## 研究目的

研究目标：确定Github上开源项目中，用户的行为和项目发布后缺陷的产生之间的关系。

## 行文布局

本文分三个部分：行为，缺陷，关系。由于笔者研究主要涉及行为和关系的部分，所以这里只对这两者进行阐述总结。

1.5版本记录：这一版本，我们将此文的行文改为方法总结，主要介绍的是方法流程。具体的一些参数，可以不说，留到后面再仔细定义。

## 行为：

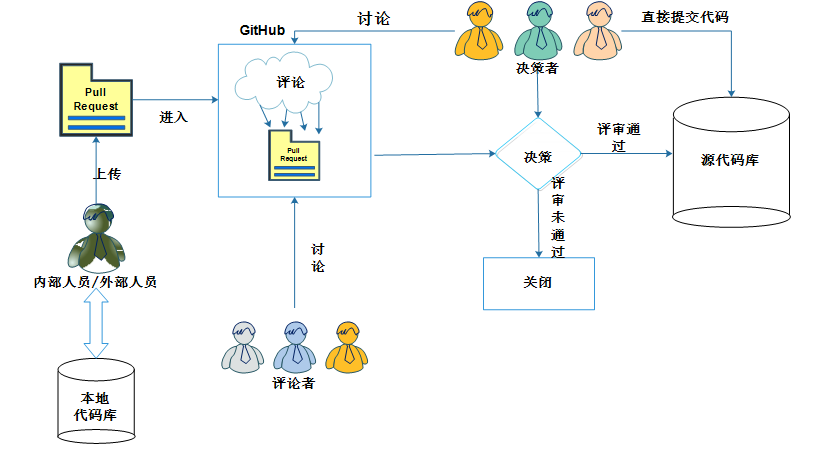
### 角色划分

根据Github开源项目的管理机制，我们观察并划分了三种贡献（行为）类别：改动，评论，决策。

下表展示的是行为及其定义。

|  |  |
| --- | --- |
| 行为 | 定义说明 |
| 改动 | 如果某位用户是某个pr的作者（author），那么这个 pr当中所有的代码改动（change），就归功于此位用户 |
| 评论 | 对pr作出评论（comment）的行为 |
| 决策 | 某位用户决定某个pr是被merge进主分支，还是不得merge进主分支，也就是被关闭或者拒绝（close 或 reject）的行为 |

下图展示的是各种角色和行为之间的关系。值得注意的是，一个用户在某一个项目当中可能扮演多种角色，比如，一个用户同时可能是决策者，也是评论者；某位用户可能既做出了代码改动，又参与了代码评论。



### 研究粒度及模块划分

为了使我们的分析具有更细致的粒度，我们对一个完整的开源项目进行了模块划分， 后续研究都是为了分析每一个角色在其发挥了贡献作用的模块中的贡献行为。

一个项目往往包含众多的文件和目录，如果研究的粒度是一整个项目，研究的难度势必增加，而且通过这种视角得到的结果不能不说是比较粗略的。所以，为了获得更加精致而准确的结果，将研究的范围从一个庞大而复杂的项目化小为一个个小部分是很有必要的。基于这种考虑，我们将项目划分成了众多的模块（module），并且在模块这一层级对用户进行行为分析以及后续的数据分析。

对于模块划分，我们采取的逻辑是：不包括其子文件夹的文件夹就是一个模块，直接从属于这一文件夹目录下的文件，同属这个模块。也就是说，一个文件夹，不包括其子文件夹，就是一个模块。例如，项目中某一个文件是/a/b/c/d/e.c，那么我们认为目录/a/b/c/d是一个模块，这一目录下的文件e.c，以及同一级目录下的文件，比如f.c, g.c同属这个模块；但是/a/b/c/d目录下还有一个子目录/h, 那么/h目录下的文件就不属于/a/b/c/d模块，而属于/a/b/c/d/h模块。

### 行为指标量化

三种行为都只是概念上的行为，为了研究行为对项目质量的影响，我们必须将行为化为可度量的指标；而为了适合数据分析的操作，我们则采用公式，对每个用户在模块中的三种行为指标进行了数值量化，量化后的改动、评论、决策三大指标分别被称为TCO，RSO，ISO。

我们参考并设计了三个指标来量化三种行为。这三大指标的具体定义如下：

#### 作者经过评审的代码改动量Traditional Code Ownership(仅考虑Author，系传统代码贡献量)

|  |  |
| --- | --- |
|  | 公式1.1 |

式子中， a(D, M)为用户D对模块M做出的改动的代码行数，CH(M)为模块M上经过改动的代码的行数之和。

直观上来说，TCO (D, M)反映了用户D在模块M中所占的代码改动量的比重。如果TCO大于0.05，则用户D就被称为模块M中的主要（major）改动者；反之则被称为次要（minor）改动者。

没有被merged的pr，不计入TCO的统计范围，因为这样的pr中所包含的代码改动不会进入到主（master）分支中，自然也谈不上引入任何缺陷。

#### 评论者贡献量Review Specific Ownership

|  |  |
| --- | --- |
|  | 公式1.2 |

式子中，r(D, M)是用户D做出评论了的、模块M中被merge了的pr数目；p(D, K)是用户D在符合条件的pr K中做出的评论条数的比例；CO(M)是模块M中被merge了的pr总数。

直观上来说，RSO (D, M) 反映了用户D在模块M中所做评论条数所占据的比重。如果RSO大于0.05，则用户D在模块M中就是一个主要评论者，反之则是次要评论者。

为了确保计算结果的统一性，RSO和TCO一样，没有被merged的pr，其所涉及的评论不计入RSO统计范围。另外，因为评论是针对pr开展的，而非针对特定模块，所以，我们认定，如果某次pr涉及到了k个模块，并且有n条评论，那么本次pr涉及到的所有k个模块都将会拥有1个pr和n条评论。

#### 决策者贡献量 Integrator Specific Ownership

|  |  |
| --- | --- |
|  | 公式1.3 |

式子中，w(D, M) 为D关闭或者merge了的对模块M的pr数量，PR(M)为模块M中被内部人员关闭或者merge了的pr总数。 ISO (D,M) 反映了模块M所涉及的pr中，被用户D决策了，也就是说关闭或merge了的比例是多少。如果ISO大于0.05，则用户D在模块M中就是一个主要决策者，反之则是次要决策者。

在公式2和3中，之所以还要考虑被关闭了的pr，是因为我们认为在评审过程中关闭pr，也是决策贡献。不过，核心用户（有过关闭或merge他人的pr的记录的用户）能决定其他用户的pr的命运，而非核心用户只能关闭自己的pr，所以称不上“决策”者，故此，我们将非核心用户关闭自己的pr这一行为所涉及的数据排除出ISO统计之外，因为我们认为这种行为最有可能的解释是：某位改动者，也就是pr的作者，通过评论或者是自己的重新审视，发现本次pr的确有不足，于是自行关闭了本次pr。

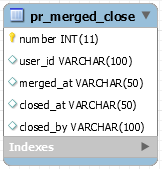
### 三大指标的计算

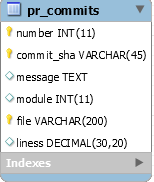
在三大指标计算的实现过程中，结合mysql和python，我们理清了三种行为指标的计算的流程。

我们以计算TCO，RSO，ISO为导向，设计了一套基于mysql和python的计算流程。我们设计了存储数据用的数据表（table），并且根据需要设计了存储计算中间值的视图（view）。在本文档中，为了显示的需要，我们将所有view都转化为table，并在view的名字前面增加了“table\_”。由于view和table在使用层面上没有区别，故在下文当中，文字表述上我们不对view和table做区分，统一以“表”或者“table”来表述。

#### 计算TCO

Pr\_commits和pr\_merged\_close表的结构如下所示：

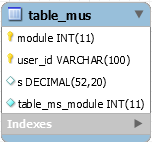




Pr\_commits存储的是pr的基本信息，包括：本pr的编号、作者的id（这个作者就是改动者）、是否被merge、什么时间关闭、被谁关闭。

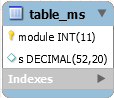
pr\_merged\_close存储的信息是pr当中commit的具体信息，包括：所属pr编号、本commit 的sha值、本commit的message、改动文件的名字、所属模块编号、代码改动行数。

