jadx
mybatis-3	mybatis	87%	119	19/05/20	37151	https://github.com/mybatis/mybatis-3
skywalking	apache	31%	361	23/05/20	37937	https://github.com/apache/skywalking
HikariCP	brettwooldridge	76%	106	08/05/20	7165	https://github.com/brettwooldridge/HikariCP
shardingsphere	apache	56%	457	22/05/20	2261	https://github.com/apache/shardingsphere
rocketmq	apache	51%	65	13/05/20	25663	https://github.com/apache/rocketmq
mockito	mockito	86%	73	20/05/20	42680	https://github.com/mockito/mockito
spring-boot-admin	codecentric	91%	163	19/05/20	12837	https://github.com/codecentric/spring-boot-admin
android-gif-drawable	koral–	8%	120	23/04/20	605	https://github.com/koral--/android-gif-drawable
grpc-java	grpc	88%	256	21/05/20	113254	https://github.com/grpc/grpc-java
OpenRefine	OpenRefine	46%	746	22/05/20	32432	https://github.com/OpenRefine/OpenRefine
hsweb-framework	hs-web	6%	34	15/05/20	1183	https://github.com/hs-web/hsweb-framework
resilience4j	resilience4j	100%	29	14/05/20	40554	https://github.com/resilience4j/resilience4j
checkstyle	checkstyle	72%	366	17/05/20	88974	https://github.com/checkstyle/checkstyle
springfox	springfox	97%	414	22/05/20	2316	https://github.com/springfox/springfox
orientdb	orientechnologies	45%	1721	20/05/20	227526	https://github.com/orientechnologies/orientdb
cryptomator	cryptomator	73%	264	22/05/20	1222	https://github.com/cryptomator/cryptomator
junit5	junit-team	57%	192	22/05/20	81494	https://github.com/junit-team/junit5
byte-buddy	raphw	85%	88	07/05/20	103140	https://github.com/raphw/byte-buddy
failsafe	jhalterman	93%	18	12/05/20	6202	https://github.com/jhalterman/failsafe
opengrok	oracle	73%	861	19/05/20	33510	https://github.com/oracle/opengrok
NullAway	uber	92%	12	06/05/20	3873	https://github.com/uber/NullAway
crate	crate	20%	108	29/04/20	163735	https://github.com/crate/crate
graphhopper	graphhopper	38%	233	22/05/20	46678	https://github.com/graphhopper/graphhopper
conductor	Netflix	69%	41	21/05/20	42032	https://github.com/Netflix/conductor

Tabela 4. Repositórios *Java* analisados

Para o conjunto de todos repositórios na linguagem Java analisados, chegamos nos seguintes valores estatísticos:

Para X sendo LOCs de Teste e Y sendo BUG Issues.

- Inclinação: 0.00353553376592266
- Intercepto: 114.31040992491808
- Erro padrão: 0.001131745221933014
- Valor P: 0.0044741706346982275
- Valor R: 0.5298728350321273

Interpretando esses valores a partir do Coeficiente de Pearson, temos que o valor R obtido é igual a 0.5298728350321273. Ou seja a correlação é moderada.

Para X sendo Coverage e Y sendo BUG Issues

- Inclinação: -2.1829019743624047
- Intercepto: 420.959977651697
- Erro padrão: 2.631509655898326
- Valor P: 0.4146585004233543
- Valor R: -0.16366777588470346

Interpretando esses valores a partir do Coeficiente de Pearson, temos que o valor R obtido é igual a -0.16366777588470346. Ou seja a correlação é desprezível.

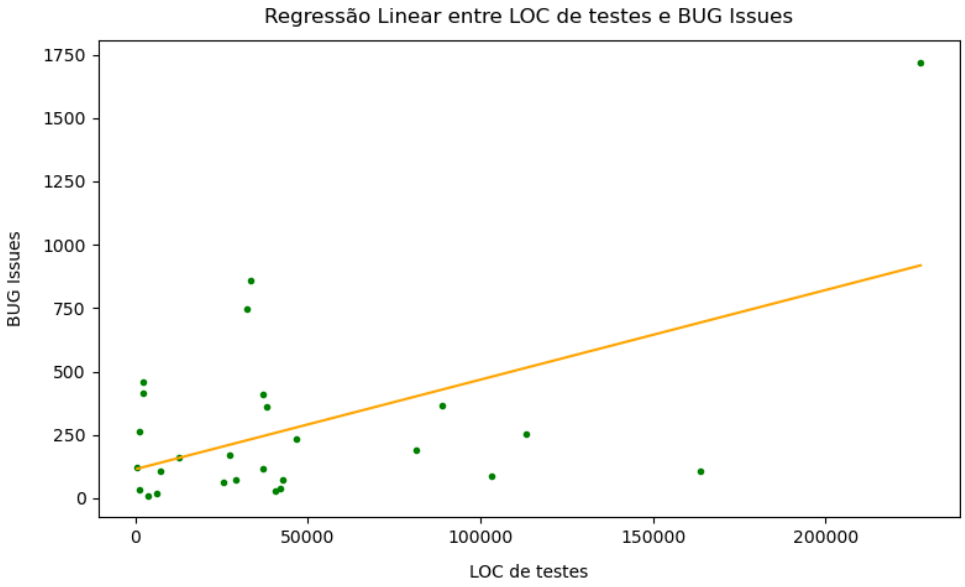

Figura 9. Gráfico de correlação entre LOC de testes e *bugs* dos repositórios Java

