

对Web前端项目测试情况的实证研究

陈俊达 刘宇航

2020年12月30日

目录

- 研究背景
- 研究问题
- 研究方法
- 数据获取
- 数据分析
- 研究结论
- 有效性威胁
- 总结和后续工作

研究背景

- Web前端的重要性的和复杂度迅速增加
- 真正的编程语言代码量增大
- 对测试的关注提高
- TypeScript的出现

研究问题

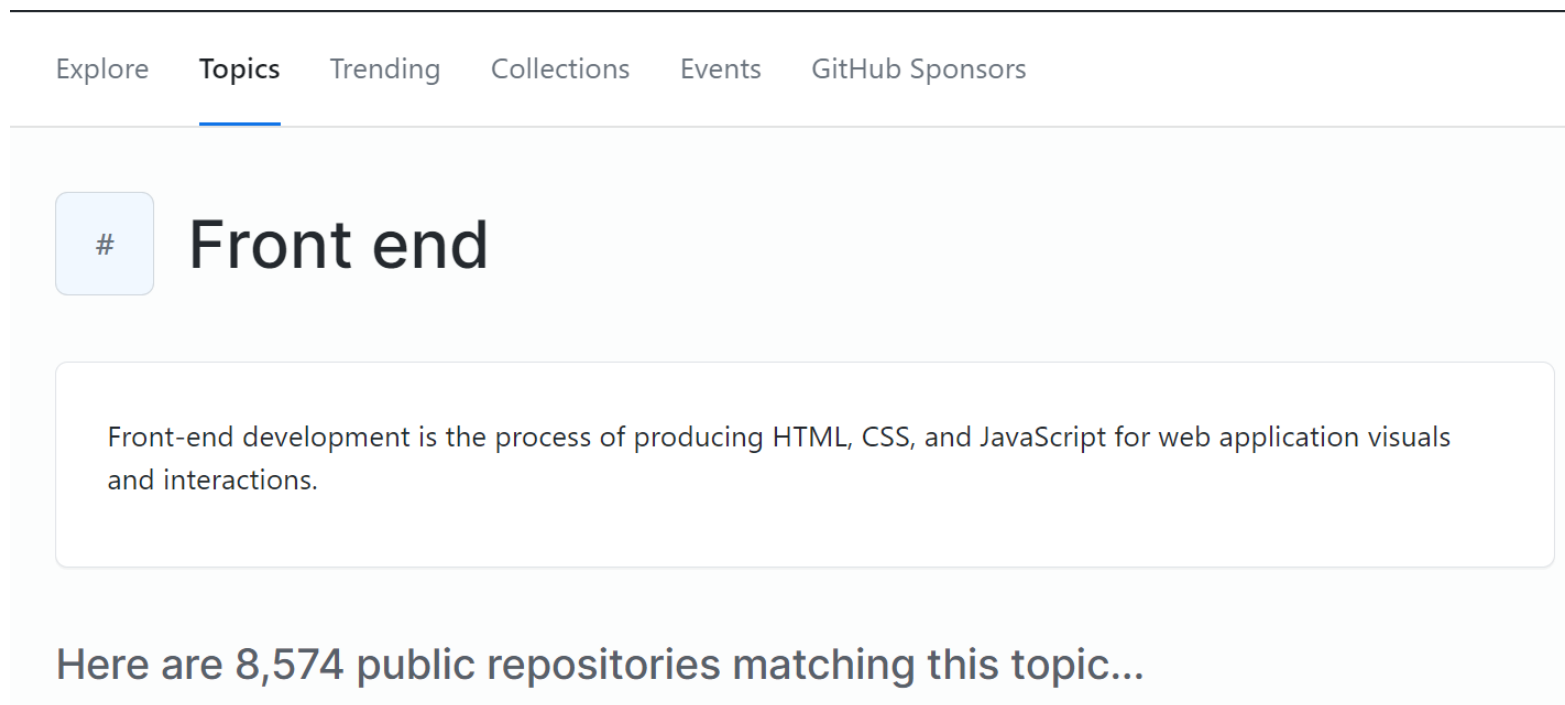
- 现代前端开发以JS、TS代码为主，只考虑JS、TS代码量
- RQ1: **测试用例的数量和整体代码量**之间有什么关系？
 - 探究前端领域的项目中，代码量和测试用例数量的具体关系
 - 可作为对项目测试充分性的初步判断标准
- RQ2: **测试用例数量与bug数量**的关系是怎么样的？
 - 直觉：测试越多，bug应该越少
- RQ3: 使用TypeScript是否能降低bug的数量？
 - TypeScript是JavaScript的带有类型系统的超集
 - 能在编译期发现很多JS只能在运行时发生的错误
 - **TS的代码量占总体代码量的比例和bug数量/总代码量**之间的关系

研究方法

- 采用**调查**的方法
- 样本：GitHub上Star最多的200个前端领域的代码
- 需要获取的数据
 - 每个项目的JS、TS代码量
 - 每个项目的bug数量
 - 每个项目的单元测试用例数量
- 数据获取半自动化

数据获取： 前端项目

- 使用GitHub提供的API搜索topic是frontend的项目， 并根据stars排序
 - 每一页(per page)最多100个项目， 可通过指定page参数指定第几页
 - 所以请求两次API， 获取2页的数据




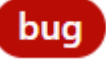
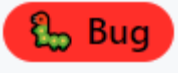
数据获取：项目JS、TS代码量

- Cloc

```
# pkucjd at LAB-DDADAAL in ~\Code\pkuhomework\test\project\projects\vue on git:dev [16:55:48]
→ cloc . --exclude-dir=node_modules
    711 text files.
    708 unique files.
    167 files ignored.

github.com/AlDanial/cloc v 1.88 T=1.78 s (390.8 files/s, 128693.5 lines/s)
-----
Language               files      blank      comment      code
-----
JavaScript              433        15781       17436       136909
HTML                    166         230         39         52765
TypeScript              18          221         76         1540
Markdown                13          305          0         1086
CSS                     11          142         17          910
Vuejs Component         27           50         12          644
JSON                    12           0           0          339
YAML                     3            9           2          139
XML                     10          604          0          110
Bourne Shell            2            16         17           85
Bourne Again Shell      2            4           1           16
-----
SUM:                    697        17362       17600       194543
-----
```

数据获取：项目bug数量

- 使用GitHub Issues统计bugs数量
- 1. 使用bug为关键词搜索这个项目中的所有可能是bug的label
 - 不同项目标识是bug的label不一样
 - React: 
 - Vue 
 - Ant-design: 
- 2. 使用每个label作为关键词，搜索带有这个label的issue的数量
- 3. 每个label的issue的数量之和就是这个项目的所有的issues的数量

数据获取：项目bug数量：运行示例

以bug为关键词搜索仓库
的label

对每一个label，搜索包含
它的label的issue数量

加和，就是仓库的所有
bugs

```
[Mon Dec 7 15:30:55 2020] Possible bug labels for react are: Status: Unconfirmed, Type: Bug
[Mon Dec 7 15:30:56 2020] react has 9755 issues.
[Mon Dec 7 15:30:57 2020] Status: Unconfirmed has 685 issues.
[Mon Dec 7 15:31:00 2020] Type: Bug has 670 issues.
[Mon Dec 7 15:31:00 2020] react has 1355 bug issues.
```

数据获取：统计单元测试数量

- 真正运行每个项目的测试用例，分析输出结果得出测试结果
- 原因：
 - 框架无法直接获得一个项目的测试用例数量
 - 静态分析it("", () => {})模式，结果不准确

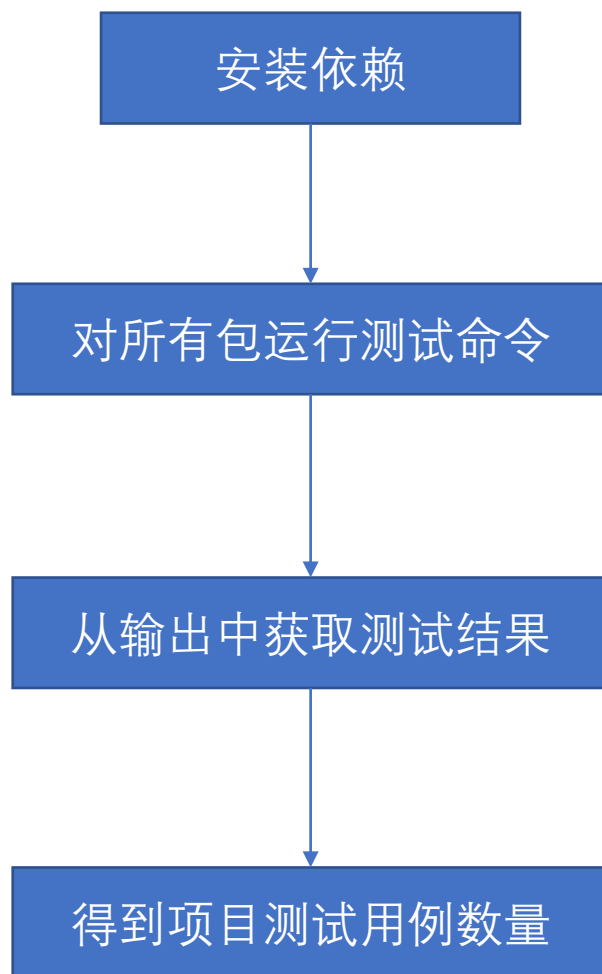
```
Test Suites: 257 passed, 257 total
Tests:       26 skipped, 6974 passed, 7000 total
Snapshots:   83 passed, 83 total
Time:        201.442s
Ran all test suites.
Done in 208.46s.
```

真实情况

```
[Mon Nov 30 16:48:19 2020] react has 4056 unit tests in total.
Time used: 555.0920205116272 s
```

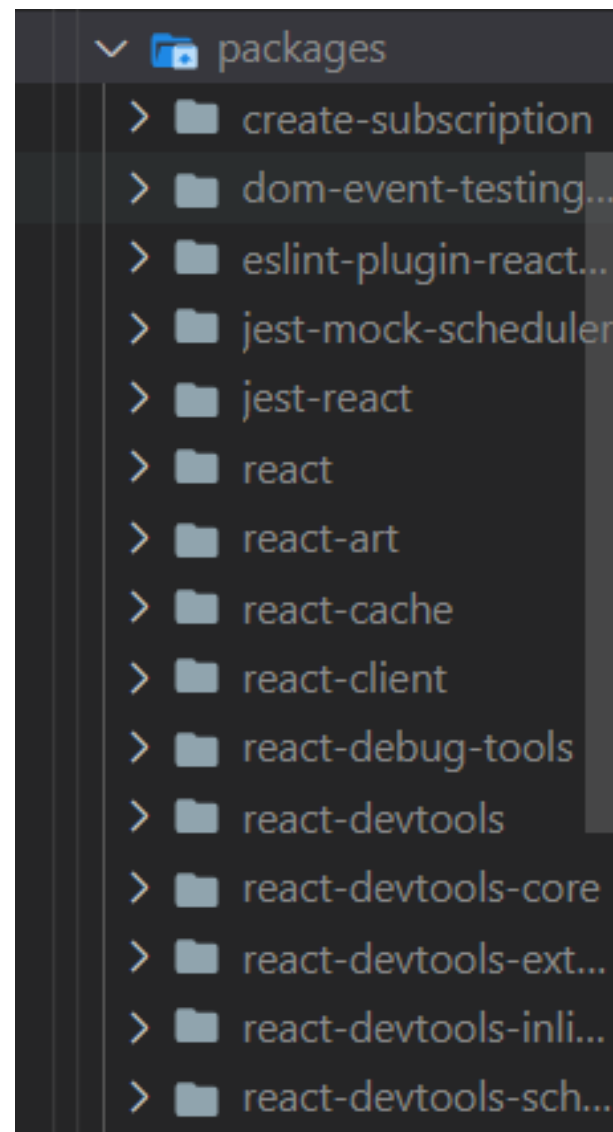
静态方法结果

数据获取：自动运行项目的测试用例



数据获取： monorepo

- Monorepo： 一个仓库有多个包
- lerna是用来管理monorepo仓库的最常用工具
- 相关命令：
 - bootstrap： 安装所有包的依赖， 链接包之间的依赖关系
 - List -json： 列出项目中的所有包的名字和路径