两个表的信息，我们只需要利用其中特定版本发布日期之前的数据记录。我们将某一个用户在某个模块当中的记录全部找出来，再将记录当中的代码行数加总，得到她在这一个模块当中总共改动了多少行代码。这一步操作得到的结果存进表m\_u\_s中：



表的前三列表示：模块编号、用户id、该module当中该用户总共改动了的代码行数。

接下来是统计某一模块当中所有用户总共做出了多少行的代码改动。统计代码行数的列命名为s。得到的统计结果存在视图m\_s当中：

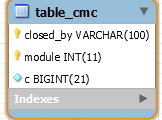


利用m\_s和m\_u\_s表中的数据进行计算，可以得到TCO的计算结果。因为m\_u\_s当中有用户D在模块M当中的改动行数的信息，m\_s表当中有模块M的总改动行数，所以结合公式1.1，我们可以直接获取我们所需的所有TCO值。至此，TCO的计算结束，结果会被存储在表result\_tco当中的tco列当中，如下图：



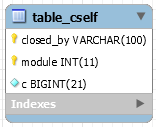
#### 计算ISO

首先利用Pr\_commits 与pr\_merged\_closed表中的信息计算出某一个用户关闭了的涉及到某一模块的pr数量，计算的结果记录在表c\_m\_c当中。表中的字值分别代表的是：用户id、模块、被这一用户关闭的涉及到这一模块的pr数量。值得注意的是，某一用户如果关闭了自己提交的pr，这种情况的“关闭”不计入计数范围；同时，在某一特定版本发布时间之后的数据也不计入计数范围。表c\_m\_c如图所示：

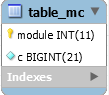


接着，我们统计核心用户关闭自己的pr的情况。核心用户指的是有能力关闭别人的pr的用户，并且我们认为这种用户关闭自己的pr是有特殊意义的。这种情况已经在之前c\_m\_c的计算中被排除，这一步中，我们将该情况统计起来用于后续的计算。

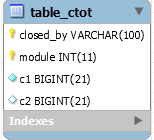
这一步的统计，需要pr\_merged\_close, pr\_commits, c\_m\_c三个表中的信息。根据我们对核心用户的定义，在c\_m\_c表当中出现的用户就是核心用户的全体。之后我们重复一次某一用户关闭了涉及某一模块的pr的总数的计算，只不过用户被限制在核心用户这一群体中。统计结果放在表c\_self中，表的结构和c\_m\_c完全一致。表如下：



之后我们计算每一个模块总共被多少个pr涉及到。这一步的统计需要表pr\_merged\_close以及pr\_commits中的信息。我们将以下两种情况排除到统计之外：自己关闭自己的pr这种情况，pr关闭的日期晚于某一特定版本时间的情况。统计结果放在表m\_c当中，表中的列值分别表示模块的编号和涉及这一模块的pr总数。表m\_c如下图所示：

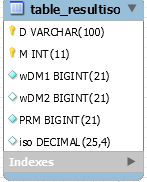


之后，我们把c\_m\_c和c\_self中的数据放在同一张表中。新表格被命名为c\_tot。如下图。C\_tot的列值分别表示：用户id，模块编号，表c\_m\_c当中保存的这个用户在这个模块当中关闭的pr数量，表c\_self当中保存的这个用户在这个模块当中关闭的pr数量。



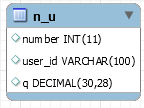
最后就是ISO值的计算了。这一步的计算涉及表格m\_c和c\_tot。我们首先要将用户关闭的涉及某一模块的pr（别人的和自己的pr）总数算出来，同时还要计算涉及到某一模块的被关闭了的pr总数（用户关闭别人的pr以及用户关闭别人的pr两种情况都要考虑）。两个值计算出来之后，直接套用公式1.3，就能够计算出所有的ISO值。ISO值被保存在表result\_iso当中，如下图，其中列D，M，iso分别表示用户id，模块编号，ISO值。

计算结果存储在表result\_iso中，如下图：

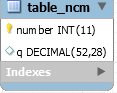


#### 计算RSO

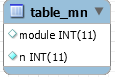
表n\_u中存储的列，分别是pr的编号number，用户的id user\_id，以及某位用户在这个pr中做出的评论数量q，如下图所示：



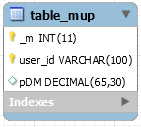
接着我们在n\_u中统计，每一个pr当中，总共有多少条评论。得到的统计数据储存在新表n\_cm中，如下图，其中列值分别表示：pr编号，这个pr当中总共有多少条评论。



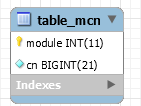
接着我们观察某一个涉及到某一个module的pr当中被merge的是哪些，注意这里没有求和操作。这个观察结果从pr\_merged\_close和pr\_commits里面获取。注意，我们不需要那些关闭时间晚于某一特定版本发布时间的pr。观察的结果被放在表m\_n当中，表如下图，两个列值分别表示：模块编号，pr编号。



随后，我们实现了公式1.2中的Σ运算。我们利用n\_cm, n\_u, m\_n, pr\_merged\_close中的信息实现运算。其中我们不考虑自己关闭自己的pr这种情况。运算的结果存储在m\_u\_p表中，如下图，列值分别表示：模块编号，用户id，公式1.2中分子的部分的运算结果。



接下来，就是计算某一模块涉及的，被merge了的pr的总数。计算的结果存储在表m\_cn当中，如下图所示，表的列值分别表示：模块编号，涉及了这一模块的被merge了的pr总量。



到这一步，可以进行最终rso的计算了。表m\_u\_p当中存储了公式1.2中的分子的值，表m\_cn存储了公式1.2当中分母的值，所以两个表的信息直接根据公式1.2进行计算即可得到所有RSO的值。RSO值存储在表result\_rso当中，如下图所示。其中列D, M, rso分别代表用户id，模块编号，这个用户在这个模块当中的RSO值。



#### 角色划分

TCO，RSO，ISO三大指标反映的都是用户D在模块M中某些行为的量化指标。根据之前所做的定义，这三种指标的不同取值，决定了每个用户在某一个模块中扮演的角色。角色如下表所示：

|  |  |  |  |
| --- | --- | --- | --- |
|  | **TCO** | **RSO** | **ISO** |
| >=0.05 | 主要改动者 | 主要评论者 | 主要决策者 |
| <0.05 | 次要改动者 | 次要评论者 | 次要决策者 |

如果某一个模块中某一个用户没有某项指标，那么这个用户在这个模块中的这项指标值为0。比如用户甲在模块1当中没有改动代码，那么在模块1中用户甲的TCO值就是0。

可以很容易得出，每一个模块中，根据三种指标的不同取值，都存在有八种类型的用户。根据这八种用户的TCO, RSO, ISO指标值是否大于0.05，也就是是主要还是次要贡献者，我们用n代表次要（minor）贡献者，用j代表主要（major）贡献者。例如，主要改动者，次要评论者，次要决策者用jnn表示。为了方便我们还用p1到p8对应了八种角色。详细的对应关系如下表所示：

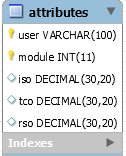
|  |  |  |
| --- | --- | --- |
| role | | 各种贡献量 |
| p1 | jjj | tco>=0.05, rso>=0.05, iso>=0.05 |
| p2 | jnj | tco>=0.05, rso<=0.05, iso>=0.05 |
| p3 | njj | tco<=0.05, rso>=0.05, iso>=0.05 |
| p4 | nnj | tco<=0.05, rso<=0.05, iso>=0.05 |
| p5 | jjn | tco>=0.05, rso>=0.05, iso<=0.05 |
| p6 | jnn | tco>=0.05, rso<=0.05, iso<=0.05 |
| p7 | njn | tco<=0.05, rso>=0.05, iso<=0.05 |
| p8 | nnn | tco<=0.05, rso<=0.05, iso<=0.05 |