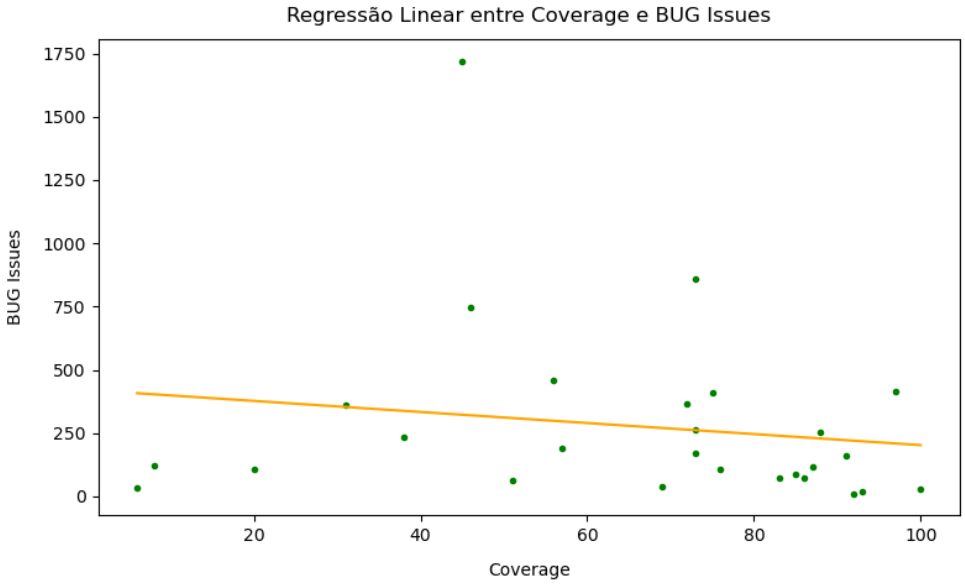

Figura 8. Gráfico de correlação entre cobertura e *bugs* dos repositórios Java

Na Figura 8 e Figura 9, vemos a correlação entre BUG Issues e Coverage dos repositórios e correlação entre BUG Issues e LOC de teste, respectivamente, da linguagem Java, com uma correlação não linear.