React是一个monorepo，由多个子包组成

数据获取： 安装依赖



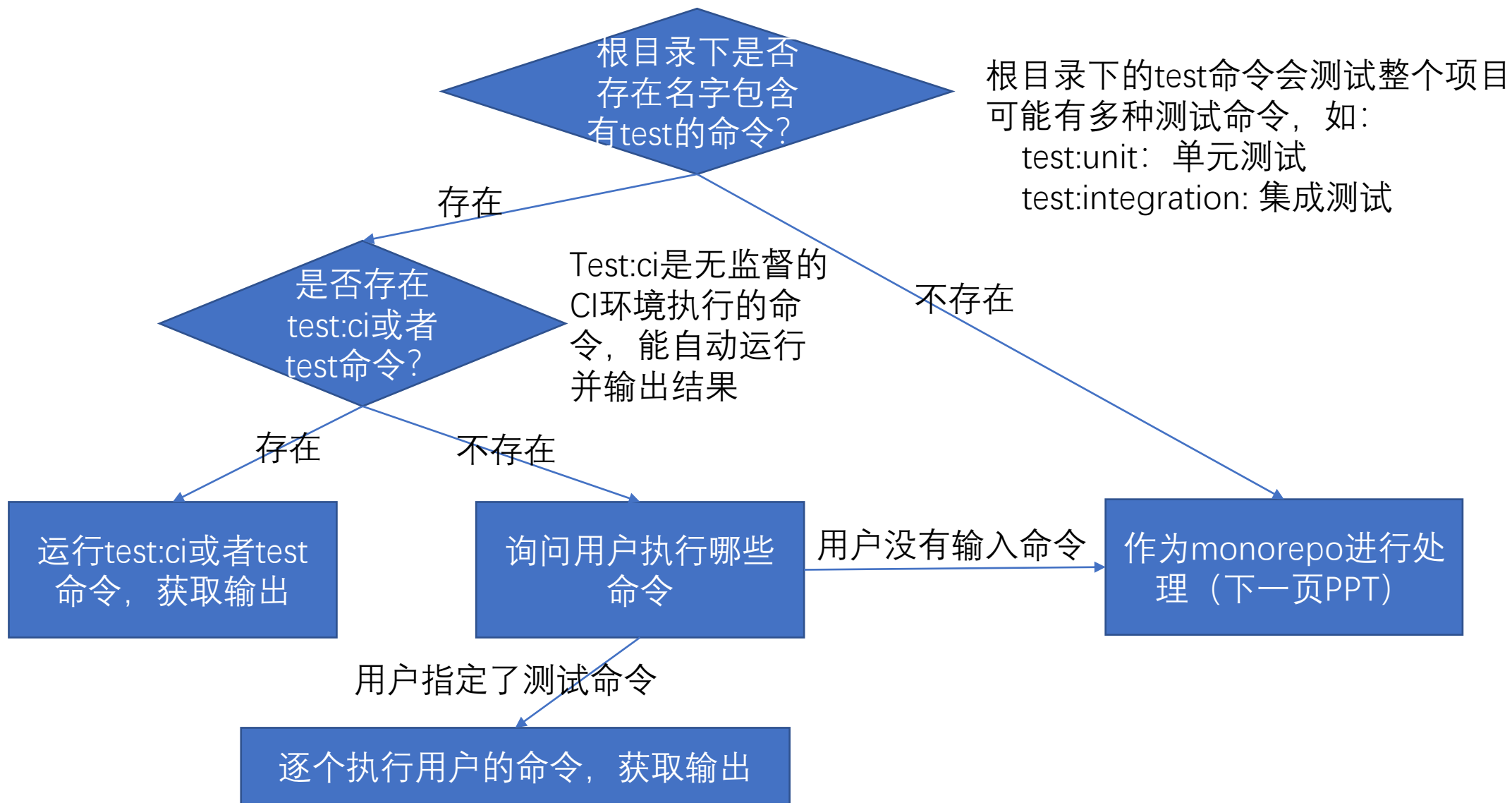
存在yarn.lock: yarn
存在package-lock.json: npm
只存在package.json: 都可以，用npm
不存在package.json: 不是代码项目，跳过

Yarn: yarn
Npm: npm install

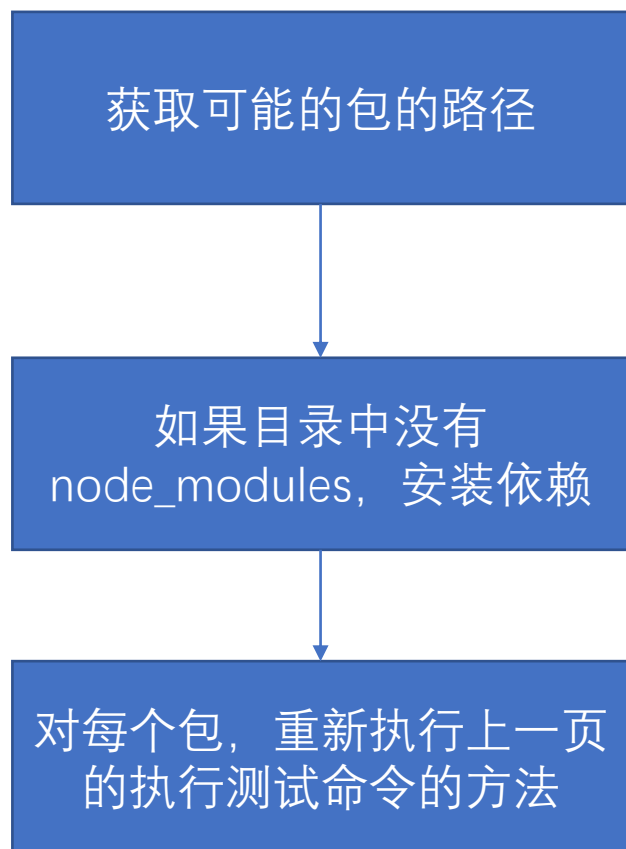
bootstrap: 初始化

安装所有子包的依赖，建立包之间的依赖关系

数据获取：对所有包运行测试命令



数据获取：对monorepo执行测试



如果使用的lerna, 执行lerna list --json
如果没使用, 认为packages或者pkgs目录下的包为包
如果不存在, 则认为此包不存在子包, 整个包的测试数量为0

一般不会出现这种情况

没有monorepo中的monorepo, 不会嵌套

数据获取：从输出中获取测试结果

- 正则表达式
- 对测试输出的每一行匹配预先定义的模式
- 并从中提取出测试总量数据
- 所有输出的和作为项目的单元测试用例数量

```
# Tests: ..... 26 skipped, 6974 passed, 7000 total
"jest_with_skipped": ("Tests:(.*)([0-9]+) skipped, ([0-9]+) passed, ([0-9]+) total", 4),
# Tests: ..... 6974 passed, 7000 total
"jest_without_skipped": ("Tests:(.*)([0-9]+) passed, ([0-9]+) total", 3),
# Tests: ..... 1 failed, 237 passed, 238 total
"jest_with_failed": ("Tests:(.*)([0-9]+) failed, ([0-9]+) passed, ([0-9]+) total", 4),
# Tests: ..... 2 skipped, 2 todo, 59 passed, 63 total
"jest_with_todo": ("Tests:(.*)([0-9]+) skipped, ([0-9]+) todo, ([0-9]+) passed, ([0-9]+) total", 5),
# TOTAL: 1113 SUCCESS
"karma_headless_chrome": ("TOTAL: ([0-9]+) SUCCESS", 1),
# ✓ 1271 tests completed
"karma": ("✓ ([0-9]+) tests completed", 1),
# 153 specs, 0 failures
"jasmine": ("([0-9]+) specs, ([0-9]+) failures", 1),
# mocha has no total count, but 4 separate output
# 2375 passing
"mocha_pass": ("(.*)([0-9]+) passing", 2),
# 399 skipped
"mocha_skipped": ("(.*)([0-9]+) skipped", 2),
# 5 failing as expected
"mocha_failing_as_expected": ("(.*)([0-9]+) failing as expected", 2),
# 552 unexpected failing
"mocha_unexpected_failing": ("(.*)([0-9]+) unexpected failing", 2),
# 125 tests passed
"ava": ("(.*)([0-9]+) tests passed", 2),
# 76 passed
"ava_2": ("(.*)([0-9]+) passed", 2),
# # tests 164
"nanohtml": ("# tests ([0-9]+)", 1),
# - - total: ..... 148
"tap": ("(.*)total:(.*)([0-9]+)", 3)
```


数据获取： 运行示例

```
< python count_ut_run.py
[Tue Dec 15 00:37:21 2020] Run npm install in projects/redux-orm
[Tue Dec 15 00:37:52 2020] Run npm run test in projects/redux-orm
[Tue Dec 15 00:37:58 2020] Tests:          278 passed, 278 total
[Tue Dec 15 00:37:58 2020] The above line might be a output of jest_wi
thout_skipped
[Tue Dec 15 00:37:58 2020] Count: 278
[Tue Dec 15 00:37:58 2020] redux-orm has 278 unit tests
```

Redux-orm自动运行测试并获取测试结果

数据获取：意外情况

- 很多，几乎每个项目都得手动检查，手动运行测试或者修改流程
- 测试无法运行、测试需要前置依赖等情况：设置为-1

项目	解决方案	原因
tui.editor	按照readme安装依赖；手动看每个项目的package.json	安装依赖比较复杂，npm install和lerna bootstrap不能用；每个测试持续运行不停止
vue-storefront	结果+17	e2e测试使用的cypress，但是test:e2e:ci无法直接运行，手动数了下，在单元测试的结果上+17
backbone.marionette	自己跑一遍yarn test	scripts里有watch参数，命令不自动停止
ng-zorro-antd	结果/2	输出了两个一样的测试输出……
zeroclipboard	-1	依赖flex，依赖java，说找不到JDK，但是我本地装了的……
ViewUI	-1	可能是node版本不兼容
userbase	-1	需要aws账号
sift.js	先build再测试	直接测试的构建好的版本
23207943	-1	用的coffeescript，不能直接跑测试
53618401	-1	Test命令错误码67，不知道为什么
85697697	-1	测的是md里的语法错误
93942470	76	不知道为什么测出3096……
259488034	-1	npm install错误128

数据分析

- 数据说明与数据预处理
- RQ1: 测试用例数量与整体代码量的关系
- RQ2: 测试用例数量与bug数量的关系
- RQ3: 使用TypeScript是否能降低bug的数量
- 多变量分析与样本内部结构

