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

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

Análise da efetividade de testes unitários nos projetos JavaScript mais populares do GitHub que possuem informação de coverage

Belo Horizonte
2020

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

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

QUESTIONS:

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

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

METRICS:

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

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

1.1. Tabela GQM:

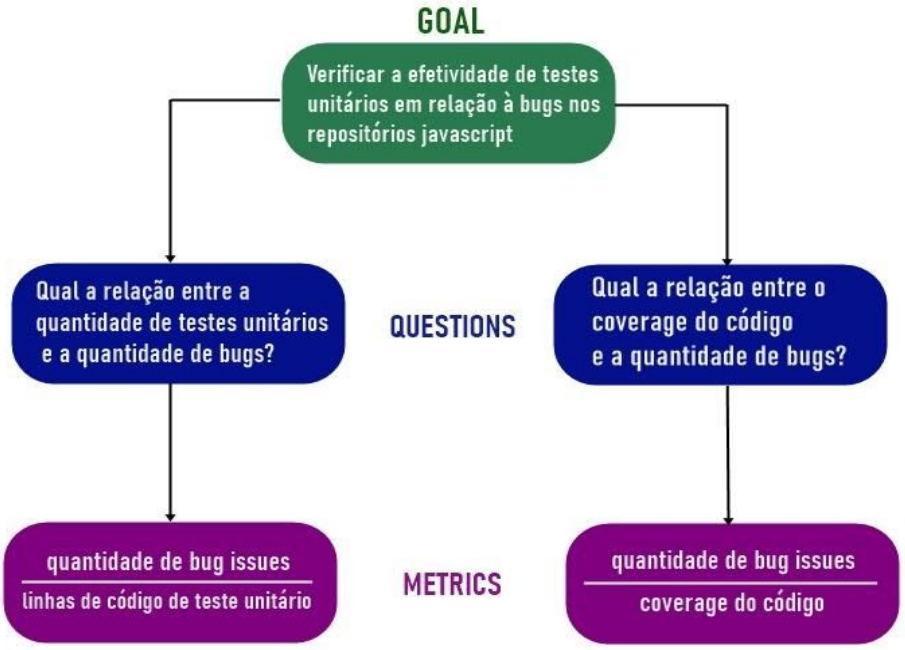

Figura 1. Tabela GQM do projeto.