4.4. JavaScript

Nome	Autor	Cobertura	Bug Issues	Último	LOC de teste	URL
vue	vuejs	96%	363	01/05/20	29686	https://github.com/vuejs/vue
react	facebook	86%	570	22/05/20	135321	https://github.com/facebook/react
bootstrap	twbs	95%	219	15/05/20	5329	https://github.com/twbs/bootstrap
react-native	facebook	9%	3125	23/05/20	84747	https://github.com/facebook/react-native
webpack	webpack	95%	818	17/05/20	19965	https://github.com/webpack/webpack
Chart.js	chartjs	92%	1232	22/05/20	18563	https://github.com/chartjs/Chart.js
next.js	zeit	52%	303	22/05/20	33581	https://github.com/zeit/next.js
Semantic-UI	Semantic-Org	89%	952	19/05/20	311	https://github.com/Semantic-Org/Semantic-UI
lodash	lodash	33%	240	12/05/20	4288	https://github.com/lodash/lodash
moment	moment	88%	251	20/05/20	92	https://github.com/moment/moment
serverless	serverless	91%	1048	22/05/20	56553	https://github.com/serverless/serverless
Ghost	TryGhost	44%	1530	22/05/20	60322	https://github.com/TryGhost/Ghost
hexo	hexojs	98%	247	20/05/20	45	https://github.com/hexojs/hexo
mermaid	mermaid-js	46%	155	22/05/20	8637	https://github.com/mermaid-js/mermaid
video.js	videojs	86%	344	15/05/20	19696	https://github.com/videojs/video.js
Rocket.Chat	RocketChat	57%	1959	22/05/20	4105	https://github.com/RocketChat/Rocket.Chat
preact	preactjs	100%	163	21/05/20	26264	https://github.com/preactjs/preact
underscore	jashkenas	97%	103	09/05/20	0	https://github.com/jashkenas/underscore
mobx	mobxjs	94%	84	20/05/20	6379	https://github.com/mobxjs/mobx
sails	balderdashy	99%	156	03/05/20	6219	https://github.com/balderdashy/sails
wepy	Tencent	39%	58	05/05/20	2472	https://github.com/Tencent/wepy
react-select	JedWatson	70%	50	21/05/20	5275	https://github.com/JedWatson/react-select
lighthouse	GoogleChrome	90%	348	20/05/20	43154	https://github.com/GoogleChrome/lighthouse
mocha	mochajs	93%	244	20/05/20	15910	https://github.com/mochajs/mocha
pug	pugjs	77%	118	16/05/20	77330	https://github.com/pugjs/pug
react-virtualized	bvaughn	100%	85	16/05/20	178	https://github.com/bvaughn/react-virtualized
scrollreveal	jlmakes	30%	73	02/05/20	551	https://github.com/jlmakes/scrollreveal
ava	avajs	87%	218	19/05/20	3290	https://github.com/avajs/ava
parse-server	parse-community	9%	169	16/05/20	76645	https://github.com/parse-community/parse-server
sharp	lovell	100%	125	17/05/20	0	https://github.com/lovell/sharp
eslint	eslint	69%	2147	23/05/20	3713	https://github.com/eslint/eslint
graphql-js	graphql	98%	30	22/05/20	40720	https://github.com/graphql/graphql-js
pkg	zeit	9%	15	06/05/20	4369	https://github.com/zeit/pkg
bower	bower	85%	163	27/04/20	1740	https://github.com/bower/bower
chalk	chalk	100%	20	27/04/20	157	https://github.com/chalk/chalk
fastify	fastify	99%	101	20/05/20	17197	https://github.com/fastify/fastify
riot	riot	100%	371	16/05/20	958	https://github.com/riot/riot
vant	youzan	0%	285	23/05/20	13399	https://github.com/youzan/vant
handsontable	handsontable	67%	1966	22/05/20	18426	https://github.com/handsontable/handsontable
loopback	strongloop	90%	345	22/05/20	17002	https://github.com/strongloop/loopback
Inquirer.js	SBoudrias	92%	50	19/05/20	265	https://github.com/SBoudrias/Inquirer.js
feathers	feathersjs	19%	38	29/04/20	11106	https://github.com/feathersjs/feathers
kitematic	docker	67%	272	17/05/20	82	https://github.com/docker/kitematic
appium	appium	89%	1327	20/05/20	3842	https://github.com/appium/appium
sweetalert2	sweetalert2	92%	65	20/05/20	3549	https://github.com/sweetalert2/sweetalert2
NodeBB	NodeBB	90%	2218	22/05/20	43	https://github.com/NodeBB/NodeBB
shields	badges	96%	240	12/05/20	7540	https://github.com/badges/shields
nightwatch	nightwatchjs	85%	162	19/05/20	21339	https://github.com/nightwatchjs/nightwatch
mathjs	josdejong	49%	154	20/05/20	45437	https://github.com/josdejong/mathjs
node-restify	restify	33%	107	06/05/20	13024	https://github.com/restify/node-restify
summernote	summernote	81%	254	23/05/20	3323	https://github.com/summernote/summernote
luxon	moment	96%	2	28/04/20	6622	https://github.com/moment/luxon
workbox	GoogleChrome	79%	108	15/05/20	15512	https://github.com/GoogleChrome/workbox
web3.js	ethereum	90%	333	23/05/20	6116	https://github.com/ethereum/web3.js
uncss	uncss	98%	11	19/05/20	47	https://github.com/uncss/uncss
debug	visionmedia	88%	23	19/05/20	121	https://github.com/visionmedia/debug
truffle	trufflesuite	73%	120	20/05/20	5427	https://github.com/trufflesuite/truffle
laverna	Laverna	95%	140	27/04/20	112	https://github.com/Laverna/laverna
bull	OptimalBits	94%	68	23/05/20	6404	https://github.com/OptimalBits/bull
openzeppelin-contracts	OpenZeppelin	100%	32	27/04/20	5489	https://github.com/OpenZeppelin/openzeppelin-contracts
stylelint	stylelint	97%	480	23/05/20	9570	https://github.com/stylelint/stylelint
hls.js	video-dev	90%	237	19/05/20	355	https://github.com/video-dev/hls.js
yargs	yargs	100%	101	11/05/20	10	https://github.com/yargs/yargs
deck.gl	visgl	83%	404	22/05/20	23232	https://github.com/visgl/deck.gl
ajv	ajv-validator	97%	44	24/04/20	8094	https://github.com/ajv-validator/ajv
react-vis	uber	93%	74	22/05/20	5791	https://github.com/uber/react-vis
openlayers	openlayers	85%	21	22/05/20	60498	https://github.com/openlayers/openlayers
react-dropzone	react-dropzone	100%	26	20/05/20	2946	https://github.com/react-dropzone/react-dropzone
ts-node	TypeStrong	82%	101	16/05/20	3	https://github.com/TypeStrong/ts-node
zeroclipboard	zeroclipboard	62%	91	01/05/20	4607	https://github.com/zeroclipboard/zeroclipboard
eslint-plugin-react	yannickcr	98%	468	15/05/20	0	https://github.com/yannickcr/eslint-plugin-react
aws-sdk-js	aws	92%	91	22/05/20	13638	https://github.com/aws/aws-sdk-js
vue-i18n	kazupon	90%	42	12/05/20	0	https://github.com/kazupon/vue-i18n
react-modal	reactjs	87%	59	12/05/20	1172	https://github.com/reactjs/react-modal
pino	pinojs	100%	15	07/05/20	6096	https://github.com/pinojs/pino
npm-check-updates	raineorshine	90%	97	18/05/20	1601	https://github.com/raineorshine/npm-check-updates
node-crawler	bda-research	90%	19	29/04/20	1631	https://github.com/bda-research/node-crawler
ui-grid	angular-ui	85%	371	13/05/20	20677	https://github.com/angular-ui/ui-grid
turf	Turfjs	92%	161	07/05/20	8930	https://github.com/Turfjs/turf

Tabela 5. Repositórios *JavaScript* analisados

Para o conjunto de todos repositórios na linguagem JavaScript analisados, chegamos nos seguintes valores estatísticos:

Para X sendo LOCs de Teste e Y sendo BUG Issues.