采集的数据

- 样本：GitHub上Star最多的200个前端领域的代码

重要字段	解释
has_code	是否包含代码
stargazers_count	项目获得的星星数(按此字段排序)
js_lines	JavaScript代码行数
ts_lines	TypeScript代码行数
issues_count	issue的数量
bugs_issues_count	bug类型issue的数量
unit_test_count	测试用例数量

id	name	html_url	has_code	stargazers_count	package_name	js_lines	ts_lines	issues_count	bugs_count	unit_test_count
11730342	vue	https://gi	TRUE	176469	yarn	136909	1540	9199	371	1539
10270250	react	https://gi	TRUE	160340	yarn	280793	685	9770	1367	7004
107111421	Front-End	https://gi	FALSE	45818		0	0	116	15	0
12256376	ionic-fram	https://gi	TRUE	42485	npm	10916	42769	18292	712	0
10865436	frontend-ch	https://gi	FALSE	28067		0	0	93	0	0
45512989	gold-miner	https://gi	FALSE	27800		0	0	4690	0	0
64355429	iview	https://gi	TRUE	23635	yarn	41784	1323	5468	137	45
79527893	hyperapp	https://gi	TRUE	18212	both	565	0	537	52	2
181807583	fe-intervi	https://gi	FALSE	15009		0	0	3264	0	0
110058856	awesome-ch	https://gi	FALSE	14219		0	0	25	0	0
104172832	ant-design	https://gi	TRUE	13119	yarn	21563	30212	2672	59	517
139596105	Front-End	https://gi	FALSE	12734		0	0	23	3	0
79723839	bit	https://gi	TRUE	12519	npm	4924	113125	1005	406	0
39759882	tui.editor	https://gi	TRUE	12099	npm	52314	7627	790	294	2083
85541218	weekly	https://gi	FALSE	10553		0	0	205	0	0
14106970	tachyons	https://gi	TRUE	10255	both	0	0	389	6	0
15720445	marko	https://gi	TRUE	9860	npm	47407	0	936	272	2779
63760777	material-ch	https://gi	TRUE	9392	npm	49897	0	236	16	0
103071620	tui.calendr	https://gi	TRUE	8641	npm	32152	518	482	91	347
13737149	gremlins.js	https://gi	TRUE	8519	npm	1548	0	77	1	37
24817507	frontend-sh	https://gi	FALSE	8374		0	0	21	0	0
6561551	perfect-sch	https://gi	TRUE	8202	npm	4569	58	725	23	0
100346588	vue-storef	https://gi	TRUE	7756	yarn	4224	33712	2351	931	673
72066935	gdbgui	https://gi	TRUE	7711	yarn	452	7276	252	12	2
52210728	growth-ebc	https://gi	FALSE	7277		0	0	21	0	0
66709063	Awsome-Fro	https://gi	FALSE	7130		0	0	9	1	0
2965621	backbone.n	https://gi	TRUE	7126	yarn	16640	0	1765	154	1045
99705100	ng-zorro-a	https://gi	TRUE	7098	both	1219	104308	3791	418	1706
4012085	zeroclipbo	https://gi	TRUE	6643	both	8670	0	463	90	-1

部分数据展示

数据预处理

- 样本：has_code为False的项目直接不考虑。
- 样本量：120

重要字段	解释
stargazers_count	项目获得的星星数(按此字段排序)
js_lines	JavaScript代码行数
ts_lines	TypeScript代码行数
issues_count	issue的数量
bugs_issues_count	bug类型issue的数量
unit_test_count	测试用例数量

id	name	html_url	stargazers_count	package_name	js_lines	ts_lines	issues_count	bugs_count	unit_test_count
11730342	vue	https://gi	176469	yarn	136909	1540	9199	371	1539
10270250	react	https://gi	160340	yarn	280793	685	9770	1367	7004
12256376	ionic-fran	https://gi	42485	npm	10916	42769	18292	712	0
64355429	iview	https://gi	23635	yarn	41784	1323	5468	137	45
79527893	hyperapp	https://gi	18212	both	565	0	537	52	2
104172832	ant-design	https://gi	13119	yarn	21563	30212	2672	59	517
79723839	bit	https://gi	12519	npm	4924	1E+05	1005	406	0
39759882	tui.editor	https://gi	12099	npm	52314	7627	790	294	2083
14106970	tachyons	https://gi	10255	both	0	0	389	6	0
15720445	marko	https://gi	9860	npm	47407	0	936	272	2779
63760777	material-ch	https://gi	9392	npm	49897	0	236	16	0
103071620	tui.calend	https://gi	8641	npm	32152	518	482	91	347
13737149	gremlins.c	https://gi	8519	npm	1548	0	77	1	37
6561551	perfect-sc	https://gi	8202	npm	4569	58	725	23	0
100346588	vue-storef	https://gi	7756	yarn	4224	33712	2351	931	673
72066935	gdbgui	https://gi	7711	yarn	452	7276	252	12	2
2965621	backbone.n	https://gi	7126	yarn	16640	0	1765	154	1045
99705100	ng-zorro-a	https://gi	7098	both	1219	1E+05	3791	418	1706
4012085	zeroclipbo	https://gi	6643	both	8670	0	463	90	-1
122026655	vue-enterp	https://gi	6537	yarn	1995	0	170	9	-1
18224920	hospitalru	https://gi	5549	both	159	27966	905	161	803
221822577	Hippy	https://gi	5422	npm	11945	6505	127	27	125
183874244	amis	https://gi	5338	both	3444	71625	717	11	238
90549237	nerv	https://gi	5268	yarn	12093	4384	133	2	7
75785147	webpacker	https://gi	4946	yarn	1448	32	1614	44	55
66830040	san	https://gi	4302	npm	37482	559	390	9	682
20839462	imba	https://gi	4066	yarn	29270	1469	394	68	285
15290592	reagent	https://gi	4039	npm	124	0	340	17	0
76130347	create-res	https://gi	3774	both	1426	128	295	7	0

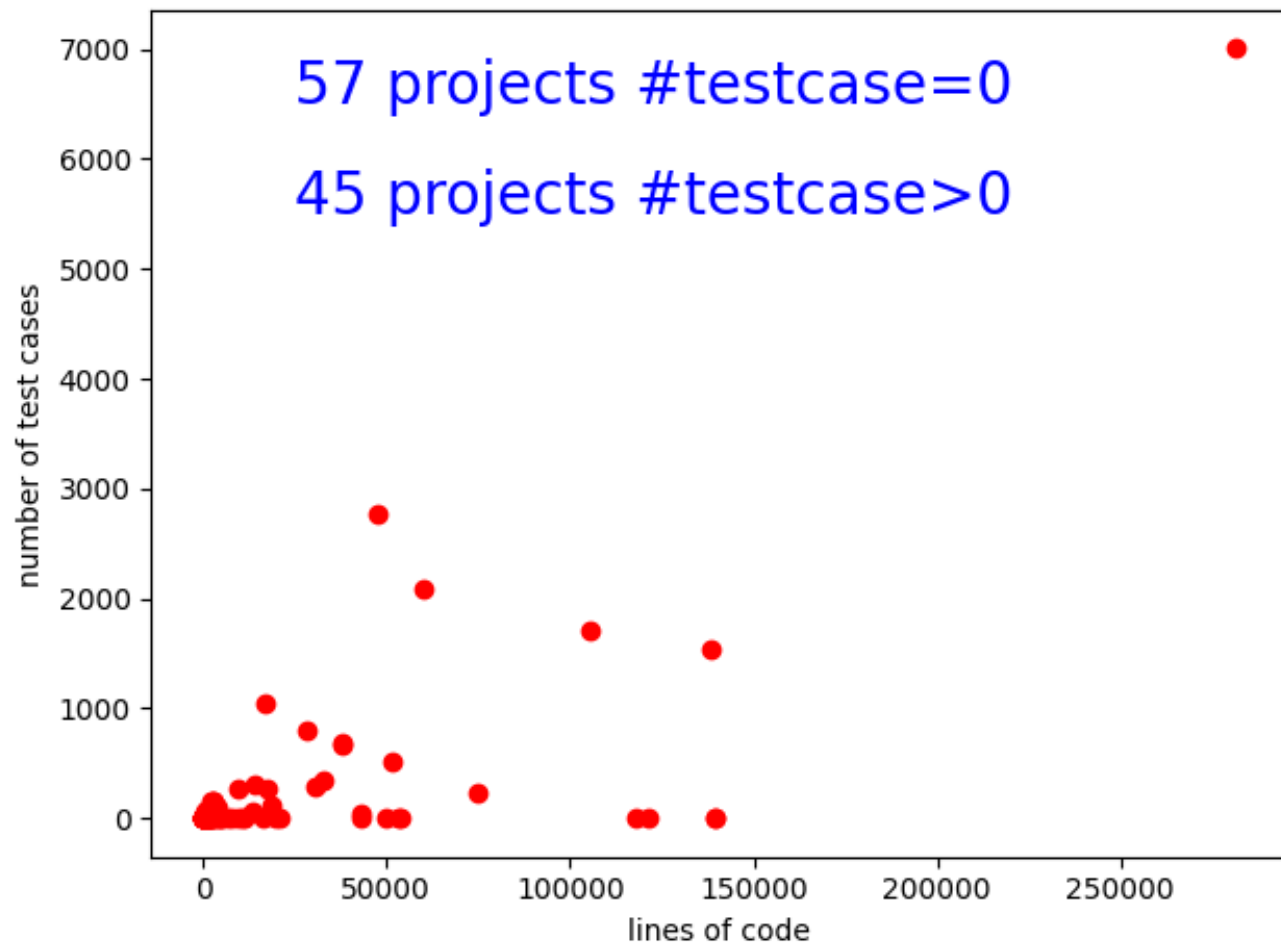
预处理后的数据

数据分析

- 数据说明与数据预处理
- RQ1: 测试用例数量与整体代码量的关系
- RQ2: 测试用例数量与bug数量的关系
- RQ3: 使用TypeScript是否能降低bug的数量
- 多变量分析与样本内部结构

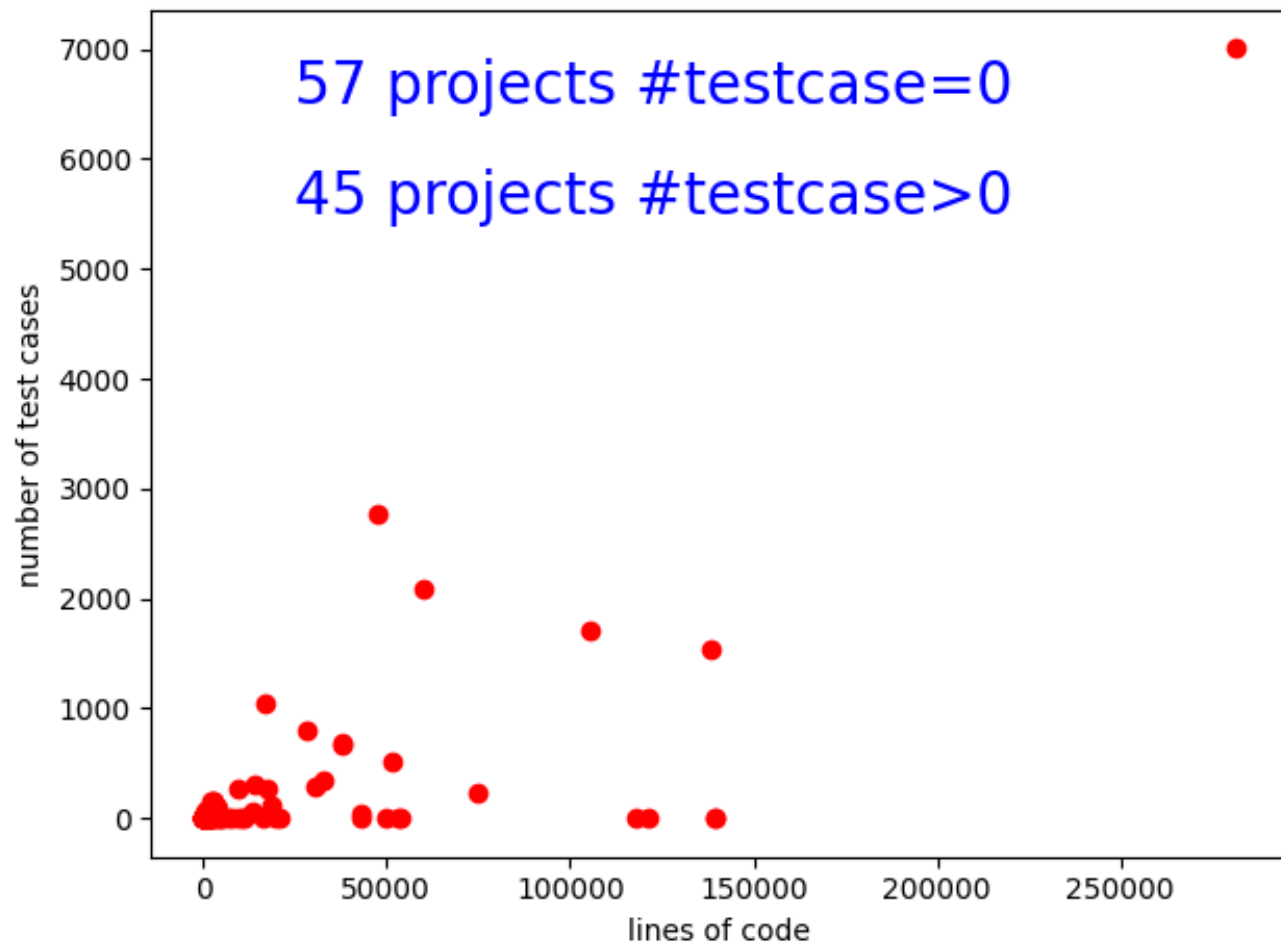
概览

- 去掉意外情况的数据
 - 剩余102个样本
- 整体代码量 = js + ts
- 绘制散点图，看看大概的分布情况
- 难以直接观察出分布规律
 - 样本点集中在原点
 - 有57个项目无测试用例