2. Metodologia

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

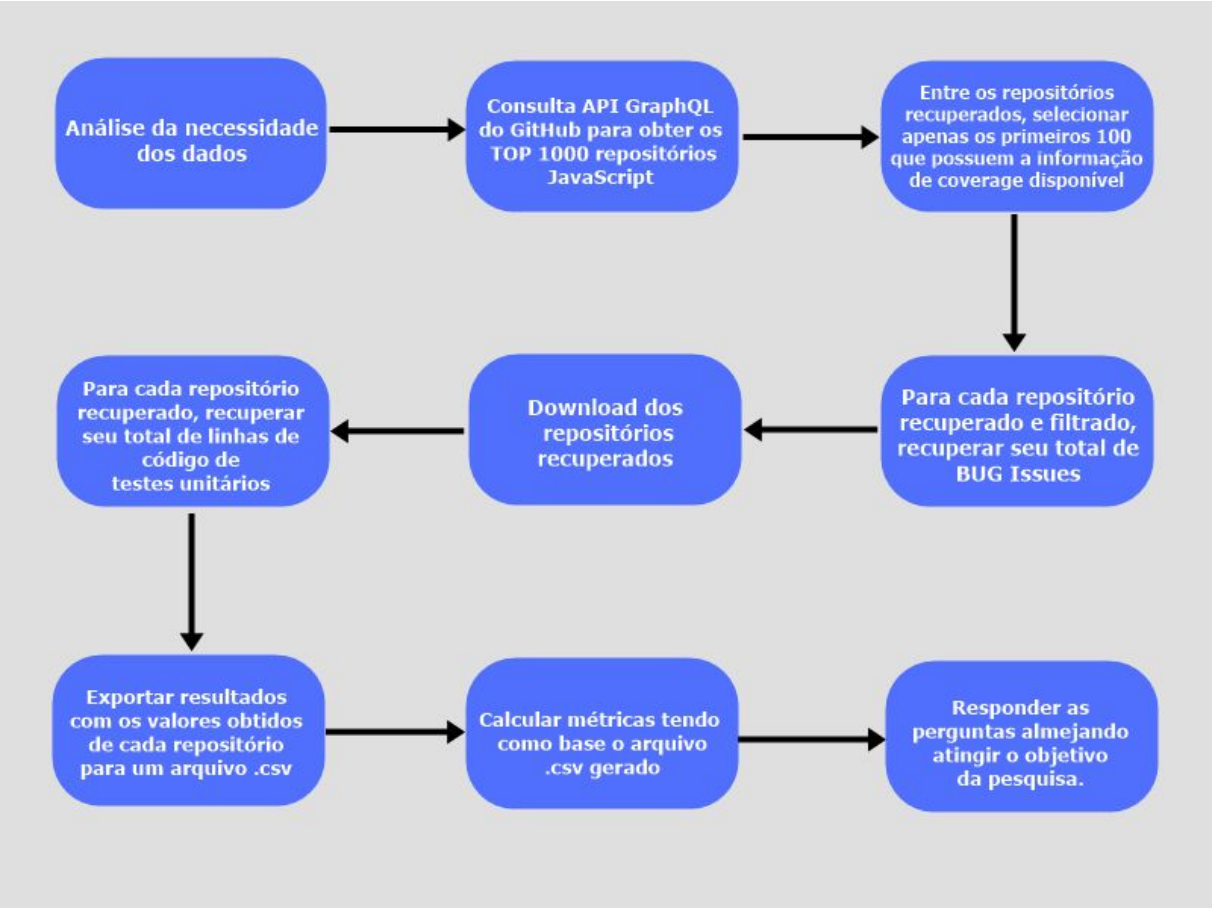

Figura 2. Overview da metodologia do projeto.

Na primeira etapa, tendo em vista que o Goal desta pesquisa depende de uma análise de repositórios GitHub cuja linguagem primária é JavaScript, faremos, através da execução de um script em Python, uma consulta a API GraphQL do GitHub que nos retornará os 1000 repositórios JavaScript que mais possuem estrela. Após a recuperação, é iniciada a etapa de recuperação de seus coverages. Para isto, são realizadas consultas para o site https://coveralls.io/github (que possui essa informação em uma div identificada por “repoShowPercentage”). Nesta consulta passamos, para cada repositório, seu owner e seu nome. Esta etapa é finalizada, após a obtenção do coverage dos primeiros 100 repositórios.

Para cada um dos 100 repositórios, será realizada mais uma consulta a API GraphQL do GitHub que 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 exatamente iguais a: "BUG", "bug", "Bug", "Type: Bug", "Type: BUG", "Type: bug","type: Bug", "type: BUG", "type: bug", "browser bug", "Error", "ERROR", "error", "Failure", "FAILURE", "failure", "Confirmed Bug", "confirmed bug", "Type: Bug / Error", "good first bug", "Good First Bug", "GOOD FIRST BUG", "confirmed-bug", "bug critical", "bug minor", "bug moderate", "type:bug", "Type:bug", "type:Bug", "Type:Bug". Nesta consulta, será retornada a quantidade total de BUG Issues do repositório.

Com a quantidade total de BUG Issues recuperadas para cada repositório, resta apenas obter a quantidade total de linhas de código de testes unitários de cada repositório. Para isso, um script será desenvolvido utilizando a biblioteca GitPython, cuja funcionalidade será baixar todos os repositórios necessários. Posteriormente, um outro script irá, através da análise do código fonte de cada repositório, realizar a contagem de linhas de código de teste unitário. Para identificar um arquivo de teste e, em seguida contar seus LOC de teste, são considerados apenas arquivos cujas extensões contenham as strings: “.spec.js”, “.karma.js”, “.test.js” e “.tests.js”.

Em seguida, todas as informações obtidas destes 100 repositórios nas etapas anteriores serão exportadas para um arquivo .csv.

Finalmente, com o arquivo .csv gerado, será realizada a análise dos dados obtidos. As métricas identificadas para buscar respostas às questões levantadas na pesquisa serão calculadas e as perguntas respondidas.