- Inclinação: 0.0065812278533104625
- Intercepto: 273.52070296932925
- Erro padrão: 0.002744256052781096
- Valor P: 0.018897882474565437
- Valor R: 0.2636300629844727

Interpretando esses valores a partir do Coeficiente de Pearson, temos que o valor R obtido é igual a 0.2636300629844727. Ou seja a correlação é desprezível.

Para X sendo Coverage e Y sendo BUG Issues

- Inclinação: -1.53641596394797
- Intercepto: 778.8376482818458
- Erro padrão: 2.589834314964686
- Valor P: 0.051869115260535656
- Valor R: -0.21956414084848558

Interpretando esses valores a partir do Coeficiente de Pearson, temos que o valor R obtido é igual a -0.21956414084848558. Ou seja a correlação é desprezível.

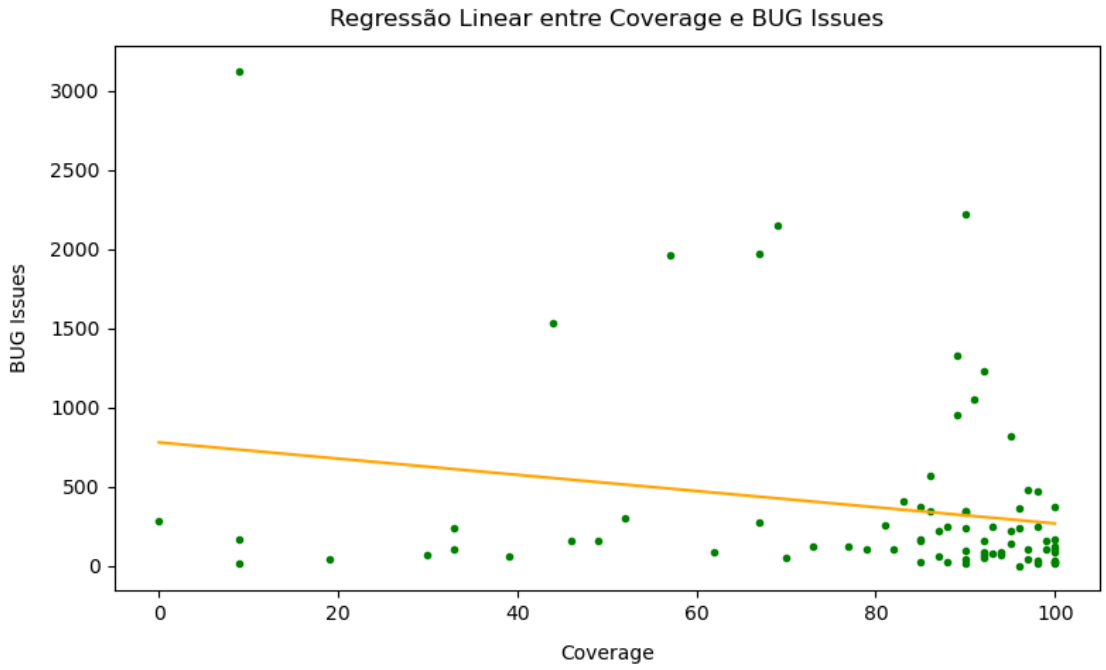

Figura 10. Gráfico de correlação entre cobertura e *bugs* dos repositórios JavaScript