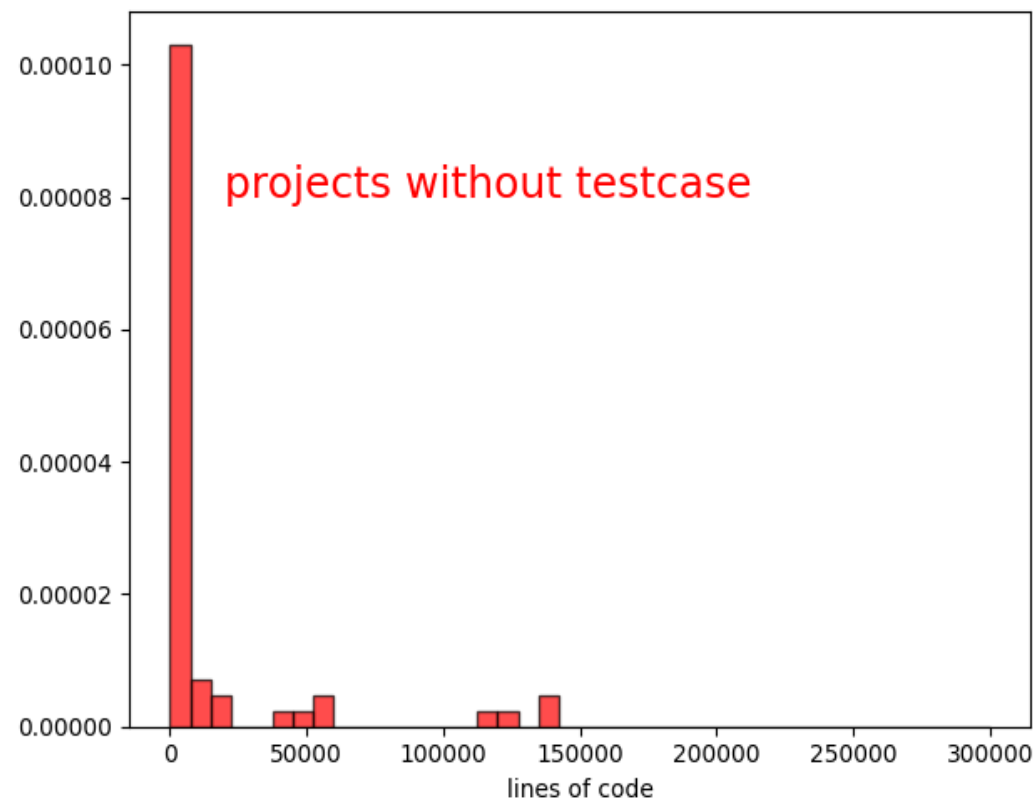
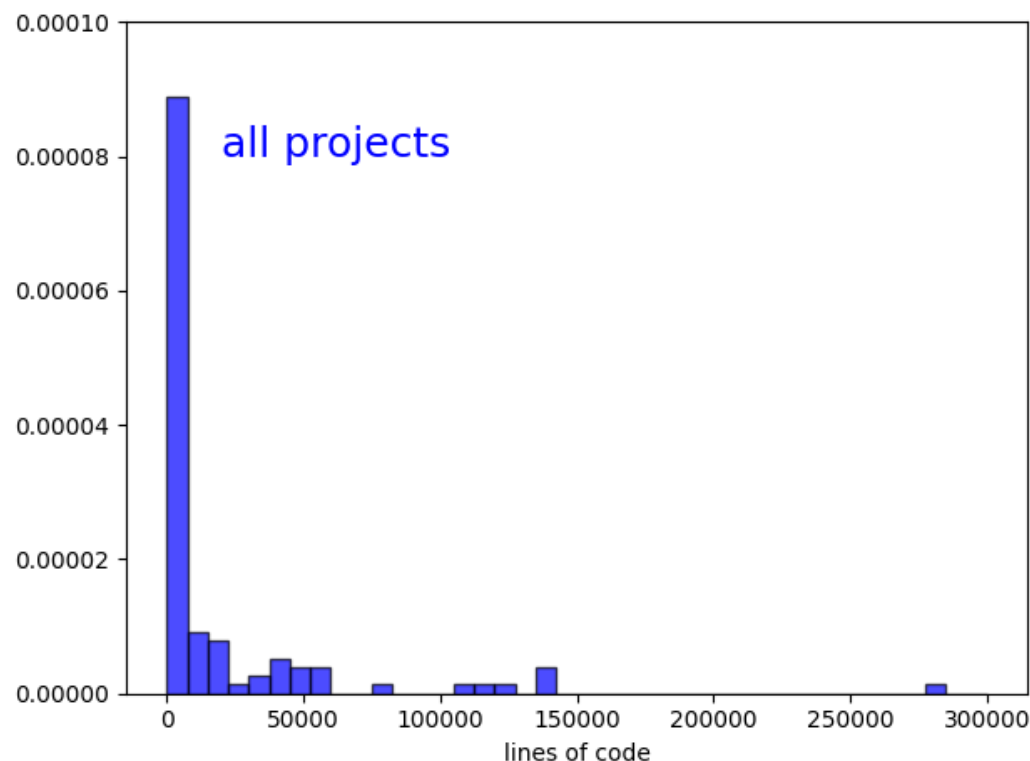
相关系数

- 相关系数可用来衡量变量之间线性相关程度。
 - 相关系数0.70
- 样本分布很不均匀，参考价值低！
 - 举个例子：{(0,0), (0,1), (1,0), (1,1)} 相关系数为0，{(0,0), (0,1), (1,0), (1,1), {10, 10}}相关系数为0.986
- 解决方式：将102个样本分成无测试用例和有测试用例两类。



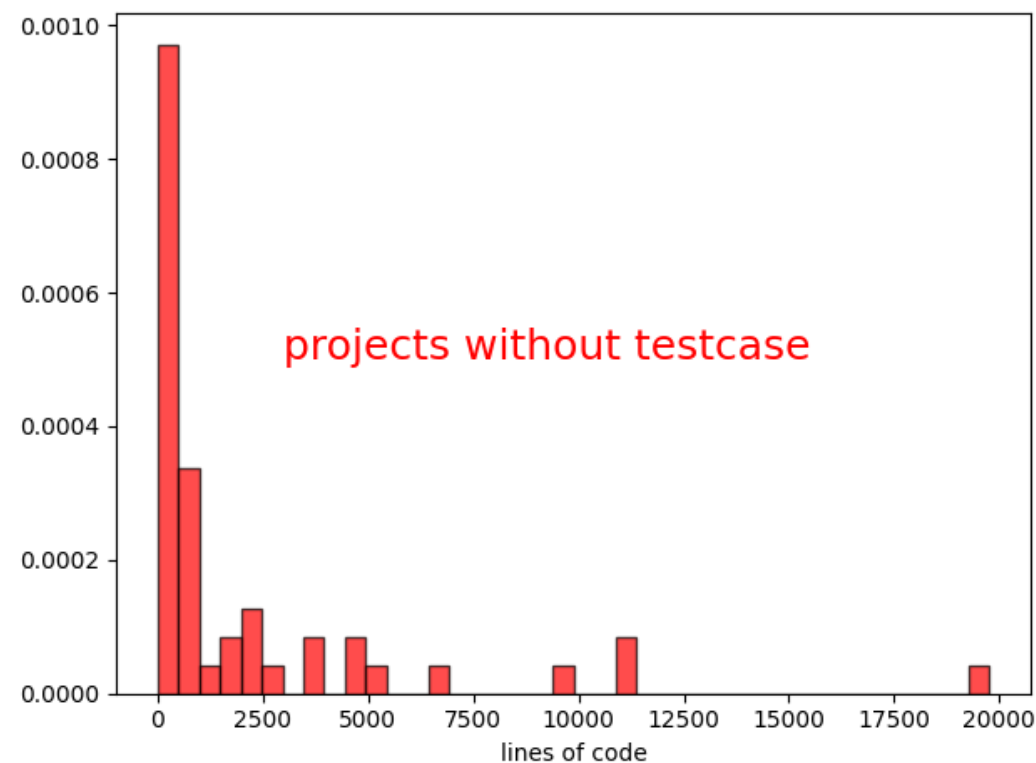
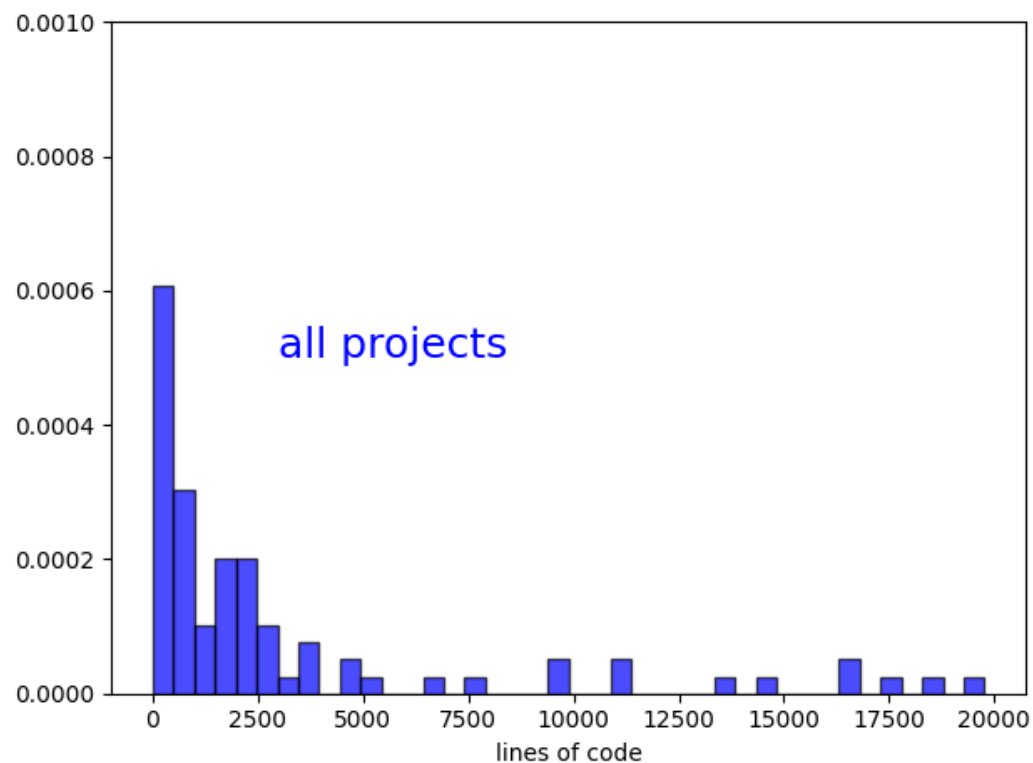
只考虑无测试用例的项目

- 所有项目代码行数的均值：19302（去掉最大值后均值为16706）
- 测试用例数量为0的项目的代码行数的均值：14749
- 绘制频率分布直方图，观察分布情况。