表格 1

1. 按照用户和模块统合三大指标

我们首先将result\_tco, result\_rso, result\_iso三个表格的数据收集在同一个表中。

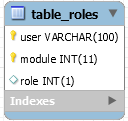
方法是：将这三个表中的数据按照相同的【用户, 模块】组合将tco，rso, iso三个指标数据凑在一起，形成一行。此表数据表明：某位用户在某一模块中的三个指标值分别是多少。如果某位用户在某个模块中没有某种行为，那么这个行为对应的指标值规定为0。得到的表命名为attributes, 如图：



1. 计算模块中每一种角色的比例

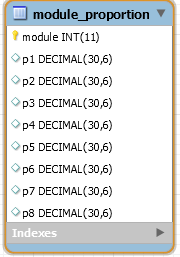
首先统计某个用户在某个模块中扮演了p1-p8当中的哪一种角色。

我们的计算方法是：直接利用attributes的三个指标数据，根据表1的贡献量划分标准，得出用户在一个模块中扮演的角色，角色在role列的取值是1到8，代表p1-p8八种角色。得到的新表命名为roles如图所示：



然后，我们计算某一个模块中八种角色占据的比例。

计算的方法是：利用roles表的数据，统计这个模块中总共有多少个贡献者（有改动、评论、决策中的一种或多种行为的用户）共有多少，再统计某个模块中p1-p8八种角色各有多少个，最后相除计算比例。得到的结果存在module\_proportion里面，module列储存了各个模块的编号，p1-p8列储存了这八种角色在某个模块中占据的比例，如图：



### 数据爬取计划的设计

根据我们这一套计算流程，我们搞明白计算所需的实际数据都是什么；结合github本身提供的api，我们最终整理了一套能够爬取我们全部所需数据的流程。

#### 表格内容

根据之前所表述的计算流程可知，所有的视图和表均源于最原始的三个表、即pr\_commits, pr\_merged\_close, n\_u的连接和选择。所以在这里，我们需要做出获得这三个表格的数据的计划。

前两个表， pr\_merged\_close涉及的是pr的基本信息，见于表格 2：

|  |  |
| --- | --- |
| 列名 | 表达的意义 |
| Number | 每一个pr的独特标识 |
| User\_id | Pr的作者的独特标识 |
| Merged\_at | 该pr是被merge还是被 close |
| Closed\_at | 该pr关闭于何时 |
| Closed\_by | 该pr为何人所关闭 |

表格 2

pr\_commits涉及pr中commit的信息，见于表格 3：

|  |  |
| --- | --- |
| 列名 | 表达的意义 |
| Number | Pr的独特标识 |
| Commit\_sha | Commit的sha值，为每个commit的独特标识 |
| Message | Commit所带的信息，在指标计算的过程中不发挥作用，而用于缺陷判断 |
| Module | Commit中代码被改动的文件所属的模块 |
| File | Commit中代码被改动的文件 |
| Lines | 文件中被改动代码的行数 |

表格 3

N\_u 涉及的是各个pr中每个用户作出的评论数量，见于表格 4：

|  |  |
| --- | --- |
| 列名 | 表达的意义 |
| Number | Pr的特殊标识 |
| User\_id | 发表评论的用户的id |
| Q | 用户在该pr当中发表评论的数量 |

表格 4

#### API获取以及数据爬取

为了获得上述表格2，3，4当中的数据，我们查询了github的开发者网页 <https://developer.github.com/v3/> 查询我们所需的api。我们最终选定的api如表格 5所示：

|  |  |
| --- | --- |
| 1 | <https://api.github.com/repos/:owner/:repo/pulls?state=closed> |
| 2 | [https://api.github.com/repos/:owner/:repo/issues/:number](https://api.github.com/repos/dotnet/corefx/issues/6769) |
| 3 | [https://api.github.com/repos/:owner/:repo/pulls/:number/commits](https://api.github.com/repos/dotnet/corefx/pulls/6769/commits) |
| 4 | <https://api.github.com/repos/:owner/:repo/commits/:sha> |
| 5 | [https://api.github.com/repos/:owner/:repo/issues/:number/comments](https://api.github.com/repos/dotnet/corefx/issues/6769/comments) |
| 6 | [https://api.github.com/repos/:owner/:repo/pulls/:number/comments](https://api.github.com/repos/dotnet/corefx/pulls/6769/comments) |

表格 5

这些api足以满足我们所需的所有数据的获取。值得注意的是，表格 5的api里:owner/：repo为项目的拥有者和项目的名称，在实际爬取的时候只需改成目标项目的拥有者及其名称即可。

1. Pr整体信息的爬取

所用API：<https://api.github.com/repos/:owner/:repo/pulls?state=closed>（对应的是表格 5中行1的url）。

作用：获得所有的关闭了的pull\_request（PR），这个api是所有的信息的根本来源。

返回：一个python列表

操作：对列表中的每一个对象（也就是每一次pr）提取如表格 6的字段：

|  |  |  |
| --- | --- | --- |
| 提取字段 | 备注 | 脚本中存储该值的变量名 |
| “Merged\_at” | 如果该字段非null，则应在csv文件中写入某个日期时间字符串，形如"2016-07-24T14:52:31Z"；若是null，则写入csv文件的时候将对应的字段值改为“closed”。 | Fin\_MergedAt |
| “Closed\_at” | 这个字段对应的是本项目关闭的时间。本次研究爬取的pr的时间范围为该某个版本前后半年，如果该字段值所代表时间不在这一年内，则跳过这个pr的爬取。 | Fin\_ClosedAt |
| “Number” | 反映了pr的编号。 | Fin\_num |
| “user” “login” | 该pr的作者的id | Fin\_user\_id |
| “Comments\_url” | 表5中行4对应的url的出处。 | - |
| “Review\_comments\_url” | 表5中行5对应的url的出处。 | - |
| “Issue\_url” | 表5中行1对应的url的出处。 | - |
| “Commits\_url” | 表5中行2对应的url的出处。 | - |

表格 6

1. 某个pr的issue信息

所用API：[https://api.github.com/repos/:owner/:repo/issues/:number](https://api.github.com/repos/dotnet/corefx/issues/6769)

作用：针对单个pr的issue信息获取。

返回：一个列表。

操作：提取字段如表格 7。

|  |  |  |
| --- | --- | --- |
| 提取字段 | 备注 | 脚本中存储该值的变量名 |
| “Closed\_by”“login” | 关闭这个pr的用户的id | Fin\_closed\_by |

表格 7

1. 某次pr当中所有的commits的基本信息

所用API：[https://api.github.com/repos/:owner/:repo/pulls/:number/commits](https://api.github.com/repos/dotnet/corefx/pulls/6769/commits)

作用：得到本次pr当中所有的commits的一部分信息。

返回：一个列表。

操作：提取字段如表格 8。

|  |  |  |
| --- | --- | --- |
| 提取字段 | 备注 | 脚本中存储该值的变量名 |
| “sha” | 该commit对应的SHA值 | Fin\_commit\_sha |
| “url” | 该commit对应的详细信息的api的url | - |

表格 8

这一步爬取至此尚未结束。

所用API：<https://api.github.com/repos/:owner/:repo/commits/:sha> ，这个API来源于表格 8中的”url”字段对应的url，也就是表格 5行4的url。

作用：得到本次pr当中某个commit的更详细信息。

返回：一个列表。

操作：提取字段如表格 9。

|  |  |  |
| --- | --- | --- |
| 提取字段 | 备注 | 脚本中存储该值的变量名 |
| “commit”“message” | 这个commit的message，里面可能包含我们进行缺陷检验的关键字，例如“defect”，“bug”，“fix”等 | Fin\_message |
| “file”字段对应的列表中的元素的“filename”字段 | 列表中的每一个元素，都是一个文件的改动信息。我们提取的字段是文件的名字 | Fin\_file |
| ”file” 字段对应的列表中的元素的“change”字段 | 对这个文件的改动数量，是删减的代码数加上新增的代码数 | Fin\_lines |

表格 9

1. 某次pr的评论信息的爬取

所用API：[https://api.github.com/repos/:owner/:repo/issues/:number/comments](https://api.github.com/repos/dotnet/corefx/issues/6769/comments) 与 <https://api.github.com/repos/:owner/:repo/pulls/:number/comments>