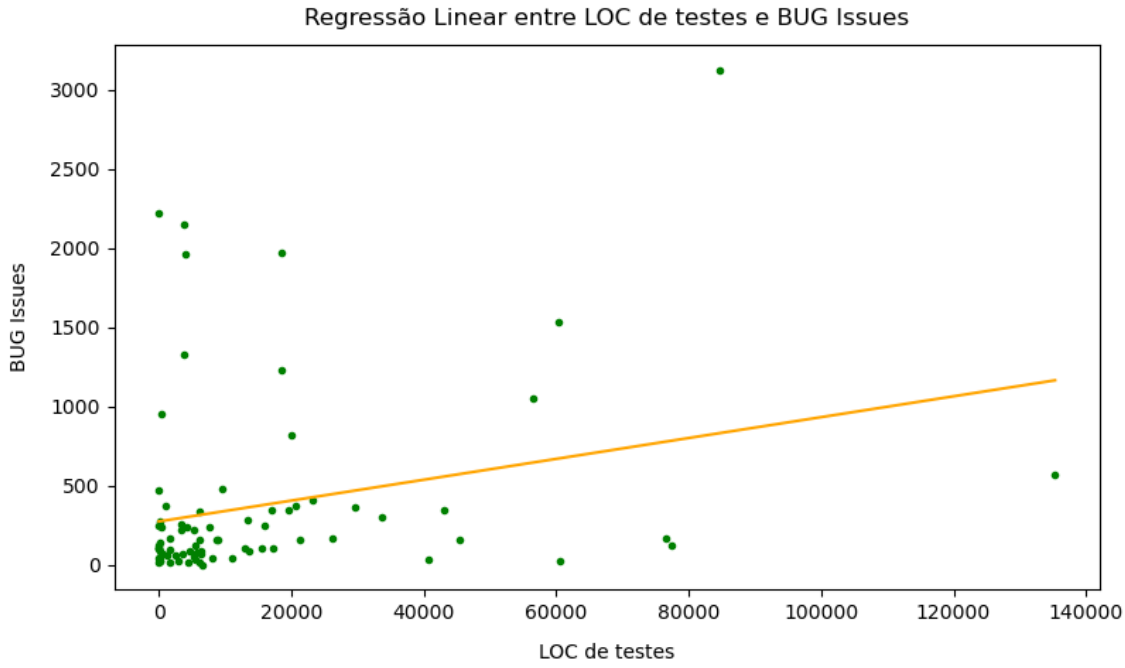

Figura 11. Gráfico de correlação entre LOC de testes e *bugs* dos repositórios JavaScript

Na Figura 10 e Figura 11, vemos a correlação entre BUG Issues e Coverage dos

repositórios e correlação entre BUG Issues e LOC de teste, respectivamente, da linguagem JavaScript, com uma correlação não linear.

4.5. Python

Nome	Autor	Cobertura	Bug Issues	Último	LOC de teste	URL
thefuck	nvbn	93%	6	09/05/20	7996	https://github.com/nvbn/thefuck
httpie	jakubroztocil	94%	84	19/05/20	2374	https://github.com/jakubroztocil/httpie
ansible	ansible	81%	16986	23/05/20	27169	https://github.com/ansible/ansible
scikit-learn	scikit-learn	34%	1091	22/05/20	87886	https://github.com/scikit-learn/scikit-learn
scrapy	scrapy	93%	227	22/05/20	22457	https://github.com/scrapy/scrapy
incubator-superset	apache	51%	2	15/05/20	21372	https://github.com/apache/incubator-superset
certbot	certbot	99%	343	21/05/20	30127	https://github.com/certbot/certbot
sentry	getsentry	90%	75	22/05/20	161338	https://github.com/getsentry/sentry
localstack	localstack	73%	284	20/05/20	2323	https://github.com/localstack/localstack
black	psf	96%	264	22/05/20	2182	https://github.com/psf/black
glances	nicolargo	40%	385	21/05/20	836	https://github.com/nicolargo/glances
ZeroNet	HelloZeroNet	54%	142	03/05/20	2781	https://github.com/HelloZeroNet/ZeroNet
zulip	zulip	77%	1440	23/05/20	26860	https://github.com/zulip/zulip
salt	saltstack	97%	9736	23/05/20	67494	https://github.com/saltstack/salt
ChatterBot	gunthercox	90%	65	01/05/20	586	https://github.com/gunthercox/ChatterBot
aiohttp	aio-libs	27%	226	20/05/20	2923	https://github.com/aio-libs/aiohttp
aws-cli	aws	89%	327	22/05/20	56767	https://github.com/aws/aws-cli
beets	beetbox	73%	328	11/05/20	21646	https://github.com/beetbox/beets
wagtail	wagtail	27%	740	22/05/20	51171	https://github.com/wagtail/wagtail
Pillow	python-pillow	88%	251	22/05/20	16156	https://github.com/python-pillow/Pillow
tpot	EpistasisLab	97%	89	21/05/20	5163	https://github.com/EpistasisLab/tpot
sympy	sympy	19%	3500	23/05/20	197579	https://github.com/sympy/sympy
tweepy	tweepy	67%	95	05/05/20	9804	https://github.com/tweepy/tweepy
pwntools	Gallopsled	68%	133	20/05/20	118	https://github.com/Gallopsled/pwntools
moviepy	Zulko	67%	113	22/05/20	2113	https://github.com/Zulko/moviepy
pip	pypa	100%	619	23/05/20	21685	https://github.com/pypa/pip
paramiko	paramiko	75%	159	08/05/20	4006	https://github.com/paramiko/paramiko
psutil	giampaolo	90%	643	22/05/20	10234	https://github.com/giampaolo/psutil
erpnext	frappe	0%	1714	23/05/20	52645	https://github.com/frappe/erpnext
loguru	Delgan	100%	21	17/05/20	5356	https://github.com/Delgan/loguru
synapse	matrix-org	65%	508	23/05/20	38391	https://github.com/matrix-org/synapse
arrow	crsmithdev	100%	68	14/05/20	95	https://github.com/crsmithdev/arrow