只考虑无测试用例的项目

- 只看代码量小于20000行的项目
- 代码量很小的项目没有测试用例的几率还是很大的。

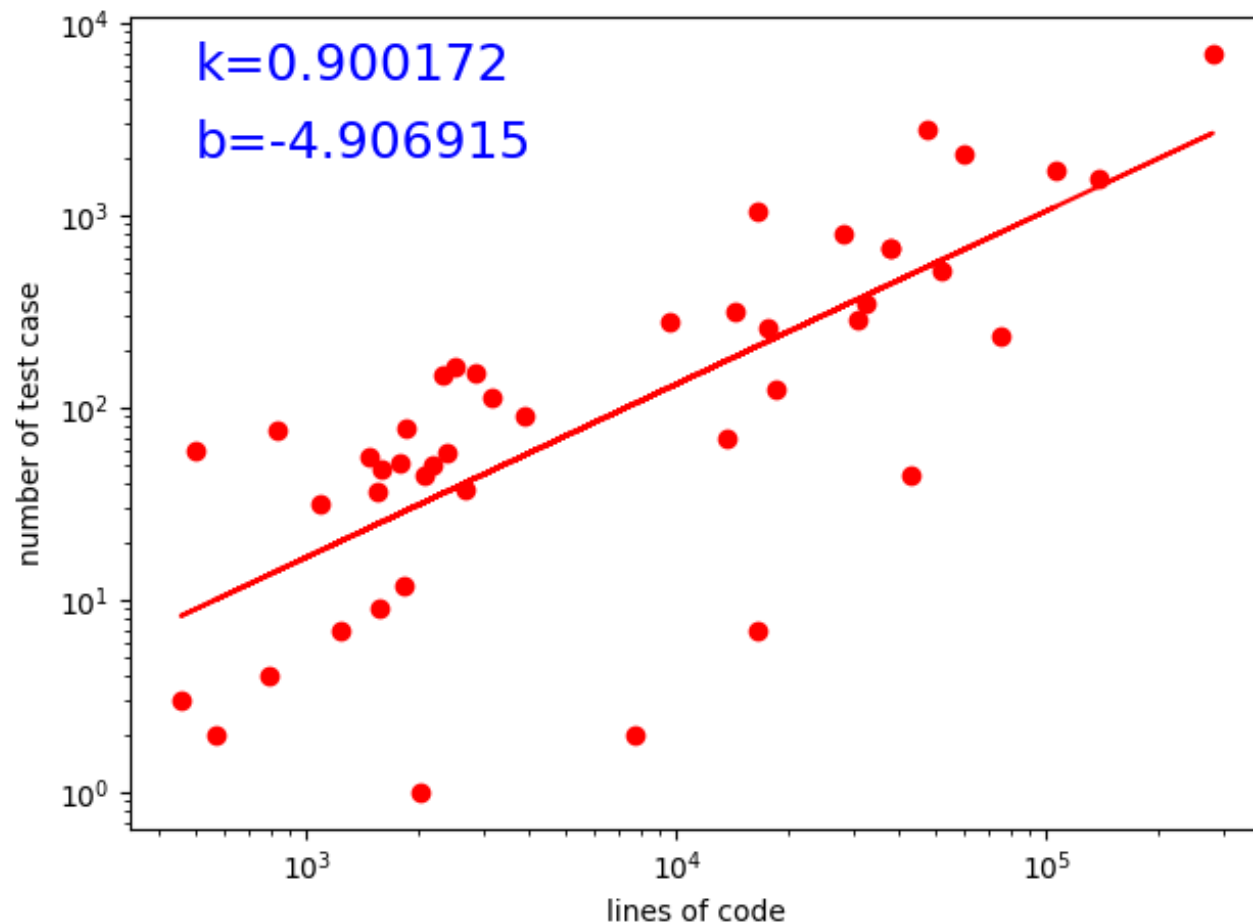


只考虑无测试用例的项目

- 注意到：有些代码量很大的项目也没有测试用例：
 - <https://github.com/home-assistant/frontend> (家庭助理, 管理家居的项目)
 - <https://github.com/kottans/frontend> (前端课程)
- 可能的原因：
 - 某些项目并不用实际部署, 因此不需要测试。
 - 主要提供某些个性化的功能, 只需要页面显示正确就行, 没必要测试。
- 结论：项目代码量很小的项目 (≤ 500 行) 很可能没有测试用例。

只考虑有测试用例的项目

- 处理方式：x轴和y轴同时取对数，可观察到存在明显线性关系。
 - 相关系数：0.7484
- 线性回归：
 - 斜率：0.90017179
 - 截距：-4.90691475

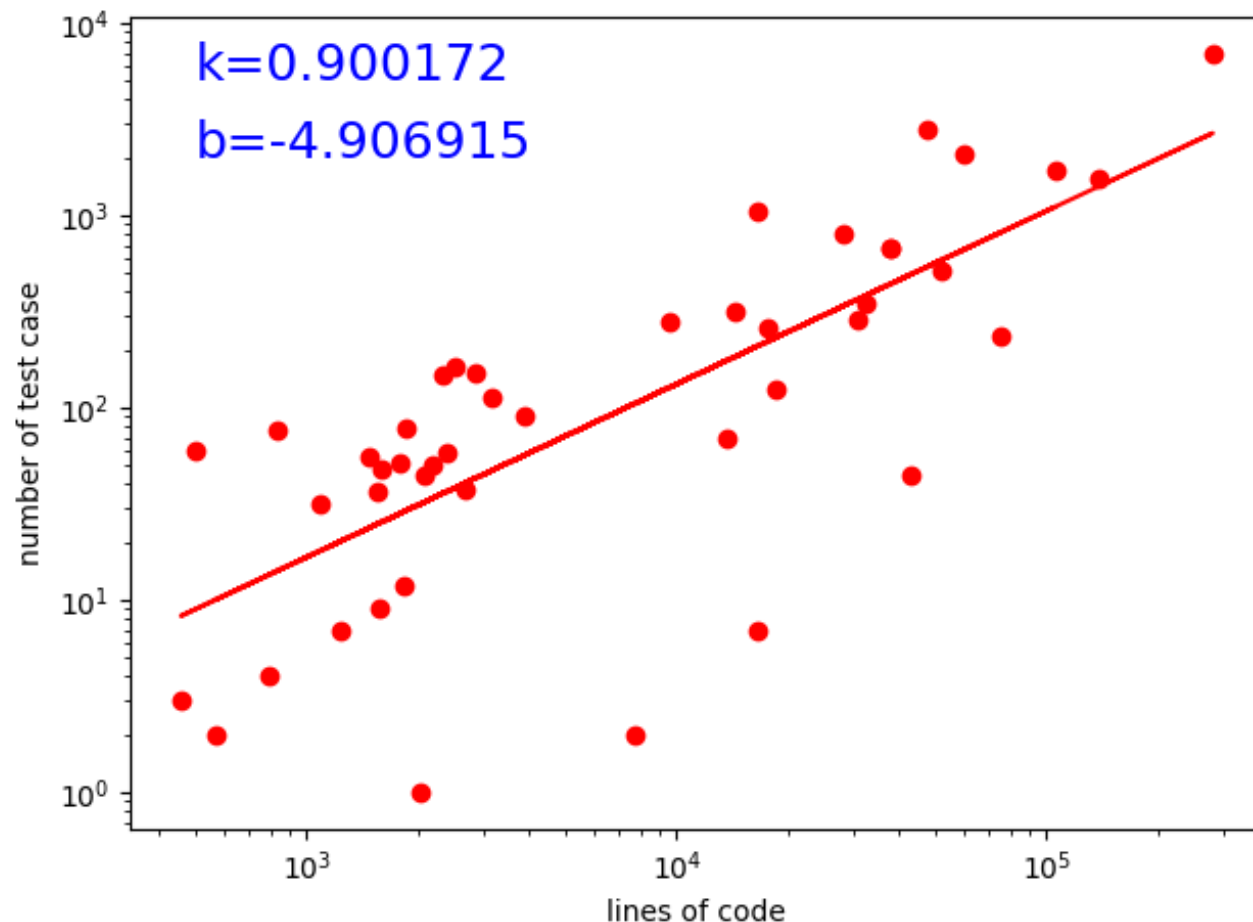


只考虑有测试用例的项目

- 结论：有测试用例的软件的测试用例数量和软件代码行数存在显著关系：

$$y = 0.0335x^{0.9}$$

- x是代码行数
- y是测试用例的数量



RQ1: 测试用例的数量和整体代码量之间有什么关系?

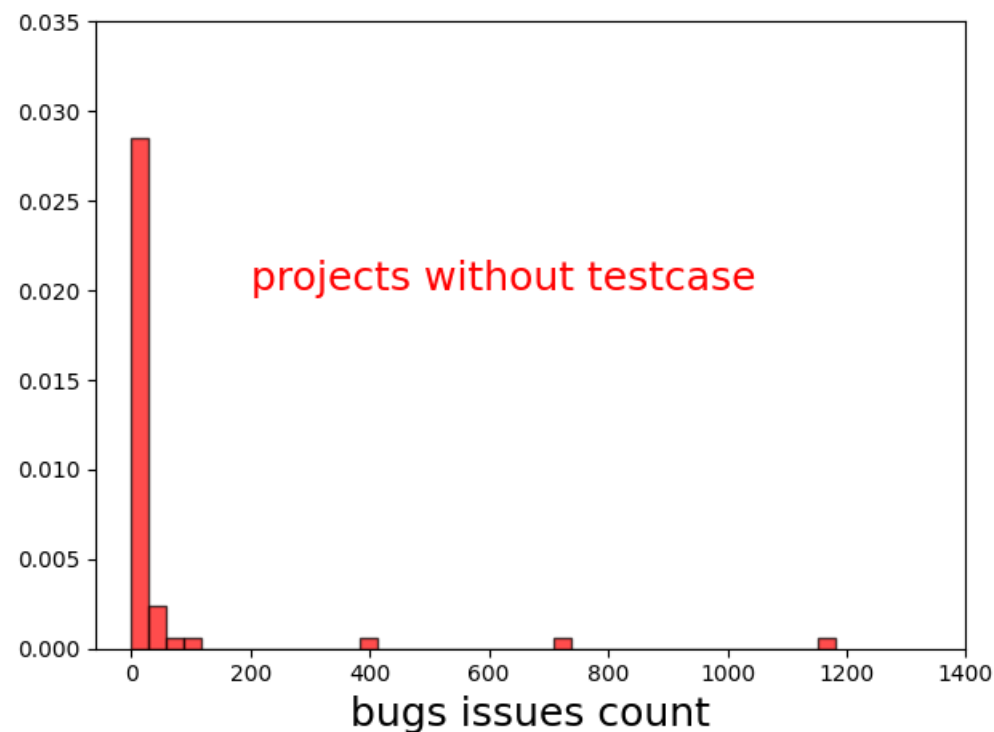
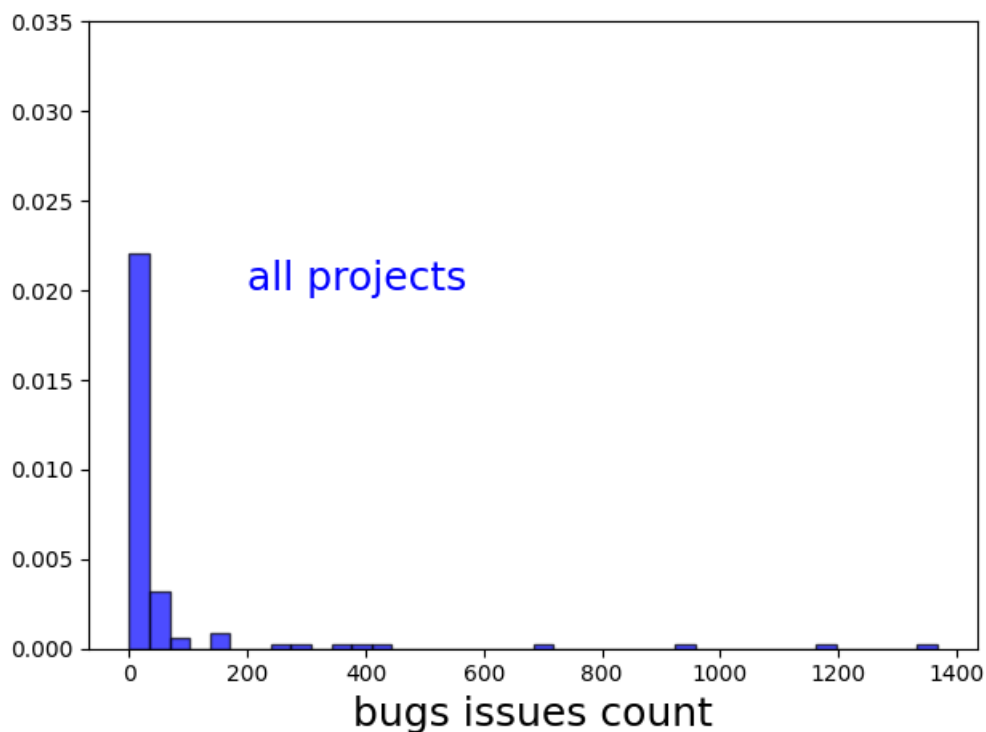
- 代码量很小的项目没有测试用例的可能性很大, 而对于代码量较大的项目, 无法确定, 这和项目本身性质有关。
- 考虑到没有测试用例的项目很多是因为根本没必要测试, 因此如果要分析对项目进行测试的情况, 应该只考虑有测试用例的样本, 分析结果显示测试用例数量与整体代码量之间存在亚线性关系。