作用：获取评论数据。前一个API获取的是issue当中的评论，后者是获取pr的评论。在我们的研究当中，两种评论不做区分，并同时参与统计。

返回：列表。

操作：提取字段如表格 10。因为两个API获得的数据形式上十分类似，爬取方法上也几乎相同，这里不作区分。

|  |  |  |
| --- | --- | --- |
| 提取字段 | 备注 | 脚本中存储该值的变量名 |
| “user””login” | 评论的用户的id，我们不记录用户的评论信息，而是统计某位用户总共做出的统计的数量 | User\_quan{用户名，评论数量} |

表格 10

#### 数据存储

数据的存储分为两个部分，一个是爬虫爬取的字段存储在.csv文件中，另一个是将.csv文件中的数据存储进MySQl数据库中。

1. 存储进.csv文件

|  |  |  |
| --- | --- | --- |
| .csv文件名 | 对应表格 5中的api | 包含的脚本中的变量 |
| 0\_GeneralInformationOfPR | 1，2行 | fin\_num，fin\_user\_id ，fin\_MergedAt，fin\_ClosedAt ，fin\_closed\_by |
| 3\_DetailedInformationOfCommit | 3，4行 | fin\_num，fin\_commit\_sha，fin\_message ，fin\_module\*，fin\_file， fin\_lines |
| 4\_Comments | 5行 | fin\_num， user\_quan\* |
| 5\_ReviewComments | 6行 | fin\_num， user\_quan |

表格 11

\*fin\_module这个变量是以爬取的到被修改的文件名为根据，按照研究粒度及模块划分中所言的逻辑进行判断得到的模块的代号。如果出现了一个新的模块，就给予这个模块一个新的编号，若非新出现的模块，就给出一个该文件所属的、已经得到的模块编号。我们将模块的编号和模块的路径存在了module.csv中，这个文件会在后续的分析过程中被使用。

\*user\_quan是一个字典，key是用户的id，value是用户的评论数量。

1. 将.csv文件存进MySQL中

|  |  |  |
| --- | --- | --- |
| .csv文件名 | MySQL数据表名 | 相应列名\* |
| 0\_GeneralInformationOfPR | Pr\_merged\_close | Number, user\_id, merged\_at, closed\_at, closed\_by |
| 3\_DetailedInformationOfCommit | Pr\_commits | Number，commit\_sha，message, module, file, liness |
| 4\_Comments | Pr\_comments | Number, user\_id, quantity |
| 5\_ReviewComments | Pr\_review\_comments | Number, user\_id, quantity |

表格 12

\*列名的顺序与表格 11脚本中变量的顺序相同。

在指标计算的过程中使用的n\_u表由pr\_comments和pr\_review\_comments合并而成，具体的实现方法是：两个表中，相同number和user\_id的数据记录的quantity进行相加，两个表各自独有的number和user\_id就直接添加进n\_u表当中。

## 关系：

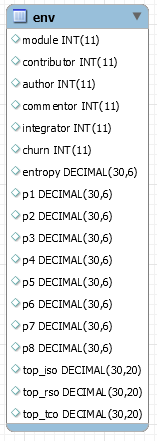
计算完某一个项目的所有模块中所有用户的各个指标数据之后，我们进入到行为和代码质量的关系的分析的阶段。

### 环境变量和反应变量

除了三大指标，还有一些变量，它们涉及到开源项目本身的特性、亦或是三大指标本身，被我们称之为环境变量（EV），可能与开发者行为一起，对项目质量起到影响。

根据我们在研究粒度及模块划分那一节中所言，我们研究项目的质量，其实是研究项目中各个模块的质量。所以，我们在每一个模块当中求出环境变量的值，为后续拟合模型以及模型评估提供数据。

我们选取了一些EV纳入到我们的研究当中，并且针对每一个模块，利用前文提到的mysql数据表实现了这些EV的计算，并将所有EV的值放在了表env中，如图：

表env表示的是，各个模块对应的各个EV的值分别是多少。

在模型拟合的过程中，我们还要设置反应变量（RV）。RV在模型拟合的过程中发挥监督学习标签的作用。

RV和我们选取的所有EV如下表。其中所有EV的计算方法也已列：

|  |  |  |
| --- | --- | --- |
| 环境变量 | 计算方法或公式 | 备注 |
| Contributor | 利用表attribute，统计某一个module当中有多少个贡献者，即：有多少行有着同一个module，就说明一个module有多少个贡献者 | 只要有做出改动、评论、决策三种行为中至少一种的人都纳入统计，做出多种开发行为的开发者只计数一次 |
| Author | 在contributor的计算方法基础上增加限制条件：记录TCO不为零的开发者的总数 | 做出过贡献的用户总数 |
| Commentor | 在contributor的计算方法基础上增加限制条件：记录RSO不为零的开发者的总数 | 做出过评论的用户总数 |
| Integrator | 在contributor的计算方法基础上增加限制条件：记录ISO不为零的开发者总数 | 做出过决策行为的用户总数 |
| Churn | 利用pr\_commits，pr\_merged\_close进行计算。将两个表的number列进行等值连接，去掉merged\_at 为“closed”和时间晚于某个版本发布时间的行，然后按照module分组，将同分组的liness列值进行加总 | 某一个模块中修改代码的行数的总和。 |
| Entropy | 根据公式进行计算\* | 熵值。如果计算出来可能得到Null值，则保留null值 |
| P1到p8 | 直接移植module\_proportion的列和值 | P1-p8在某一个模块中所占的比例 |
| Top\_iso | 利用表attribute进行计算，按照module进行分组，利用max()函数挑出最大的iso值，作为top\_iso的值 | 某个模块中出现过的最大的ISO值 |
| Top\_rso | 利用表attribute进行计算，按照module进行分组，利用max()函数挑出最大的rso值，作为top\_rso的值 | 某个模块中出现过的最大的RSO值 |
| Top\_tco | 利用表attribute进行计算，按照module进行分组，利用max()函数挑出最大的tco值，作为top\_tco的值 | 某个模块中出现过的最大的TCO值 |
| Size | 在module.csv当中，获得模块的编号和其对应的目录。之后利用pr\_commits和pr\_merged\_close表获取本次研究设定的时间期限内，一个模块中被合并进主分支的改动中改动了的文件有哪些。利用python的文本行数统计功能获得每一个文件的行数后加总可得到size值。 | 模块包含的被合并进主分支的改动中，被改动了的文件的代码行数总和 |
| 反应变量（responsive variable，RV） | 计算方法或公式 | 备注 |
| Status | 如要判断模块m是否有缺陷，我们寻找所有涉及到了模块m的commit当中的message（保存在表pr\_commits中）当中，是否包含关于缺陷的关键词\*。如果有包含关键词，说明这个模块m有缺陷，该值为1；否则说明这个模块没有缺陷，该值为0。 | 作为模型拟合的数据标签。 |

表格 13

\*熵，即entropy，其计算公式为：，公式中的n是模块M中文件的个数，是出现在文件k中的，对模块M的改动代码的比例。在计算的过程中，我们采取了一些措施：1. 不统计merged\_at为“closed”、时间晚于时间限制的行。 2. 计算的时候，如果某个模块的代码改动总行数，也就是构成分母的值若为0，这样的行不计入统计。 3. 构成的分子的值若为0，则的值定为0。 4. 计算的时候，如果n为1，则H(M)的值定为null。熵反映了在一个模块中代码改动的分布情况。熵越大说明在一个模块中代码改动分布于越多的文件中，也就是说在这个模块中代码改动情况越复杂。

\*关键词包括：'error', 'bug', 'fix', 'issue', 'mistake', 'incorrect', 'fault', 'defect', 'flaw','patch'。

表格13中的size和status在表env中不存在，因为这两个EV是另外单独生成的。表env当中的所有数据，附上size和status两个EV，会被存储在文件“env\_reponame\_result.csv”中，其中reponame是项目的名称。

### 拟合模型

计算了EV的值之后，我们利用得到的数据进行模型拟合。由于我们探究的是行为是否会导致某一模块产生缺陷，所以利用逻辑回归模型（lrm）来拟合这个二元关系是再合适不过。在拟合过程中，为了提升模型的准确度，降低拟合难度，我们进行了冗余EV的删除；以下是拟合模型的步骤：