python-for-android	kivy	62%	17	11/05/20	5276	https://github.com/kivy/python-for-android
pytest	pytest-dev	93%	1132	22/05/20	15865	https://github.com/pytest-dev/pytest
boto3	boto	91%	80	14/05/20	9739	https://github.com/boto/boto3
spyder	spyder-ide	70%	3322	22/05/20	31014	https://github.com/spyder-ide/spyder
sh	amoffat	92%	33	12/05/20	3266	https://github.com/amoffat/sh
gevent	gevent	87%	52	19/05/20	88929	https://github.com/gevent/gevent
django-extensions	django-extensions	70%	59	07/05/20	6895	https://github.com/django-extensions/django-extensions
jedi	davidhalter	94%	321	18/05/20	8547	https://github.com/davidhalter/jedi
imbalanced-learn	scikit-learn-contrib	99%	19	11/05/20	7987	https://github.com/scikit-learn-contrib/imbalanced-learn
mycroft-core	MycroftAI	48%	89	19/05/20	9809	https://github.com/MycroftAI/mycroft-core
pip-tools	jazzband	100%	77	21/05/20	3977	https://github.com/jazzband/pip-tools
python-docs-samples	GoogleCloudPlatform	65%	446	23/05/20	36356	https://github.com/GoogleCloudPlatform/python-docs-samples
conan	conan-io	2%	628	22/05/20	72267	https://github.com/conan-io/conan
kafka-python	dpkp	84%	33	23/04/20	5365	https://github.com/dpkp/kafka-python
st2	StackStorm	73%	292	19/05/20	43846	https://github.com/StackStorm/st2
scikit-image	scikit-image	59%	223	22/05/20	27954	https://github.com/scikit-image/scikit-image
rqalpha	ricequant	65%	45	21/05/20	0	https://github.com/ricequant/rqalpha
moto	spulec	94%	80	19/05/20	85379	https://github.com/spulec/moto
cvat	opencv	66%	201	22/05/20	9459	https://github.com/opencv/cvat
librosa	librosa	99%	86	18/05/20	9074	https://github.com/librosa/librosa
hy	hylang	42%	145	22/05/20	619	https://github.com/hylang/hy
RxPY	ReactiveX	93%	30	20/05/20	31525	https://github.com/ReactiveX/RxPY
autopep8	hhatto	99%	21	03/05/20	0	https://github.com/hhatto/autopep8
DeepCTR	shenweichen	95%	1	21/05/20	1021	https://github.com/shenweichen/DeepCTR
sphinx	sphinx-doc	39%	2098	22/05/20	1505	https://github.com/sphinx-doc/sphinx
pyod	yzhao062	96%	12	18/05/20	4159	https://github.com/yzhao062/pyod
connexion	zalando	96%	78	12/05/20	6158	https://github.com/zalando/connexion
mlxtend	rasbt	92%	16	22/05/20	13097	https://github.com/rasbt/mlxtend
freqtrade	freqtrade	97%	217	21/05/20	939	https://github.com/freqtrade/freqtrade
securedrop	freedomofpress	92%	289	22/05/20	8766	https://github.com/freedomofpress/securedrop
python-slackclient	slackapi	62%	50	21/05/20	5531	https://github.com/slackapi/python-slackclient
Flask-AppBuilder	dpgaspar	74%	88	20/05/20	6595	https://github.com/dpgaspar/Flask-AppBuilder
qiskit-terra	Qiskit	76%	494	22/05/20	28109	https://github.com/Qiskit/qiskit-terra
python-slackclient	slackapi	62%	50	21/05/20	5531	https://github.com/slackapi/python-slackclient
apscheduler	agronholm	94%	97	19/05/20	3333	https://github.com/agronholm/apscheduler