数据分析

- 数据说明与数据预处理
- RQ1: 测试用例数量与整体代码量的关系
- RQ2: 测试用例数量与bug数量的关系
- RQ3: 使用TypeScript是否能降低bug的数量
- 多变量分析与样本内部结构

只考虑无测试用例的项目

- 采用类似的处理方式，分成无测试用例和有测试用例两种情况。
- 所有项目bug issue数量的均值：73.94
- 没有测试用例的项目的bug issue数量的均值：49.12



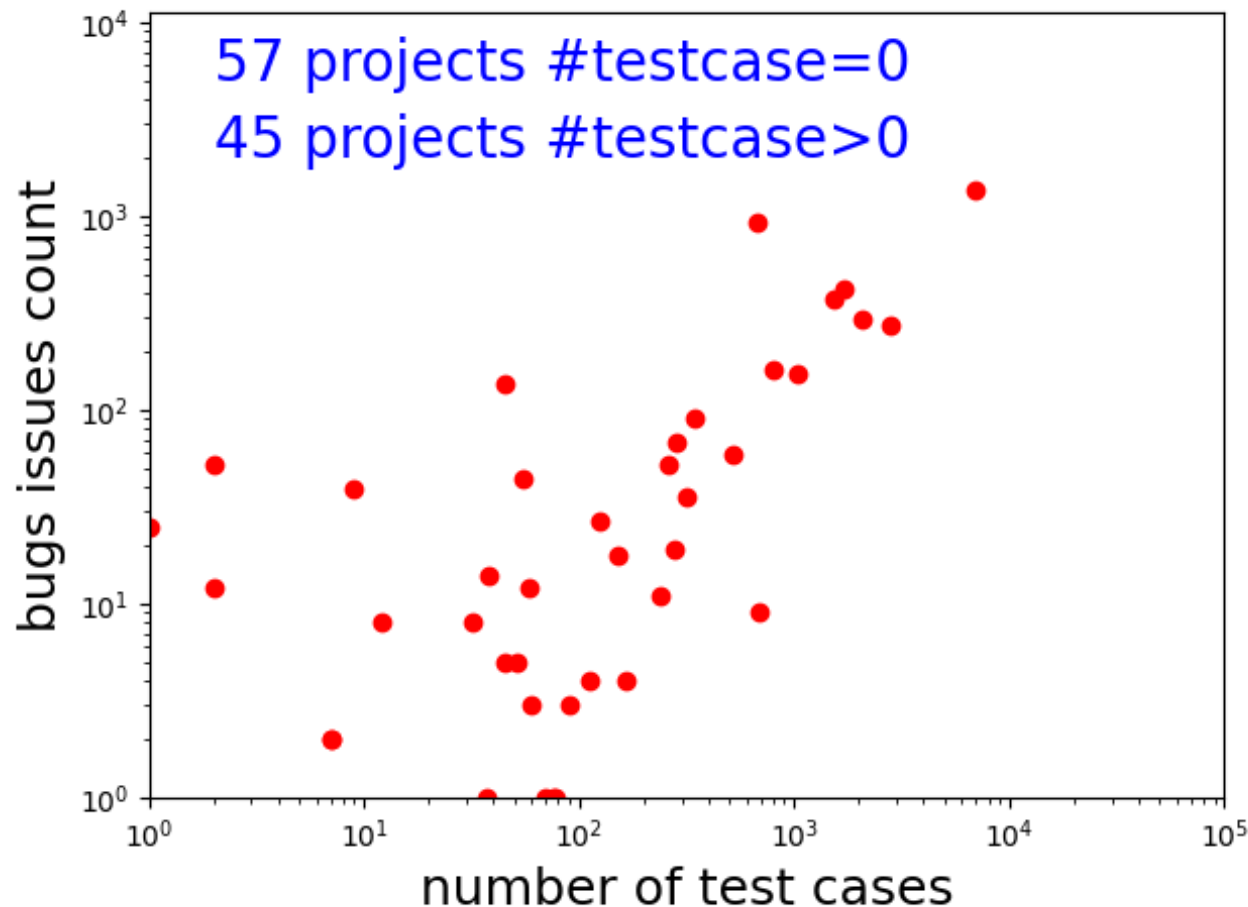
只考虑有测试用例的项目

- 处理方式：x轴和y轴同时取对数，可观察到存在一定线性关系。

- 相关系数：0.57

$$y = 1.6x^{0.55}$$

- y是bug issue数量
- x是测试用例数量



RQ2: 测试用例数量与bug数量的关系是怎样的?

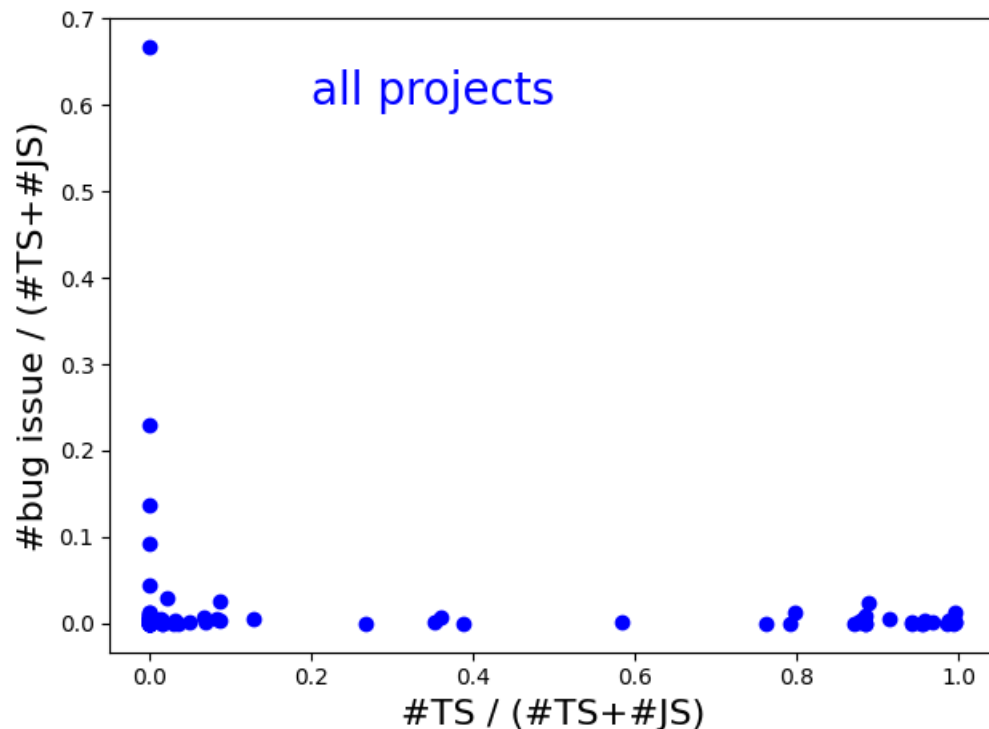
- 测试用例数量多的项目bug issue数量也比较多 ✓
- 增加测试用例会使得bug变多 ✗

数据分析

- 数据说明与数据预处理
- RQ1: 测试用例数量与整体代码量的关系
- RQ2: 测试用例数量与bug数量的关系
- RQ3: 使用TypeScript是否能降低bug的数量
- 多变量分析与样本内部结构

理想与现实

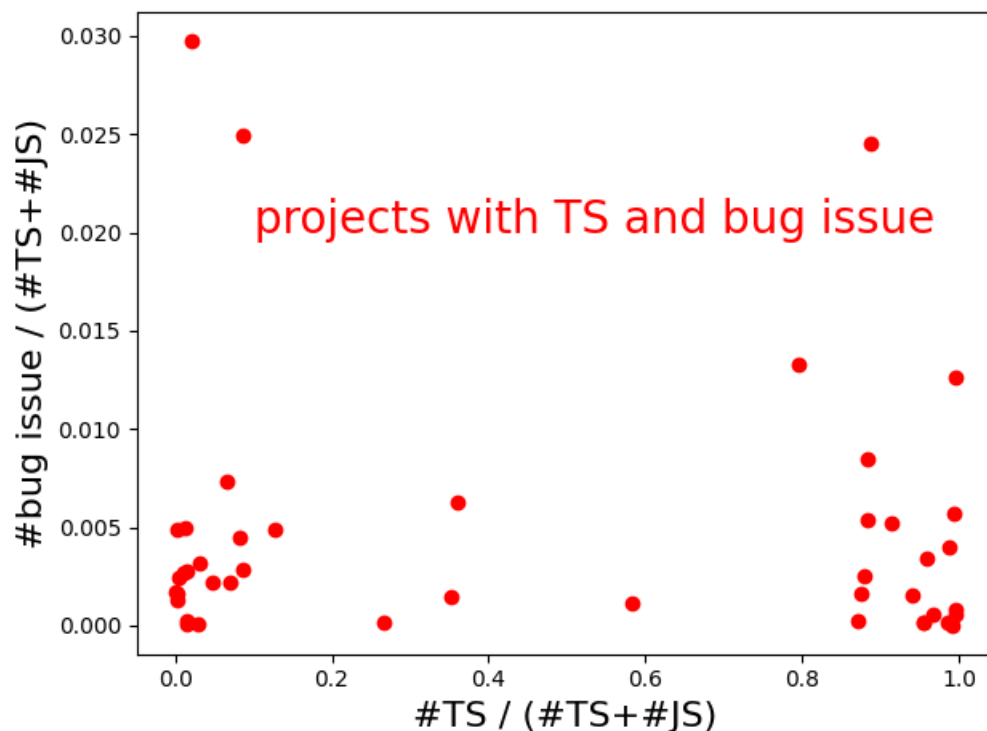
- TS的代码量占总体代码量的比例和bug数量/总代码量之间的关系。
 - 或许随着TS代码占比的提高bug数量会减少?
 - but...