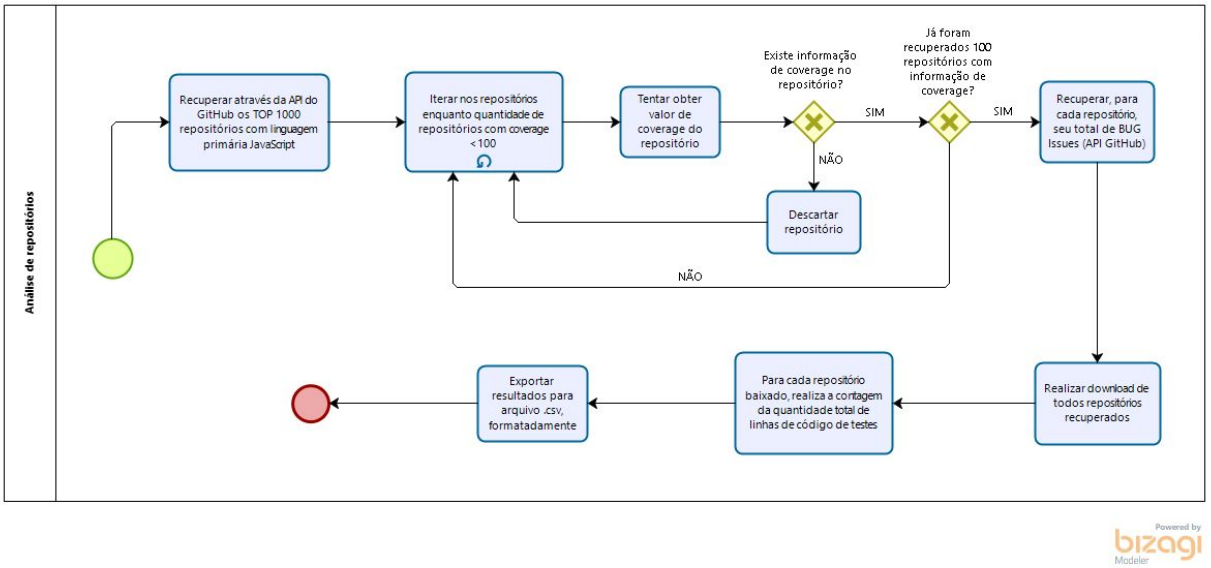

Figura 3. Exemplo unificado do processo realizado pelos scripts que compõem a solução que coleta os dados necessários dos repositórios.

3. Resultados

Com a análise do arquivo .csv gerado, foi possível calcular as métricas:

Nome	Owner	Coverage	BUG Issues	LOC de teste	URL	MÉTRICA 1	MÉTRICA 2
vue	vuejs	96%	45	28658	https://github.com/vuejs/vue	0,001570242	46,88
react	facebook	86%	105	18	https://github.com/facebook/react	5,833333333	122,09
bootstrap	twbs	95%	12	5237	https://github.com/twbs/bootstrap	0,002291388	12,63
react-native	facebook	9%	311	0	https://github.com/facebook/react-native	#DIV/0!	3455,56
webpack	webpack	95%	28	14211	https://github.com/webpack/webpack	0,001970305	29,47
Chart.js	chartjs	94%	110	18466	https://github.com/chartjs/Chart.js	0,005956894	117,02
express	expressjs	100%	4	0	https://github.com/expressjs/express	#DIV/0!	4,00
Semantic-UI	Semantic-Org	89%	254	311	https://github.com/Semantic-Org/Semantic-UI	0,816720257	285,39
next.js	zeit	52%	10	30584	https://github.com/zeit/next.js	0,000326968	19,23
lodash	lodash	33%	5	4288	https://github.com/lodash/lodash	0,001166045	15,15
moment	moment	88%	26	0	https://github.com/moment/moment	#DIV/0!	29,55
serverless	serverless	91%	287	52464	https://github.com/serverless/serverless	0,005470418	315,38
Ghost	TryGhost	44%	38	0	https://github.com/TryGhost/Ghost	#DIV/0!	86,36
gulp	gulpjs	100%	1	60	https://github.com/gulpjs/gulp	0,016666667	1,00
hexo	hexojs	98%	20	0	https://github.com/hexojs/hexo	#DIV/0!	20,41
mermaid	mermaid-js	53%	42	8914	https://github.com/mermaid-js/mermaid	0,004711689	79,25
video.js	videojs	86%	6	19342	https://github.com/videojs/video.js	0,000310206	6,98
Rocket.Chat	RocketChat	57%	310	2737	https://github.com/RocketChat/Rocket.Chat	0,113262696	543,86
async	caolan	99%	0	130	https://github.com/caolan/async	0	0,00
preact	preactjs	100%	8	25721	https://github.com/preactjs/preact	0,00031103	8,00
react-boilerplate	react-boilerplate	100%	0	4609	https://github.com/react-boilerplate/react-boilerplate	0	0,00
underscore	jashkenas	97%	3	0	https://github.com/jashkenas/underscore	#DIV/0!	3,09
request	request	97%	1	0	https://github.com/request/request	#DIV/0!	1,03
Modernizr	Modernizr	97%	0	0	https://github.com/Modernizr/Modernizr	#DIV/0!	0,00
faker.js	Marak	99%	2	0	https://github.com/Marak/faker.js	#DIV/0!	2,02
date-fns	date-fns	97%	0	0	https://github.com/date-fns/date-fns	#DIV/0!	0,00
cheerio	cheeriojs	99%	7	0	https://github.com/cheeriojs/cheerio	#DIV/0!	7,07
mobx	mobxjs	94%	4	9	https://github.com/mobxjs/mobx	0,444444444	4,26
layui	sentsin	26%	0	0	https://github.com/sentsin/layui	#DIV/0!	0,00
ace	ajaxorg	92%	0	0	https://github.com/ajaxorg/ace	#DIV/0!	0,00
sails	balderdashy	99%	25	5957	https://github.com/balderdashy/sails	0,004196743	25,25
wepy	Tencent	39%	8	2249	https://github.com/Tencent/wepy	0,003557137	20,51
react-select	JedWatson	70%	18	4642	https://github.com/JedWatson/react-select	0,003877639	25,71
lighthouse	GoogleChrome	91%	63	160	https://github.com/GoogleChrome/lighthouse	0,39375	69,23
mocha	mochajs	93%	54	14353	https://github.com/mochajs/mocha	0,00376228	58,06
pug	pugjs	77%	12	77057	https://github.com/pugjs/pug	0,000155729	15,58
normalizr	paularmstrong	100%	1	2723	https://github.com/paularmstrong/normalizr	0,000367242	1,00
react-virtualized	bvaughn	100%	24	0	https://github.com/bvaughn/react-virtualized	#DIV/0!	24,00
scrollreveal	jlmakes	30%	4	551	https://github.com/jlmakes/scrollreveal	0,007259528	13,33
ava	avajs	87%	20	2	https://github.com/avajs/ava	10	22,99
commander.js	tj	79%	0	3531	https://github.com/tj/commander.js	0	0,00
parse-server	parse-community	9%	18	76458	https://github.com/parse-community/parse-server	0,000235423	200,00
passport	jaredhanson	99%	0	7098	https://github.com/jaredhanson/passport	0	0,00
sharp	lovell	100%	5	0	https://github.com/lovell/sharp	#DIV/0!	5,00
eslint	eslint	69%	44	0	https://github.com/eslint/eslint	#DIV/0!	63,77
graphql-js	graphql	98%	7	0	https://github.com/graphql/graphql-js	#DIV/0!	7,14
pkg	zeit	9%	5	0	https://github.com/zeit/pkg	#DIV/0!	55,56
hubot	hubotio	76%	0	0	https://github.com/hubotio/hubot	#DIV/0!	0,00
mysql	mysqljs	98%	7	0	https://github.com/mysqljs/mysql	#DIV/0!	7,14