Tabela 6. Repositórios *Python* analisados

Para o conjunto de todos repositórios na linguagem Python analisados, chegamos nos seguintes valores estatísticos:

Para X sendo LOCs de Teste e Y sendo BUG Issues.

- Inclinação: 0.014417959373896313
- Intercepto: 434.75623621152454
- Erro padrão: 0.008146243875985166

- Valor P: 0.0814353008104824
- Valor R: 0.21442192390533651

Interpretando esses valores a partir do Coeficiente de Pearson, temos que o valor R obtido é igual a 0.21442192390533651. Ou seja a correlação é desprezível.

Para X sendo Coverage e Y sendo BUG Issues

- Inclinação: -4.450920538756481
- Intercepto: 1104.4751616892186
- Erro padrão: 12.078448237619083
- Valor P: 0.7136971685513529
- Valor R: -0.045659257632628104

Interpretando esses valores a partir do Coeficiente de Pearson, temos que o valor R obtido é igual a -0.045659257632628104. Ou seja a correlação é desprezível.

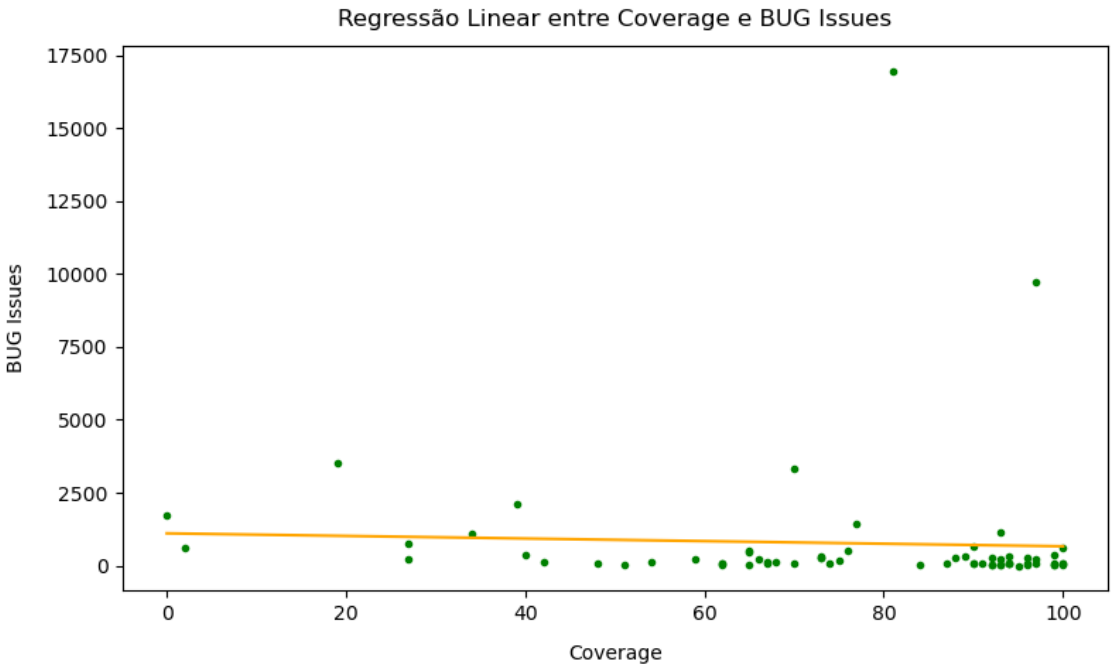

Figura 12. Gráfico de correlação entre cobertura e *bugs* dos repositórios Python

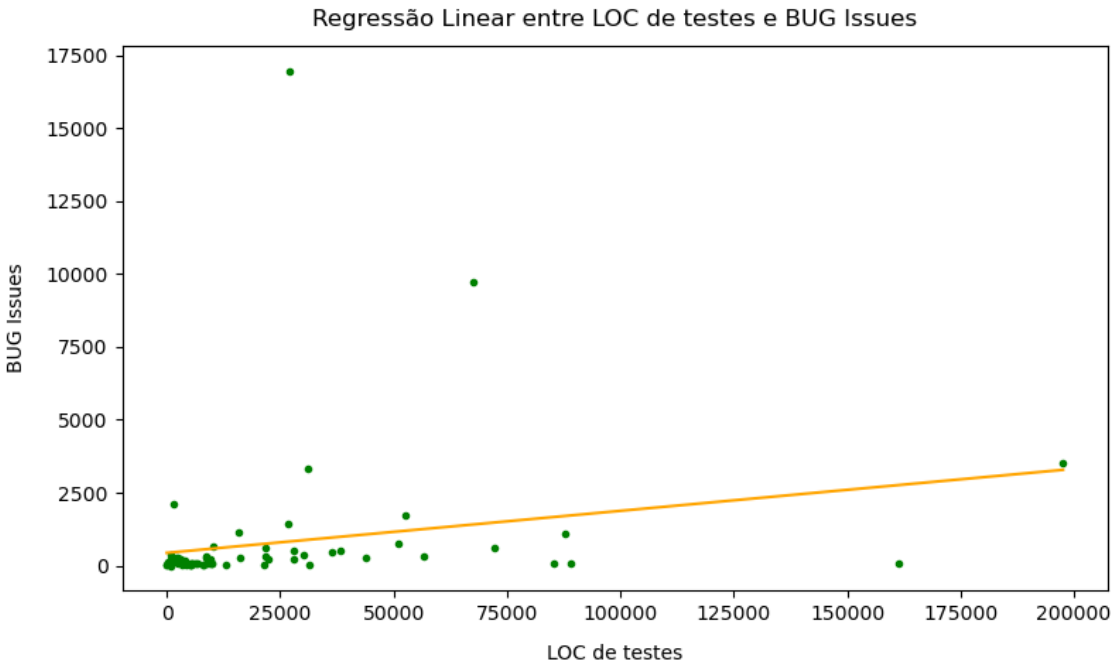

Figura 13. Gráfico de correlação entre LOC de testes e *bugs* dos repositórios Python

Na Figura 12 e Figura 13, vemos a correlação entre BUG Issues e Coverage dos repositórios e correlação entre BUG Issues e LOC de teste, respectivamente, da linguagem

Python, com uma correlação não linear.

5. Análise dos resultados

Os resultados da pesquisa permitiram verificar a validade da hipótese nula. Para isso, foram observadas as linguagens de forma isolada, utilizando o coeficiente de Pearson para identificar se há uma relação considerável entre a quantidade de *bugs* e LOC ou cobertura de testes do projeto.

Após esta verificação, chegou-se a conclusão que a correlação entre as variáveis de ambas as métricas anteriormente apresentadas é desprezível. Com ressalvas à linguagem *Java* — que obteve uma correlação moderada de aproximadamente 0,5 para a métrica 1 (

$$\frac{\text{Quantidade de bug issues}}{\text{Linhas de código de teste unitário}}$$

) — Nas demais linguagens testadas, o coeficiente de Pearson foi sempre próximo de 0. Desta forma, não é possível confirmar a hipótese nula para as linguagens abordadas neste estudo.

6. Trabalhos relacionados

O artigo *Learning Test-Driven Development by Counting Lines*[14] 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.

Driving Software Quality: How Test-Driven Development Impacts Software Quality[2] 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*[12] 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.

A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last? [4] 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?*[9] 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 desenvolvimento orientado à testes? 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". 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*[8] 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 em *Quality of Testing in Test Driven Development*[1] 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.

On the effectiveness of the test-first approach to programming[3] 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. O artigo *Effects of Developer Experience on Learning and Applying Unit Test-*

Driven Development[10] 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 sobre desenvolvimento orientado a testes, temos *A structured experiment of test-driven development* [5], 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.