- 相关系数-0.0845
- 极端的样本很多

理想与现实

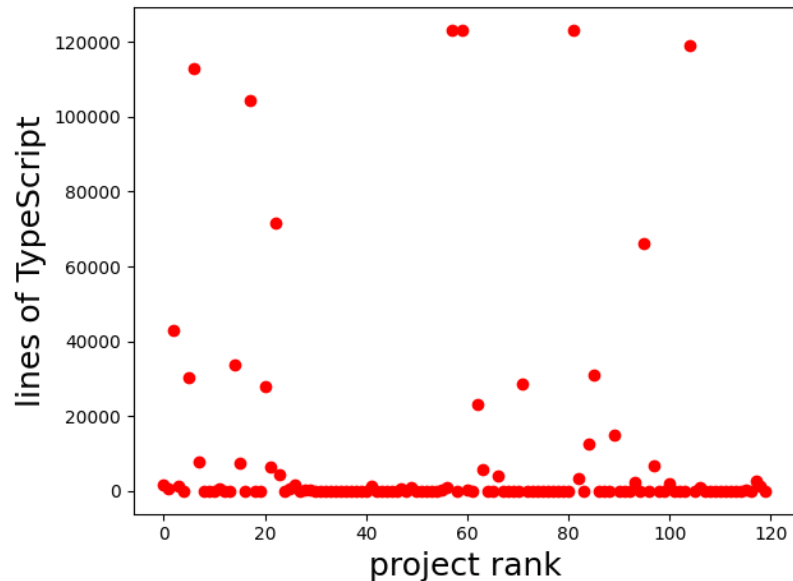
- 或许极端样本影响了相关系数和散点图的绘制
 - 只考虑有TS代码并且有bug issue的项目
 - but...



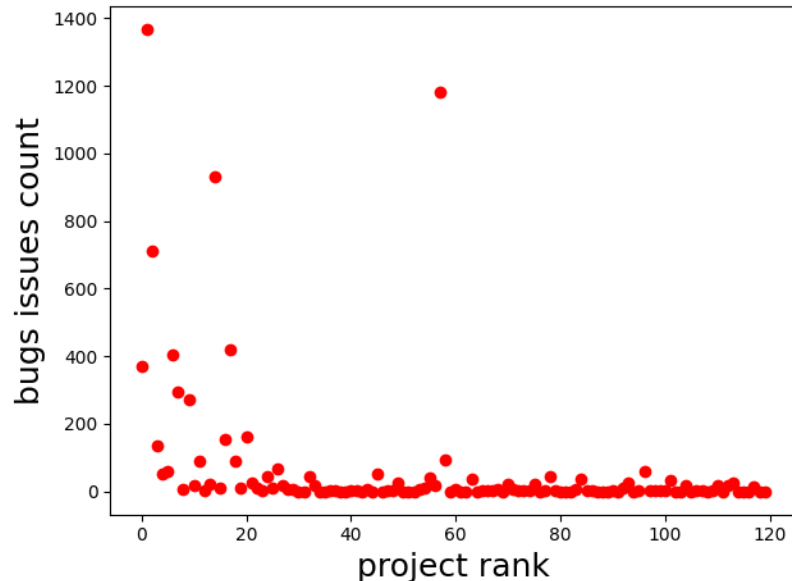
- 相关系数-0.052
- 样本集中在TS占比为0和TS占比为1的直线附近

与其他特征分布情况的对比

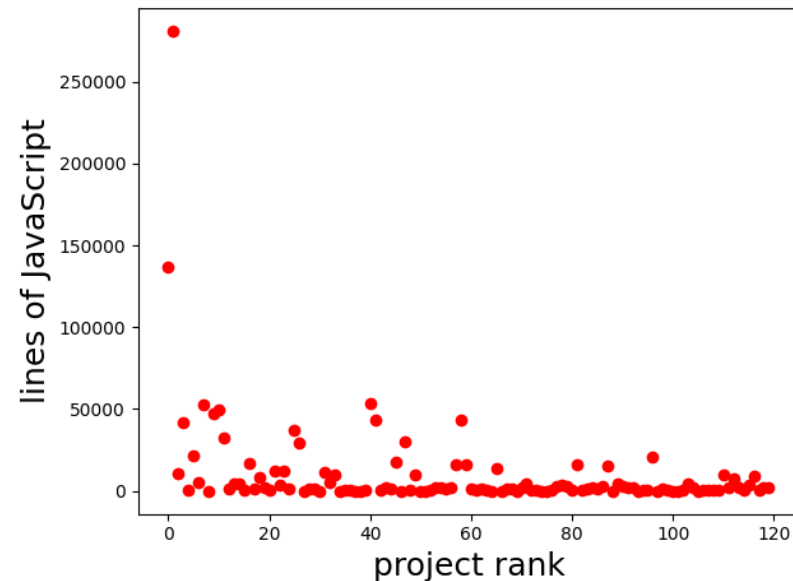
- 或许只是运气不好
 - 或许只是我们所期待的结果和实际不符



TypeScript



bug issue



JavaScript

RQ3: 使用TypeScript是否能降低bug的数量?

- 未观测到显著关系

数据分析

- 数据说明与数据预处理
- RQ1: 测试用例数量与整体代码量的关系
- RQ2: 测试用例数量与bug数量的关系
- RQ3: 使用TypeScript是否能降低bug的数量
- 多变量分析与样本内部结构

数据分析结果的进一步思考

- 实际情况中，潜在影响因素很多
 - 项目本身性质
 - 项目的社区
 - 软件开发的难易
- 3个RQ其实后面能对其作用的因素很多
- 没有绝对的控制变量对比实验
- 单单拎出两个变量进行对比，得出某种分布关系，然后强加自己的理解，不合理。

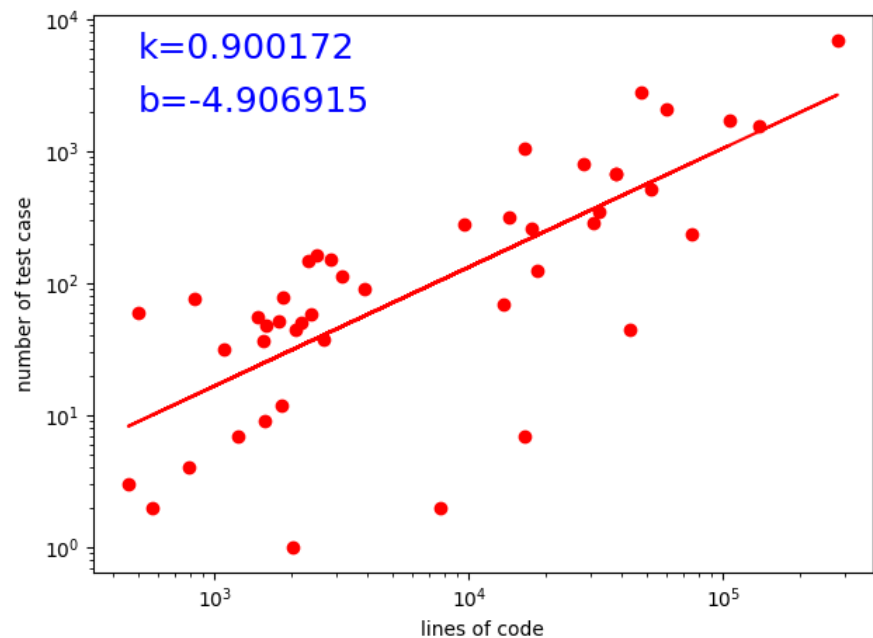
数据分析结果的进一步思考

- 实际情况中，潜在影响因素很多
 - 项目本身性质
 - 项目的社区
 - 软件开发的难易
- 3个RQ其实后面能对其作用的因素很多
- 没有绝对的控制变量对比实验
- 单单拎出两个变量进行对比，得出某种分布关系，然后强加自己的理解，不合理。

多变量分析

- 对样本的所有特征综合分析，可以得出更加客观的结论
- 六维空间？无法直接观察
- 能否在不进行直接观察的情况下得到样本的分布规律？

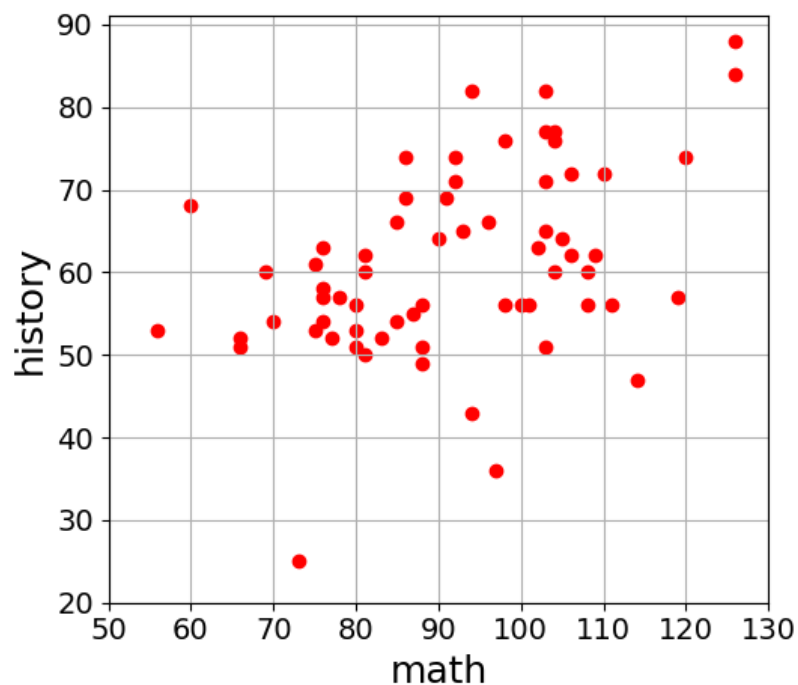
重要字段	解释
stargazers_count	项目获得的星星数(按此字段排序)
js_lines	JavaScript代码行数
ts_lines	TypeScript代码行数
issues_count	issue的数量
bugs_issues_count	bug类型issue的数量
unit_test_count	测试用例数量



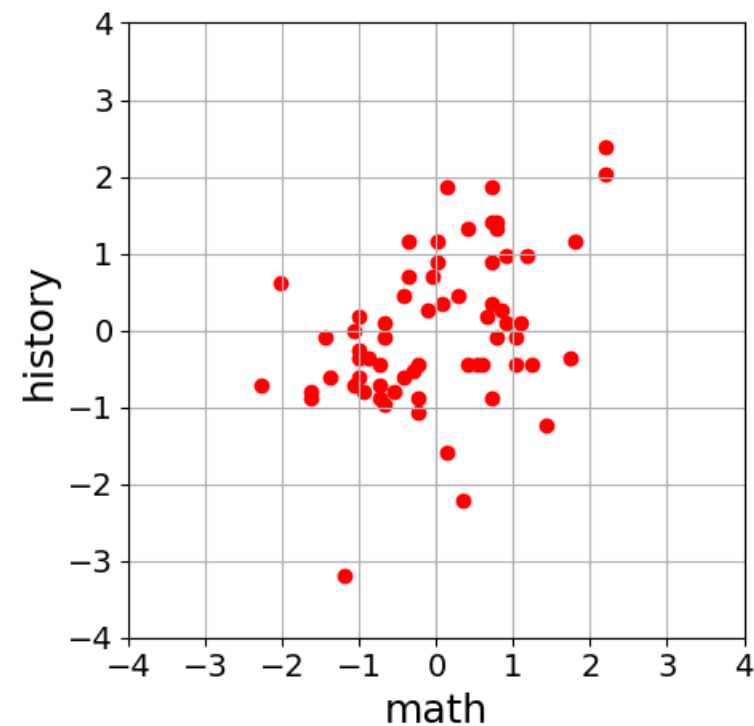
主成分分析——一个简单的例子

- 分析数学成绩和历史成绩的关系
 - 数学成绩和历史成绩成正比：学生本身的优秀程度
 - 数学成绩和历史成绩成反比：偏科
- 我们想知道的
 - 哪个因素占主导？
 - 两个因素的影响各占多大比重？
 - 数学成绩和历史成绩和这两个因素有什么关系？