bower	bower	85%	0	0	https://github.com/bower/bower	#DIV/0!	0,00
dva	dvajs	86%	0	2890	https://github.com/dvajs/dva	0	0,00
Mock	nuysoft	100%	3	0	https://github.com/nuysoft/Mock	#DIV/0!	3,00
medium-editor	yabwe	94%	53	10364	https://github.com/yabwe/medium-editor	0,005113856	56,38
chalk	chalk	100%	2	0	https://github.com/chalk/chalk	#DIV/0!	2,00
fastify	fastify	99%	5	24742	https://github.com/fastify/fastify	0,000202086	5,05
inferno	infernojs	95%	2	61339	https://github.com/infernojs/inferno	3,26057E-05	2,11
riot	riot	100%	1	958	https://github.com/riot/riot	0,001043841	1,00
svgo	svg	77%	9	0	https://github.com/svg/svgo	#DIV/0!	11,69
ws	websockets	100%	0	6118	https://github.com/websockets/ws	0	0,00
vant	youzan	91%	5	13226	https://github.com/youzan/vant	0,000378043	5,49
handsontable	handsontable	67%	554	18077	https://github.com/handsontable/handsontable	0,030646678	826,87
charts	frappe	38%	6	10	https://github.com/frappe/charts	0,6	15,79
loopback	strongloop	90%	4	16418	https://github.com/strongloop/loopback	0,000243635	4,44
pouchdb	pouchdb	31%	1	0	https://github.com/pouchdb/pouchdb	#DIV/0!	3,23
node-redis	NodeRedis	100%	2	9192	https://github.com/NodeRedis/node-redis	0,000217581	2,00
Inquirer.js	SBoudrias	92%	7	0	https://github.com/SBoudrias/Inquirer.js	#DIV/0!	7,61
feathers	feathersjs	19%	3	6410	https://github.com/feathersjs/feathers	0,000468019	15,79
kitematic	docker	67%	31	0	https://github.com/docker/kitematic	#DIV/0!	46,27
johnny-five	rwaldron	92%	3	0	https://github.com/rwaldron/johnny-five	#DIV/0!	3,26
dotenv	motdotla	100%	0	0	https://github.com/motdotla/dotenv	#DIV/0!	0,00
appium	appium	89%	15	664	https://github.com/appium/appium	0,022590361	16,85
co	tj	100%	0	0	https://github.com/tj/co	#DIV/0!	0,00
sweetalert2	sweetalert2	92%	1	2723	https://github.com/sweetalert2/sweetalert2	0,000367242	1,09
You-Dont-Need-Lodash-Underscore	you-dont-need	100%	0	0	https://github.com/you-dont-need/You-Dont-Need-Lodash-Underscore	#DIV/0!	0,00
browser-sync	BrowserSync	79%	9	0	https://github.com/BrowserSync/browser-sync	#DIV/0!	11,39
NodeBB	NodeBB	90%	12	0	https://github.com/NodeBB/NodeBB	#DIV/0!	13,33
shields	badges	96%	16	6939	https://github.com/badges/shields	0,002305808	16,67
virtual-dom	Matt-Esch	98%	0	0	https://github.com/Matt-Esch/virtual-dom	#DIV/0!	0,00
react-dates	airbnb	81%	28	0	https://github.com/airbnb/react-dates	#DIV/0!	34,57
You-Dont-Need-Momentjs	you-dont-need	100%	0	0	https://github.com/you-dont-need/You-Dont-Need-Momentjs	#DIV/0!	0,00
nightwatch	nightwatchjs	85%	4	0	https://github.com/nightwatchjs/nightwatch	#DIV/0!	4,71
jquery-mobile	jquery	43%	0	0	https://github.com/jquery/jquery-mobile	#DIV/0!	0,00
supertest	visionmedia	97%	1	0	https://github.com/visionmedia/supertest	#DIV/0!	1,03
node-red	node-red	78%	7	0	https://github.com/node-red/node-red	#DIV/0!	8,97
mathjs	josdejong	49%	10	45168	https://github.com/josdejong/mathjs	0,000221396	20,41
node-restify	restify	33%	9	13024	https://github.com/restify/node-restify	0,000691032	27,27
falcor	Netflix	93%	8	12898	https://github.com/Netflix/falcor	0,000620251	8,60
semantic-release	semantic-release	99%	1	7248	https://github.com/semantic-release/semantic-release	0,000137969	1,01
react-slingshot	coryhouse	92%	0	1081	https://github.com/coryhouse/react-slingshot	0	0,00
marko	marko-js	91%	8	2931	https://github.com/marko-js/marko	0,002729444	8,79
nock	nock	100%	5	0	https://github.com/nock/nock	#DIV/0!	5,00
markdown-it	markdown-it	100%	1	0	https://github.com/markdown-it/markdown-it	#DIV/0!	1,00
summernote	summernote	81%	34	3273	https://github.com/summernote/summernote	0,010388023	41,98
luxon	moment	96%	0	6620	https://github.com/moment/luxon	0	0,00
dataloader	graphql	100%	0	1293	https://github.com/graphql/dataloader	0	0,00
art-template	aui	80%	2	0	https://github.com/aui/art-template	#DIV/0!	2,50
web3.js	ethereum	91%	23	0	https://github.com/ethereum/web3.js	#DIV/0!	25,27
uncss	uncss	98%	2	0	https://github.com/uncss/uncss	#DIV/0!	2,04
connect	senchalabs	100%	0	0	https://github.com/senchalabs/connect	#DIV/0!	0,00
debug	visionmedia	88%	8	0	https://github.com/visionmedia/debug	#DIV/0!	9,09
MÉDIA		82%	28	6841	**Devido às divisões por zero, é impossível calcular a média**	--	71,04
MEDIANA		92%	5	14	**Devido às divisões por zero, é impossível calcular a mediana**	--	7,11
					MÉTRICAS: M1: Qnt. BUG Issues/LOC Teste unitário		
					M2: Qnt BUG Issues/Coverage		