The Impact of Agile Software Development Process on the Quality of Software Product[7] 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 bom produto.

Current State of the Research in Agile Quality Development[6] discute a respeito das dúvidas presentes em organizações sobre a qualidade do software quando utilizado o desenvolvimento ágil. 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*[11] 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.

Por fim *An Empirical Study of Bugs in Test Code*[13] traz a discussão a respeito dos bugs em código de teste. 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.

7. Ameaças à validade

Neste artigo, foram identificadas duas validades de construção. Primeiramente, o filtro de *bugs* pode ser julgado ineficiente, por não abrangir todos os rótulos utilizados em *issues* do *GitHub* relacionados à erros em um software. Além disso, as extensões procuradas para a contagem de linhas de código em arquivos podem não abrangir todas as nomenclaturas utilizadas, portanto, há a possibilidade de contagem errada por não considerar alguns arquivos de teste.

Além disso uma possível ameaça de conclusão está no fato de que a comunidade poderia não estar reportando as *issues* diretamente no repositório do *GitHub*, e sim através de outras plataformas como *Bugzilla*. Este fato pode influenciar diretamente na conclusão de nosso trabalho, uma vez que ele impacta diretamente na análise dos dados obtidos na execução da metodologia, influenciando a conclusão final da pesquisa.

8. Conclusão

Este trabalho possibilitou a recuperação de informações em repositórios GitHub de múltiplas linguagens através de *Web Crawling* e uso de *Scripts* em *Python*. Os resultados explicitam que, embora haja forte correlação entre LOC de testes e número de *Bugs* reportados em *Java*, no geral, a correlação foi inexistente, contradizendo a hipótese nula.

Portanto, com o cálculo das métricas supracitadas nos tópicos desta pesquisa, conseguimos responder todas as questões do trabalho. A primeira questão, onde há a indagação se existe relação entre a quantidade de testes unitários e quantidade de BUG Issues, temos a conclusão que esta relação inexiste. A mesma conclusão foi obtida quando calculamos a métrica que responde a segunda questão, que é descobrir a relação entre o coverage do código dos repositórios e a quantidade de bugs. Portanto, podemos dizer que a hipótese nula foi refutada e a hipótese alternativa foi confirmada, ou seja, testes unitários não impactam nos BUG Issues dos repositórios.

Finalmente, demonstra-se fundamental uma análise mais aprofundada na razão pela qual esta hipótese inicial tenha sido refutada e, para trabalhos futuros, recomenda-se verificar também a participação e interação da comunidade destes repositórios juntamente com os dados obtidos neste trabalho, uma vez que este fator pode ter colaborado para a hipótese alternativa.

Referências

[1] Sasikumar Cauevic Punnekkat e Daniel Sundmark. «Quality of Testing in Test Driven Development». Em: i (2012). DOI: 10.1109/QUATIC.2012.49.

[2] How Test-driven Development e Impacts Software. «Driving Software Quality : How Test-Driven Development Impacts Software Quality». Em: (2006), pp. 70–71.

[3] Hakan Erdogmus, Maurizio Morisio e Ieee Computer Society. «On the Effectiveness of the Test-First Approach to Programming». Em: 31.3 (2005), pp. 226–237.

[4] Davide Fucci, Hakan Erdogmus e Ieee Burak Turhan. «A Dissection of the Test-Driven Development Process : Does It Really Matter to Test-First or to Test-Last ?» Em: 6.1 (2016). DOI: 10.1109/TSE.2016.2616877.

[5] Boby George e Laurie Williams. «A structured experiment of test-driven development». Em: 46 (2004), pp. 337–342. DOI: 10.1016/j.infsof.2003.09.011.

[6] P. Jain, L. Ahuja e A. Sharma. «Current state of the research in agile quality development». Em: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. 2016, pp. 1177–1179.

[7] Parita Jain, Arun Sharma e Laxmi Ahuja. «The Impact of Agile Software Development Process on the Quality of Software Product». Em: *2018 7th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)* (2018), pp. 812–815. DOI: 10.1109/ICRITO.2018.8748529.

[8] Reid Kaufmann e David Janzen. «Implications of test-driven development a pilot study». Em: out. de 2003, pp. 298–299. DOI: 10.1145/949344.949421.

[9] Sami Kollanus. «Test-Driven Development - Still a Promising Approach ?» Em: (2010). DOI: 10.1109/QUATIC.2010.73.

[10] Roberto Latorre. «Effects of Developer Experience on Learning and Applying Unit Test-Driven Development». Em: 40.4 (2014), pp. 381–395.

[11] Navnath Shete. «An Empirical Study of Test Cases in Software Testing». Em: 978 (2014).

[12] Christopher D Thomson, Mike Holcombe e Anthony J H Simons. «What Makes Testing Work : Nine Case Studies of Software Development Teams». Em: (2009). DOI: 10.1109/TAICPART.2009.12.

[13] Arash Vahabzadeh e Ali Mesbah. «An Empirical Study of Bugs in Test Code». Em: (2015), pp. 101–110.

[14] Bas Vodde e Nokia Networks. «Learning Test-Driven Development by Counting Lines». Em: (2007).