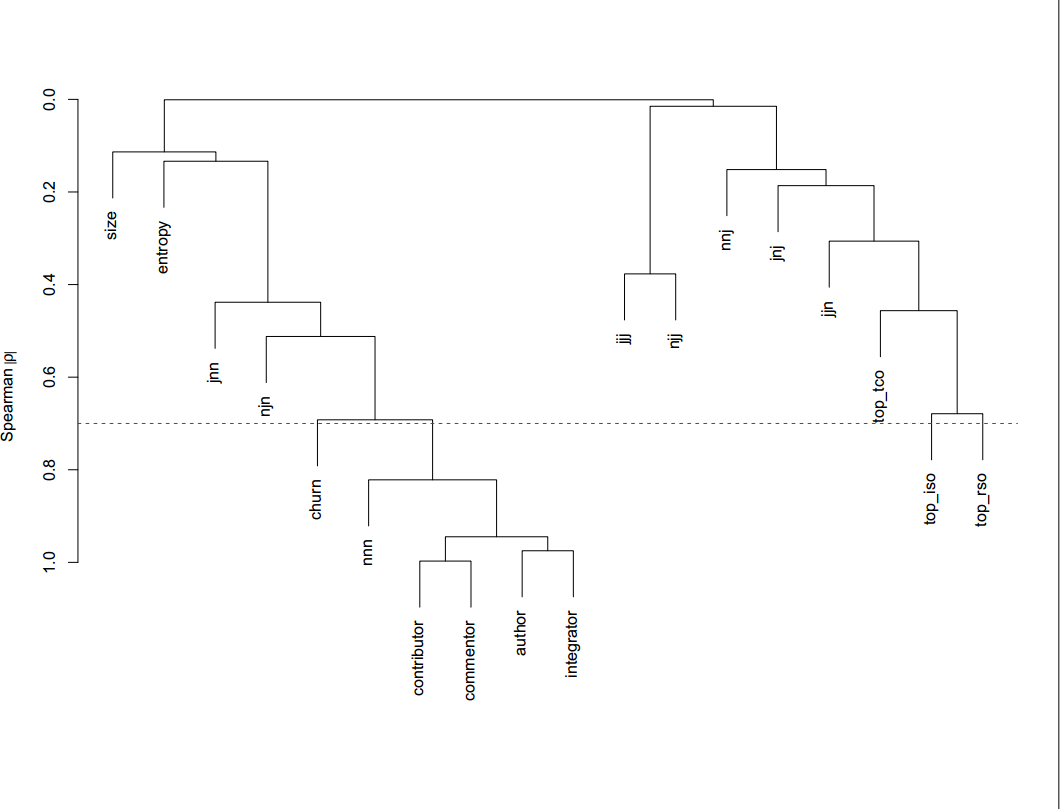
#### 数据准备

存储在文件“env\_reponame\_result.csv”中的数据，为了与爬虫获取到的原始数据进行区分，我们称之为拟合数据。拟合数据会被导入到R语言中，存储为dataframe格式的数据结构。之后我们将dataframe中p1-p8的列名全部改为表格1中定义的由n, j组合成的名称，这样的变量名在后续的数据分析当中会比较明显和直观。

#### 相关性分析

可能有一些EV变量之间可能高度相关(correlation)，如果不去除一些具有相关关系的EV，可能会影响模型拟合的结果。我们利用Spearman相关系数，一种等级相关系数，来判断哪些EV之间有着高度的相关关系。

首先，我们利用varclus函数对环境变量进行相关性分析。分析的结果，也就是Spearman值，会被反映在一张聚类等级图上，图像的名称为correlation.pdf，如图所示：



这张图中，相关度在0.7以上的变量就可以被认定为是高度相关的。当中相关度在0.7以上的一个聚类中，一个EV会被选择留下，作为这一聚类EV的代表参与后续的各个步骤，聚类中的其余EV则被排除在其后的步骤之外。被排除的EV会被记录在documents.txt文件中。在实际操作的过程中，我们更倾向于保留计算相对简单的EV，例如contributor这样纯粹的统计变量，但是这不绝对，这取决于我们的研究目标。

#### 冗余变量分析

经过上一步的相关分析之后，我们可以认为，剩余的EV之间不再具有高度的相关关系。但是这并不能代表这些变量当中就没有冗余的变量。何谓冗余变量？那些和其他EV相比，没有独特的标志(unique signal)的变量，就是冗余变量。冗余变量会扭曲EV和RV之间的建模的关系（modelled relationship）。所以，冗余变量有必要被去除。

Rms包中的Redun函数可以分析变量是否冗余。被判断出来是冗余的变量，不进入后续的分析步骤中。冗余变量会被记录在documents.txt中。

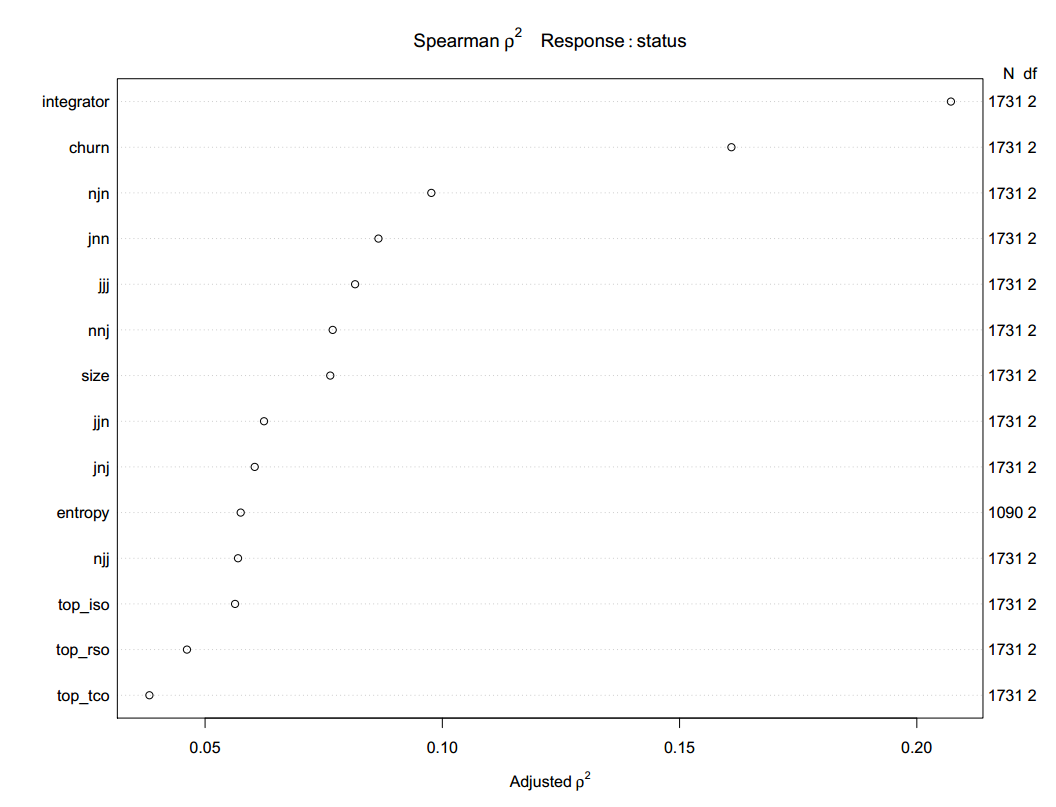
#### 计算自由度分配上限

一个过拟合的模型会根据（based on）建立这个模型所用的数据集的特征，来夸大或者低估EV和RV之间的关系，也就是说，这样的模型反映出来的关系是不准确的。过拟合往往会因为使用了超出模型所能承受的量的自由度（Degree of freedom）而发生。为了降低过拟合风险，我们计算了模型的自由度使用上限并据此安排每个变量在拟合过程中的自由度分配量。计算的方法是：原始数据中，status列为true的行数和false的行数，哪一个更小，就选取那一个值然后除以10，得到的就是自由度分配上限。该值会被记录在documents.txt中。