Porém, como é possível verificar na tabela acima, não foi possível calcular a métrica 1, uma vez que a quantidade total de LOC de teste de alguns repositórios foi igual a 0 (devido ao fato de seguir um padrão diferente de extensões, fugindo do .spec, .karma, .test e .tests, não sendo, portanto, contabilizado). Com a divisão não podendo ser concluída para o cálculo da métrica 1, filtramos todos repositórios cuja

quantidade total contabilizada de LOCs de teste fosse igual a 0 e a retiramos da tabela. Posteriormente, refizemos os cálculos:

Nome	Owner	Coverage	BUG Issues	LOC de teste	URL	MÉTRICA 1	MÉTRICA 2
vue	vuejs	96%	45	28658	https://github.com/vuejs/vue	0,001570242166	46,88
react	facebook	86%	105	18	https://github.com/facebook/react	5,833333333333	122,09
bootstrap	twbs	95%	12	5237	https://github.com/twbs/bootstrap	0,002291388199	12,63
webpack	webpack	95%	28	14211	https://github.com/webpack/webpack	0,001970304694	29,47
Chart.js	chartjs	94%	110	18466	https://github.com/chartjs/Chart.js	0,005956893751	117,02
Semantic-UI	Semantic-Org	89%	254	311	https://github.com/Semantic-Org/Semantic-UI	0,816720257235	285,39
next.js	zeit	52%	10	30584	https://github.com/zeit/next.js	0,000326968349	19,23
lodash	lodash	33%	5	4288	https://github.com/lodash/lodash	0,001166044776	15,15
serverless	serverless	91%	287	52464	https://github.com/serverless/serverless	0,005470417810	315,38
gulp	gulpjs	100%	1	60	https://github.com/gulpjs/gulp	0,016666666667	1,00
mermaid	mermaid-js	53%	42	8914	https://github.com/mermaid-js/mermaid	0,004711689477	79,25
video.js	videojs	86%	6	19342	https://github.com/videojs/video.js	0,000310205770	6,98
Rocket.Chat	RocketChat	57%	310	2737	https://github.com/RocketChat/Rocket.Chat	0,113262696383	543,86
async	caolan	99%	0	130	https://github.com/caolan/async	0,000000000000	0,00
preact	preactjs	100%	8	25721	https://github.com/preactjs/preact	0,000311029898	8,00
react-boilerplate	react-boilerplate	100%	0	4609	https://github.com/react-boilerplate/react-boilerplate	0,000000000000	0,00
mobx	mobxjs	94%	4	9	https://github.com/mobxjs/mobx	0,444444444444	4,26
sails	balderdashy	99%	25	5957	https://github.com/balderdashy/sails	0,004196743327	25,25
wepy	Tencent	39%	8	2249	https://github.com/Tencent/wepy	0,003557136505	20,51
react-select	JedWatson	70%	18	4642	https://github.com/JedWatson/react-select	0,003877638949	25,71
lighthouse	GoogleChrome	91%	63	160	https://github.com/GoogleChrome/lighthouse	0,393750000000	69,23
mocha	mochajs	93%	54	14353	https://github.com/mochajs/mocha	0,003762279663	58,06
pug	pugjs	77%	12	77057	https://github.com/pugjs/pug	0,000155728876	15,58
normalizr	paularmstrong	100%	1	2723	https://github.com/paularmstrong/normalizr	0,000367242012	1,00
scrollreveal	jlmakes	30%	4	551	https://github.com/jlmakes/scrollreveal	0,007259528131	13,33
ava	avajs	87%	20	2	https://github.com/avajs/ava	10,000000000000	22,99
commander.js	tj	79%	0	3531	https://github.com/tj/commander.js	0,000000000000	0,00
parse-server	parse-community	9%	18	76458	https://github.com/parse-community/parse-server	0,000235423370	200,00
passport	jaredhanson	99%	0	7098	https://github.com/jaredhanson/passport	0,000000000000	0,00
dva	dvajs	86%	0	2890	https://github.com/dvajs/dva	0,000000000000	0,00
medium-editor	yabwe	94%	53	10364	https://github.com/yabwe/medium-editor	0,005113855654	56,38
fastify	fastify	99%	5	24742	https://github.com/fastify/fastify	0,000202085523	5,05
inferno	infernojs	95%	2	61339	https://github.com/infernojs/inferno	0,000032605683	2,11
riot	riot	100%	1	958	https://github.com/riot/riot	0,001043841336	1,00
ws	websockets	100%	0	6118	https://github.com/websockets/ws	0,000000000000	0,00
vant	youzan	91%	5	13226	https://github.com/youzan/vant	0,000378043248	5,49
handsontable	handsontable	67%	554	18077	https://github.com/handsontable/handsontable	0,030646678099	826,87
charts	frappe	38%	6	10	https://github.com/frappe/charts	0,600000000000	15,79
loopback	strongloop	90%	4	16418	https://github.com/strongloop/loopback	0,000243635035	4,44
node-redis	NodeRedis	100%	2	9192	https://github.com/NodeRedis/node-redis	0,000217580505	2,00
feathers	feathersjs	19%	3	6410	https://github.com/feathersjs/feathers	0,000468018721	15,79
appium	appium	89%	15	664	https://github.com/appium/appium	0,022590361446	16,85
sweetalert2	sweetalert2	92%	1	2723	https://github.com/sweetalert2/sweetalert2	0,000367242012	1,09
shields	badges	96%	16	6939	https://github.com/badges/shields	0,002305807753	16,67
mathjs	josdejong	49%	10	45168	https://github.com/josdejong/mathjs	0,000221395678	20,41
node-restify	restify	33%	9	13024	https://github.com/restify/node-restify	0,000691031941	27,27
falcor	Netflix	93%	8	12898	https://github.com/Netflix/falcor	0,000620251202	8,60
semantic-release	semantic-release	99%	1	7248	https://github.com/semantic-release/semantic-release	0,000137969095	1,01
react-slingshot	coryhouse	92%	0	1081	https://github.com/coryhouse/react-slingshot	0,000000000000	0,00
marko	marko-js	91%	8	2931	https://github.com/marko-js/marko	0,002729443876	8,79
summernote	summernote	81%	34	3273	https://github.com/summernote/summernote	0,010388023220	41,98
luxon	moment	96%	0	6620	https://github.com/moment/luxon	0,000000000000	0,00
dataloader	graphql	100%	0	1293	https://github.com/graphql/dataloader	0,000000000000	0,00
MÉDIA		81%	41,2641509	12908,41509		0,346114569317	59,17
MEDIANA		91%	8	6118		0,001043841336	15,15
					MÉTRICAS: M1: Qnt. BUG Issues/LOC Teste unitário		
					M2: Qnt BUG Issues/Coverage		