主成分分析——一个简单的例子

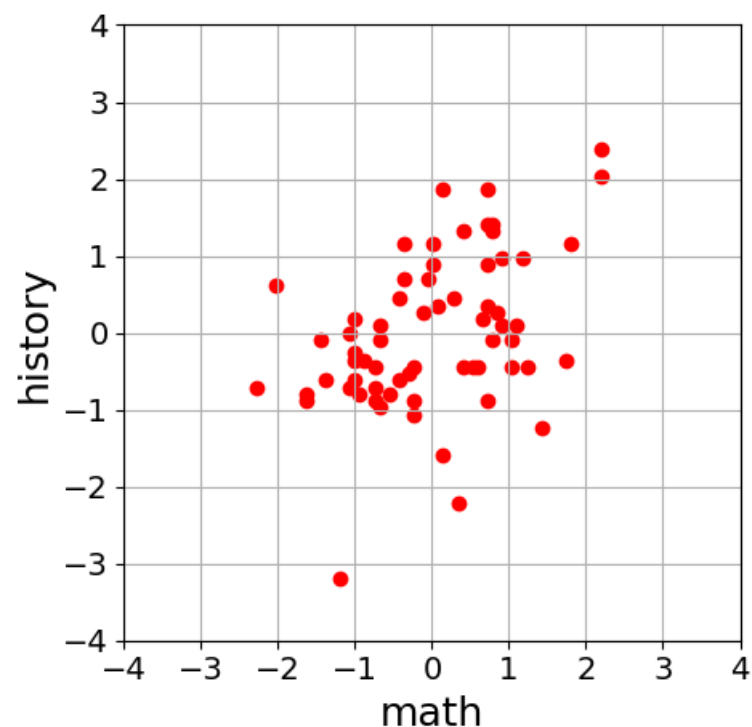


规范化 (均值为0, 方差为1)



画图是为了便于理解，实际上这个过程是纯计算的

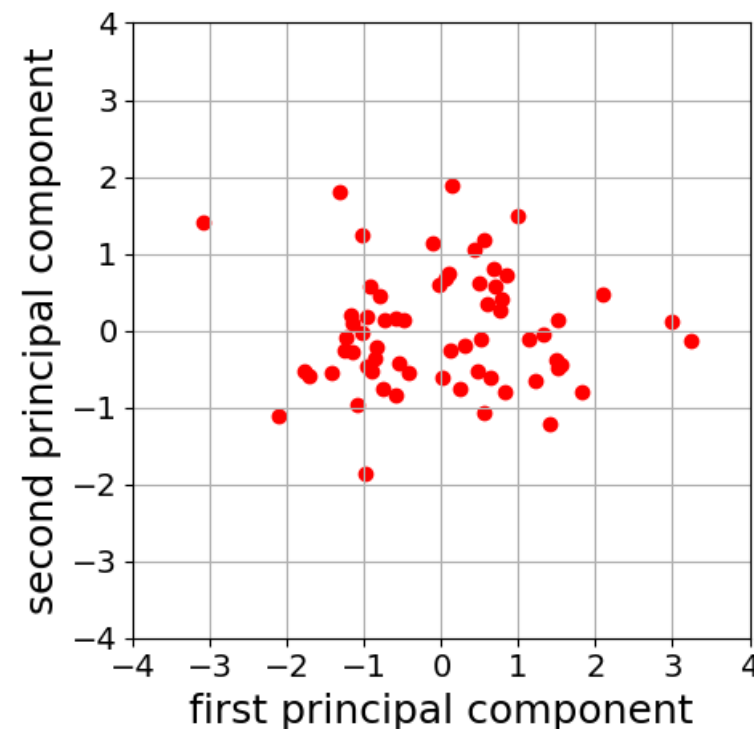
主成分分析——一个简单的例子



正交变换



$$\begin{bmatrix} 0.707 & 0.707 \\ 0.707 & -0.707 \end{bmatrix}$$



画图是为了便于理解，实际上这个过程是纯计算的

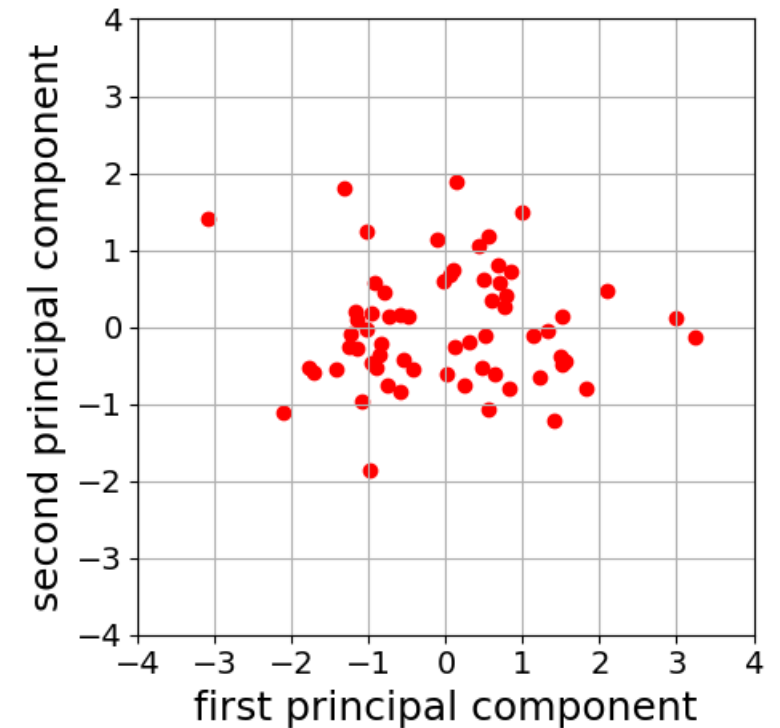
主成分分析——一个简单的例子

- 样本分布规律用主成分的方向（正交变换矩阵）来度量
- 分布规律对样本的决定程度用主成分的方差来度量 [1.459 0.572]
- 主成分与原特征之间的关系用因子负荷量来度量

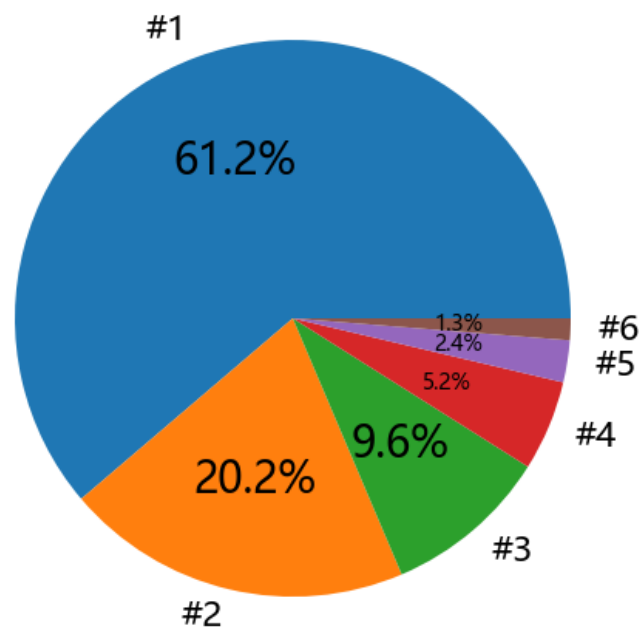
$$\begin{bmatrix} 0.707 & 0.707 \\ 0.707 & -0.707 \end{bmatrix}$$

$$\begin{bmatrix} 0.854 & 0.854 \\ 0.535 & -0.535 \end{bmatrix}$$

$$\begin{bmatrix} 0.729 & 0.729 \\ 0.286 & 0.286 \end{bmatrix}$$



多变量分析（六维）



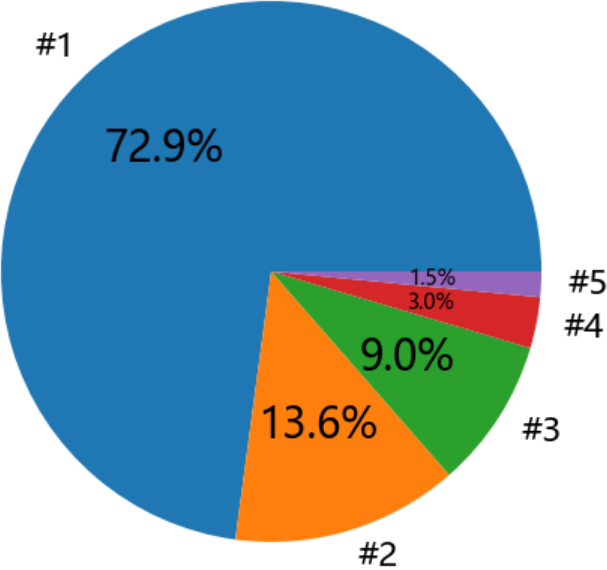
#1	3.71
#2	1.223
#3	0.584
#4	0.318
#5	0.148
#6	0.076

	star	js	ts	issue	bug issue	test case
#1	0.894	0.908	0.205	0.774	0.835	0.865
#2	-0.228	-0.299	0.927	0.209	0.367	-0.21
#3	0.158	-0.181	-0.211	0.577	-0.063	-0.38
#4	0.302	0.13	0.25	-0.059	-0.364	-0.104
#5	0.146	0.045	-0.043	-0.171	0.201	-0.23
#6	-0.143	0.211	0.008	0.042	-0.017	-0.097

	star	js	ts	issue	bug issue	test case
#1	0.799	0.824	0.042	0.598	0.698	0.748
#2	0.052	0.09	0.859	0.044	0.135	0.044
#3	0.025	0.033	0.044	0.333	0.004	0.145
#4	0.092	0.017	0.063	0.003	0.133	0.011
#5	0.021	0.002	0.002	0.029	0.041	0.053
#6	0.02	0.044	0	0.002	0	0.009

ts代码量和其他特征基本不相关

多变量分析（五维）



#1	3.68
#2	0.686
#3	0.453
#4	0.154
#5	0.077

	star	js	issue	bug issue	test case
#1	0.904	0.92	0.767	0.817	0.872
#2	-0.071	-0.329	0.588	0.296	-0.373
#3	0.369	0.092	0.2	-0.474	-0.211
#4	0.176	0.052	-0.186	0.173	-0.235
#5	-0.144	0.21	0.041	-0.009	-0.101

	star	js	issue	bug issue	test case
#1	0.817	0.846	0.588	0.668	0.761
#2	0.005	0.109	0.345	0.087	0.139
#3	0.136	0.008	0.04	0.224	0.044
#4	0.031	0.003	0.035	0.03	0.055
#5	0.021	0.044	0.002	0	0.01

多变量分析结论

- TS代码量和其他五维特征基本不相关
- 第一主成分说明样本的最主要分布规律是五维特征同时增加或减少
- 第二主成分说明js代码量和测试用例的数量多的项目issue和bug issue数量比较少
- 第三主成分说明bug issue数量的多的项目star数量会偏少

	star	js	issue	bug issue	test case
#1	0.904	0.92	0.767	0.817	0.872
#2	-0.071	-0.329	0.588	0.296	-0.373
#3	0.369	0.092	0.2	-0.474	-0.211
#4	0.176	0.052	-0.186	0.173	-0.235
#5	-0.144	0.21	0.041	-0.009	-0.101

研究结论

- RQ1: 测试用例的数量和整体代码量之间确实存在正相关关系, 但是使得测试用例数量增加的因素绝不仅仅是代码量, 还涉及到项目属性、项目的受欢迎度等多种因素。
- RQ2: 测试用例数量的增加可能有助于减少bug数量, 但是并不足以使得测试用例数量多的项目反而bug issue少。
- RQ3: 样本无法体现TS代码占比高的项目bug issue较少, 但这其中涉及因素太多, 很难说明TypeScript是否能减少bug。

有效性威胁

- 数据的准确性
 - Bug的issues数量作为bug数量
 - 获取测试用例数量过程太多意外情况：有记录的34个
- 数据量较小
- 统计量的代表性
- 选取的仓库的代表性

总结

- 使用一个半自动化的过程从github上拉取前端领域的仓库并获得其代码量、测试用例数量等数据
- 通过数据分析说明了当前开源社区中前端项目的测试情况
- 代码量、测试用例数量、star数量等存在正相关关系
- 测试用例的数量可能会减少bug的数量
- TS代码量和占比对bug数量的占比的影响不明显

后续工作

- 使用更多的数据
 - 使用更多的前端topic下的项目
- 数据收集更准确、更自动化
 - 手动运行测试?
 - 使用NLP等方法从输出结果中提取测试用例数量等数据

谢谢！