#### 非线性性检验

所谓自由度，是回归模型的一种叫knot的特性。何为knot？knot就是EV和RV之间关系（relationship）的方向变化的次数，knot的值越大，说明EV和RV之间的关系就越不可能是线性的。如此一来，和RV的关系越不可能是线性的EV，就要安排越多的自由度，才能更准确地拟合模型。我们如何判断RV和EV之间的关系的形状呢？我们计算每一个EV的Spearman multiple 值。此值越大，说明这个EV与RV的关系越不可能是线性的，越应该分配更多的自由度。

计算每一个变量的值，生成的图被命名为rou2.pdf，如图所示：



#### 模型拟合

模型拟合利用R语言的RMS包当中的lrm（EV1, EV2, EV3 …….）（逻辑回归模型）方法。结合上述的步骤4、5，我们在模型拟合的过程中做出如下操作：

Knot的值，也就是自由度的值，总共有三个值：5,3和1。分配的knot值总量不能超过上限值。分配knot用rcs函数来实现，形如：rcs( EV, 3)。Knot值分配的结果会被记录在documents.txt文件中。我们在实现上述模型拟合的方法的时候，优先为值大的变量安排足够的自由度。为什么要这么做？是为了更有效率的分配本就有限的自由度上限。具体操作的时候我们采取的原则是，先从值最大的环境变量开始分配，knot取值先从5开始，如果拟合出错，就改小分配的knot值，没有出错的话就给下一个环境变量分配。

拟合得到的模型，在本文的后面部分用fit\_model表示。

## 模型评估

### 模型整体分析

对于拟合得到的模型，我们分析模型的AUC值来判断模型的解释能力，我们只采用那些解释能力强的模型进行接下来的分析。我们还通过计算AUC optimism值来判断模型解释能力大小这一结论是否具有可靠性。

AUC是衡量二分类模型优劣的一种重要评价指标，表示正例排在负例前面的概率。AUC值越接近1，越能说明这是一个能把正例从负例中区别出来的模型；该值越接近0.5，越能说明该模型对于正负例的判断与随机猜测无异。所以我们应该采用AUC值更靠近1的模型。

如果模型发生了过拟合，则AUC值就会高估模型的解释能力。我们通过计算AUC 的optimism值来反映模型的可靠性（reliability of our models）。这种计算方法使用的是一种bootstrap方法。这种方法的原理是：从原始数据中有放回地取样，然后对比在取出的样本上拟合出来的模型和原始数据中拟合出来的模型之间的差别就是optimism，这个过程重复1000次取平均值作为最终的optimism。Optimism值越小，说明原始数据上拟合的模型的表现就越加可靠。

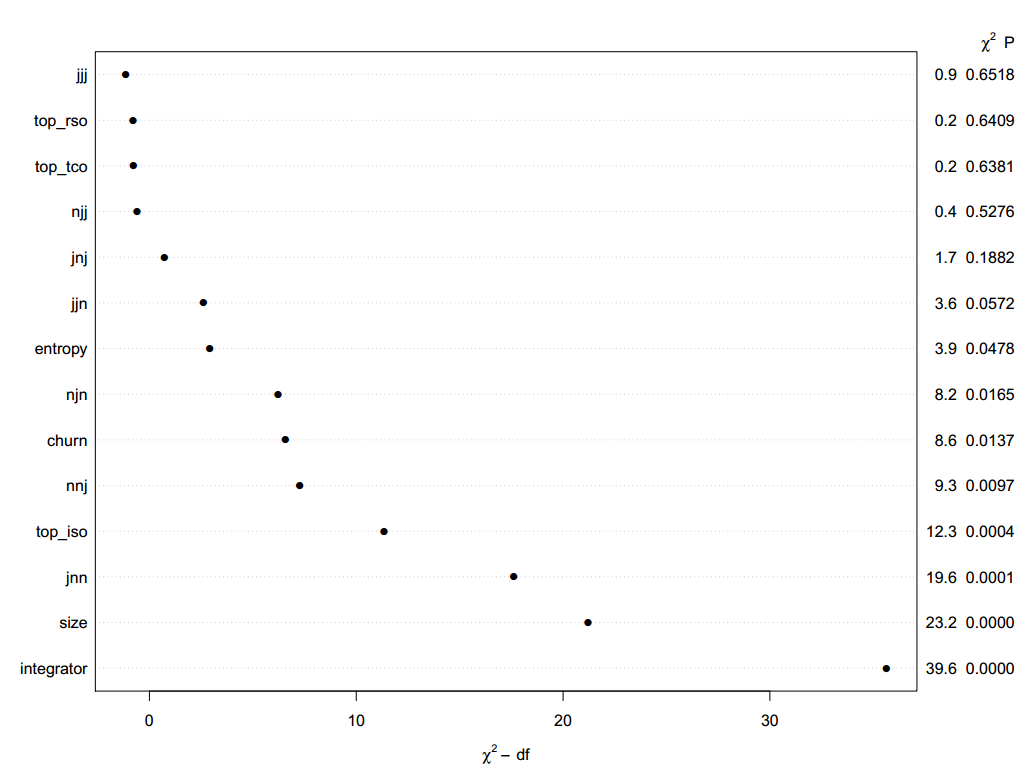
R语言的代码实现的步骤如下：利用validate函数，参数是fit\_model, 参数B取值1000，意为重复1000次。得到返回对象，存在变量val中。Val[1,1]的值的一半加上0.5，是AUC的值，存在变量AUC中。Val[1, 5]的值的一半加上0.5，是减掉optimism之后的AUC值，存在变量AUC\_optimism\_reduced中。所以，AUC\_optimism\_reduced – AUC得到的差就是乐观度的值，存在变量AUC\_optimism当中。

### 解释变量分析

选取了解释能力强的模型后，我们从两个方面分析，每一个EV在拟合的模型中发挥了什么样的作用。

首先我们利用每一个EV的值来判断每一个EV对于模型的影响力的大小。值越大的EV，在模型拟合中的影响力越大,或者说对于模型的影响程度越大。这也就是说， 值越大，就越加强有力地说明某一模块有或是没有发布后缺陷，反之则越没有效力。

值及其p值将由anova(fit\_model) 函数求得。我们着重研究的EV是值的对应p值足够小的EV，较小的p值说明该值具有统计学意义，值得研究。在我们的研究当中，我们拟定p值小于0.05的变量具有统计学意义，这样的变量在后文当中也被称作有效变量。值和p值存储在图片chi2\_and\_pValue.pdf当中，如图：