E, com a filtragem, todas as métricas puderam ser calculadas com sucesso.

4. Análise dos resultados

A partir dos resultados obtidos podemos concluir, inicialmente, que não encontramos uma relação clara entre a quantidade de testes unitários e a quantidade de bugs e nem entre o coverage do código e a quantidade de bugs.

Dentre os repositórios analisados, não há um padrão claro entre os valores obtidos, visto que a quantidade de bugs (BUG Issues), que é o valor base para nossos cálculos, é bastante discrepante entre os projetos, fato confirmado ao analisar a média e a mediana desta coluna. O que resultou nos valores divergentes encontrados em nossas métricas.

5. Trabalhos relacionados

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

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

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

O artigo **A Dissection of the Test-Driven Development Process: Does It Really Matter to Test-First or to Test-Last?** (D. Fucci, 2017) tem como objetivo

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

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

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

que o grupo que realizou os testes anteriormente foi mais produtivo e o código produzido pelos grupos tinham complexidade semelhante. Este artigo se mostra fundamental, pois possui relação direta com as implicações que utilização de TDD têm em relação à qualidade do software e, consequentemente, quantidade de bugs e complexidade.

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

Outros dois artigos importantes acerca de TDD devem ser citados, o primeiro: **Effects of Developer Experience on Learning and Applying Unit Test-Driven Development** (R. Latorre, 2014) é resultado de um experimento com **30 programadores** para avaliar a dificuldade de aprendizado de Frameworks orientados à teste. Este se relaciona à esta pesquisa pois a dificuldade de aprendizado pode ser um fator crucial ao número de bugs, mesmo adotando

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

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

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

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

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

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

6. Referências

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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