同时，我们也对卡方值和p值在documents.txt当中进行存档。在文档中存档的数据形式如下所示：

Factor Chi-Square d.f. P

size 104.37 4 <.0001

Nonlinear 87.90 3 <.0001

integrator 20.16 3 0.0002

Nonlinear 0.42 2 0.8104

njj 11.12 3 0.0111

Nonlinear 0.36 2 0.8340

nnn 2.57 4 0.6315

Nonlinear 1.85 3 0.6033

jjj 19.53 2 0.0001

Nonlinear 0.02 1 0.8750

njn 1.13 2 0.5693

Nonlinear 0.94 1 0.3333

churn 0.87 2 0.6468

Nonlinear 0.87 1 0.3506

jjn 2.13 1 0.1442

entropy 0.90 1 0.3424

top\_rso 1.34 1 0.2465

nnj 0.31 1 0.5765

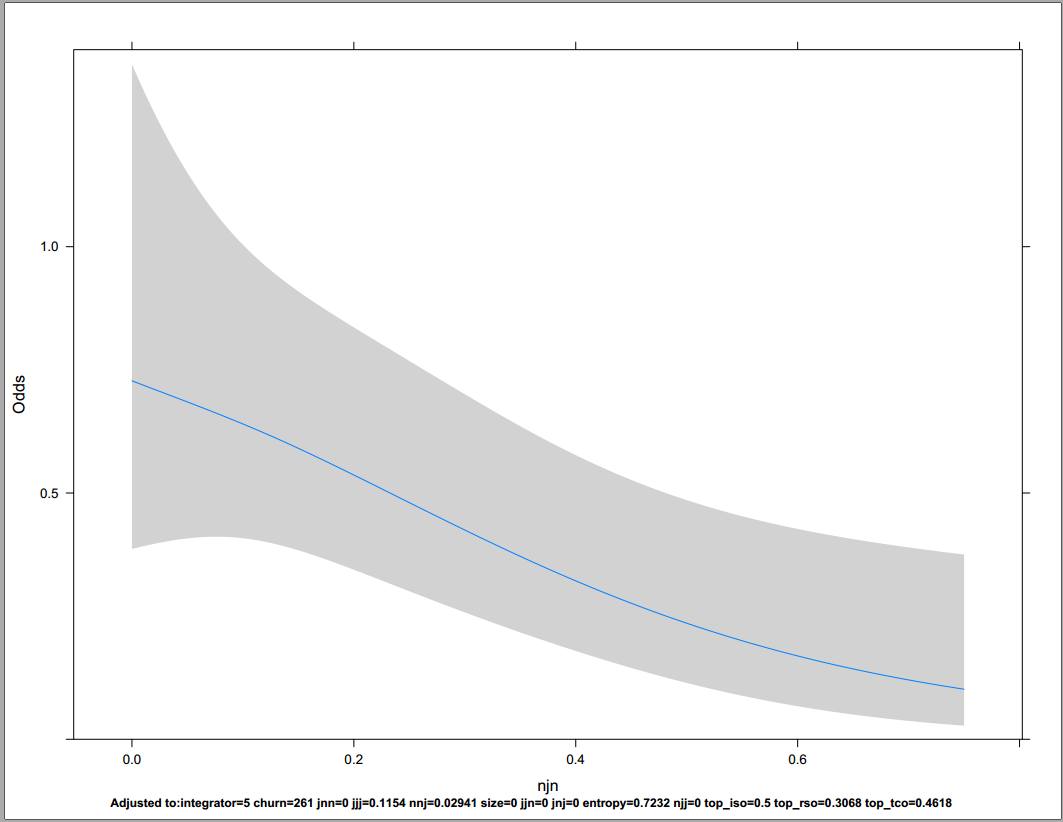
top\_tco 2.81 1 0.0935

jnn 3.50 1 0.0614

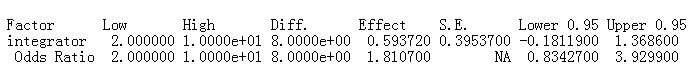
TOTAL NONLINEAR 92.70 13 <.0001

TOTAL 212.73 26 <.0001

注意这里，有一些EV的下一行会是一个Factor列值为Nonlinear的行，这些行表示的是上一行所表示的EV被分配了非线性自由度；如果EV的下一行不是Nonlinear行，说明这个EV在anova(fit\_model) 函数的运行过程中没有分配非线性自由度。

值的大小只是说明了EV对模型影响程度的大小，我们还需要利用predict图和odds ratio来判断EV对于是否产生发布后缺陷的取值的影响方向，也就是判断某个EV究竟是促进缺陷的产生，还是防止。Predict图如下：图中蓝色的线若向上说明，待测EV（此图中是njn）的值越大，某一模块越有可能出现缺陷；向下则反之。待测EV的值会从一个值变到另一个，而其他的EV则保持在某一个值上不变，这个值往往是中位数（median）。灰色的区间是95%的置信区间。

Odds ratio是一个值。我们通过得到odds ratio的值来反映某一个EV对于RV的边缘效应（partial effect）。Odds ratio暗示（indicate）了，当某一个EV的值的增长时，一个模块有发布后缺陷的可能性的变化。Odds ratio值越大，EV对于这个可能性的边缘效应越大。我们通过summary(fit\_model) 函数得到odds ratio。不过，summary函数得到的值还要经过处理才能满足我们的分析需求。如图是运用了函数之后得到的数据结果：

得到的数据的“Odds Ratio”一行的effect列的值减去1，得到的数字设为t。将t转化为百分比，就表示当某一EV的值发生变化而其他EV保持在某一个值不变时，odds相应地发生了的变化的百分比。我们基于t值来估计边缘效应，其实也就是说，将t值作为边缘效应的衡量指标。如果边缘效应值为正，说明这个EV越大，越有可能导致某一个模块有发布后缺陷的产生，反之则越不可能；边缘效应的规模的大或者小，反映了随着这个EV的值的改变，模型中odds值的改变量的多少。

至此我们就完成了某个项目的全部分析过程，我们已经可以得到针对某一个项目的结论了。

## 研究过程

文章之前的部分详细描述了数据如何获取、如何计算、如何保存，完整地介绍了本次研究所用的方法，缕清了所有流程。在本次研究过程中，上述步骤都被编写为完整的脚本文件，这些脚本包括：严格执行了数据爬取计划的爬虫脚本，环境变量计算的脚本，反应变量计算的脚本，模型拟合的脚本，模型评估的脚本。

上述脚本能够在理论层面完成所有的分析步骤，但是实际研究的时候，很多具体问题的应对措施或者是脚本运行的一些参数，是结合软件项目的特点，根据软件工程的经验以及小范围对某些项目尝试进行全部操作后得到的结论来确定的。这些问题就包括选取什么样的项目作为研究对象，上文一直提到的时间期限设定为多少等。在这一部分内容当中，我们会将本次研究中涉及到的所有这些问题进行描述和总结。

### 项目的选取

进入到模型评估阶段的数据是环境变量和反应变量，每一行数据都是某个模块的特征和标签取值，所以模块数量越多，则进入到模型评估阶段的数据行数就可能越多。行数越多，则越有助于模型的拟合结果得到具有一般性的结论。并且，热门项目的开发、评论、决策行为也会比较多，能够提供更多相应数据，更好地反映用户的真实行为特征，同样有助于拟合出更有一般性的结论。所以，github当中的一些热门的、文件夹数量较多的项目在本次研究中被作为目标项目。针对这些项目，所有的原始数据会被爬取以用作后续研究。下表中展示了我们最终选定的项目。



### 时间期限的设定

所谓时间期限，在本次研究中是如下定义的：选定的项目发布日期之前的时间期限长度内，我们利用这些数据计算环境变量；选定的项目发布日期之后的时间期限长度内，我们利用这些数据计算反应变量。

通过对项目kubernetes开展了数据爬取、指标和变量计算、模型拟合和评估的全流程之后，我们发现，以六个月为时间期限长度相比于以两个月为时间期限长度，自由度分配上限更大，并且有效的环境变量更多。同时，p1，p2，p3，p4这四个环境变量有效，便于我们得出研究结论。

所以，我们决定采取六个月为时间期限长度爬取数据。

### 主（master）分支的选取

在原本的研究计划当中，merge行为指的是贡献请求里的代码改动被合并进主分支当中。根据实际的项目数据爬取结果，我们发现并不是所有的改动都被合并进主分支当中，有的项目的代码改动主要被合并到其他分支当中。有的项目虽然在研究的期限内，依旧有大量的代码贡献请求产生，但是在更早以前，就没有合并入主分支的贡献请求了。所以，我们认为，存在这样一种情况：项目的最主要分支已经更换，而非主（master）分支。为了规避这种情况，我们决定在利用任何api进行数据爬取的时候，不在api请求中加入分支限定。

### 带有“Merge pull request#”信息的数据的去留

我们使用了自己的项目做了一些实验，发现，只要合并一个贡献请求，系统就会自动生成一条以“Merge pull request#“打头的commits信息。这样的条目包含重复信息，如果将这样的条目计入统计，那么某次贡献请求涉及的代码改动行数、某个模块涉及的代码改动行数就会变多。例如，如果3\_DetailedInformationOfCommit文件当中包含了编号为1的贡献请求的commits信息，其中对b模块中的a文件的改动行数是2；那么后来这个贡献请求被合并进主要分支，会生成另一条信息，其中对a文件的改动也是2。那么计算b模块的代码总改动行数时，就会多2。

考虑到上述情况的存在所以，我们不记录这样自动生成的commit条目。

## 统计结果分析

该部分主要内容是：本文2.2部分拟合模型和3模型评估部分的结果展示以及分析。

### 相关性分析结果

对所有的项目进行了环境变量相关性分析，得到的结果统计如下：



所有项目的拟合数据进行Spearman相关性分析之后，生成的如2.2里面相关图表所示的图形。在这张表格当中，列表示的是某一个项目的相关性情况，各个行表示的是不同的环境变量。一列当中，数字相同的环境变量，它们之间的相关性超过0.7，互为高相关性变量。比如，akka项目中，contribution、author、commentor、nnn的值为1，他们同属一个高相关性变量簇；ingetrator和top\_iso的值同为2，他们属于同一个高相关性变量簇。

根据Patanamon等人的文章所述，这些高相关性变量取其中的无论哪一个进入后续的模型拟合过程，其结果，包括后续的模型分析过程的结果，都不会有太大变化。所以在后续的过程当中，当我们分析某些项目的某个环境变量的特性时，其特性可以推广到所有与之有高相关性的变量上。

在这张表格当中，如下情况可以被观察到：在相当一部分的项目当中，commentor和contributor总是相关，这两个变量在较多的项目当中也和nnn相关；top\_iso和integrator总是相关；author和churn总是相关；top\_rso和njn在一些项目中相关。

我们可以得出如下的观察结论：贡献者数量和评论者数量成高度相关关系；决策者的数量和模块的最高ISO指标值成高度相关关系；模块中代码改动贡献者数量与代码改动总行数高度相关。

### 项目基本情况汇总

所有的项目都有的一些分析结果的汇总。这些分析结果包括：自由度分配上限，拟合模型时花费的自由度的量，AUC值，AUC乐观度值，总量，非线性总量。



--画下划线的表项表示这个项目的分析结果不存在。

--总量和非线性总量数值后面的符号的解释如下：所有的相关的计算结果都有对应的p值。如果p值大于等于0.05，值后标注“’”；p值在[0.01, 0.05)之间，标注“\*”；p值若在[0.001, 0.01)之间，标注“\*\*”；否则标注“\*\*\*”。

所有项目的模型拟合过程，自由度使用量都没有超过上限值。并且，所有的项目的AUC值都保持在0.728-0.904之间，属于较高水平；并且AUC乐观度在0.008-0.03之间，属于较低水平。根据Patanamon等人的文章所说，说明这些模型的解释能力强，可靠性高。由此我们有理由相信，利用这样的模型能够得到较为可信的分析结果。

### 解释变量分析结果

利用3.2部分相应图表，我们可以得知某一个项目所有参与了模型拟合的EV的解释权重的值以及相应的p值。我们将所有的和p值进行了汇总。经过观察，我们发现，有一些项目的EV中有相当一部分的比重只有0%，我们决定在后续的分析中去除这些项目。我们选取了八个项目进入到后续的分析步骤。这八个项目的分析结果如下：

线性自由度的分析结果汇总：



--每一个表项所表示的是：该变量的线性值占该项目总线性值的比重/该变量被分配的线性自由度。

非线性自由度的分析结果汇总：



--每一个表项所表示的是：该变量的非线性值占该项目总非线性值的比重/该变量被分配的非线性自由度。“-/-”表示该项目中，该EV没有被分配非线性自由度。非线性表中的“?’/-”中的“?”表示的是，这个变量有被分配非线性自由度，但是这个项目的所有EV拟合出来的模型并没有非线性总值（如上表所示）。

--以上所说的每个项目的总（非）线性自由度，可以在上面的表中查到。

每一个项目的总值已经在前面的表中显示出来了。之所以计算值所占比重，目的是用一个统一的视角去查看同一个变量在不同项目当中占据的解释权重。

1. 通过观察发现，一个项目中，只有较少的EV会占据10%以上的解释权重。例如在线性表的ansible项目当中，参与模型拟合并且进行解释权重计算的EV有13个，但是占据10%以上解释权重的只有4个；再如，nixpkgs项目当中，没有一个EV占据了10%以上的权重。

所以我们认为：占据10%以上解释权重的EV对模型有较大影响。

之后，我们结合相关性表，对分析表和的数据进行改动：高相关性变量簇里面参与到模型拟合并生成了模型评估结果的EV，其参与模型拟合、模型评估得到的结果复制到未参与到模型拟合、模型评估的EV的相应区域当中。例如：roslyn表格中，contributor，author，commentor，integrator，top\_iso这五个EV属于同一个高相关性变量簇，其中integrator参与到了模型拟合、模型评估的过程当中，得到了等相应分析值。我们将这个integrator的相应分析值复制给另外四个EV，认定这四个EV如果参与了模型拟合、模型评估，也会得到非常相近的值。经过这样的处理之后，我们得到了其他的观察结论。

1. jjj，jnj，njj，nnj，jjn，jnn，njn，nnn这八个指标，对项目产生了10%以上的解释权重的变量只有如下：jjj，njj，jjn，jnn，njn，nnn。其中，nnn在最多的项目当中有较大影响；jnn，jjj次之；njj、jjn、njn又次之，但是jjn、njn能够达到19%-24%的较高的影响力。

所以，我们可以认为，改动、评论、决策行为都是次要程度的用户对代码的质量在比较普遍的情况下是有较大影响的；三种行为都是主要程度的用户也有较大影响；具有主要改动、评论行为的用户的影响力较大，而具有主要决策行为的用户相较之下影响较小。

因为我们的本次实验关注的重点是用户行为和质量的关系，所以我们主要研究的EV就是jjj-nnn这八个指标。

### 预测图和odds ratio结果

预测图能够很直观地让我们知道某个EV对代码质量的影响的类型是正还是负；而odds ratio则能够以量化的形式地说明某个EV的改变能够对代码质量产生影响的类型以及影响的程度。

Odds ratio值的结果如下表所示：



这张表格显示的是在进行odds ratio计算的过程当中，一个EV在别的EV值不变的情况下，从某个值变到了另一个值得情况。其中每一个表项表示的就是这个值的变化情况。值得注意的是，所有的值都是从低到高变化的。



上表表示的是odds发生变化的百分比，如本文3.2部分所述。

接下来的段落，我们结合预测图和odds ratio的结果来分析每一个EV对项目质量的影响。

#### 用户组成情况

我们先对用户组成的EV，也就是jjj-nnn这八个变量的预测图和odds ratio进行分析。这八个EV的取值对项目质量的影响是本次研究的重点目标。

因为我们只对结果具有统计学意义的项目的EV作预测图，所以我们首先统计这八个EV的结果分别在哪些项目里具有统计学意义。而有一些项目中，待研究的EV可能并未进入到模型拟合阶段，我们尽量参考和这些EV同处一个高相关性变量簇、且其结果具有统计学意义的其他EV的预测图来估计待研究EV的预测图形态；其他高相关性簇中的EV的预测图也有可能提供更多具有参考价值的预测图，帮助我们获得更加有说服力的观察结论。

|  |  |  |
| --- | --- | --- |
| EV（环境变量） | 在哪些项目中该EV进入模型拟合步骤，并且其结果有统计学意义 | 参考其他项目的其他EV的预测图 |
| Jjj | ansible, corefx, elasticsearch | - |
| Jnj | prestashop | - |
| Njj | cocos2d\_x, coreclr, corefx, elasticsearch, kubernetes | - |
| Nnj | coreclr, kubernetes, roslyn | - |
| Jjn | ansible, coreclr | - |
| Jnn | ansible, cocos2d\_x, coreclr, elasticsearch, kubernetes, prestashop, roslyn | - |
| Njn | ansible, cocos2d\_x, coreclr, corefx, elasticsearch, roslyn | - |
| nnn | coreclr, prestashop | corefx的integrator, roslyn的integrator |

Jjj：在odds ratio里面，绝大多数项目当中jjj的增加会导致代码出错的概率降低。结合预测图，所以我们有理由相信，改动、评论、决策的主要实行者对项目的质量有积极作用。

Jnj：根据odds ratio的计算结果可见，大多数项目当中，jnj的数值上升，会导致代码质量提升。但是根据结果可以看出，除了prestashop项目之外，没有一个项目的值是有统计学意义的。而prestashop的预测图中，虽然曲线的置信区间在增长，但是总体的odds 值不算大，我们依然可以相信有足够的数据来支持这个预测结果。但是jnj具有统计意义的项目太少，我们认为这样的预测结果不能用在其他软件项目上，所以我们并没有十足的把握相信主要改动、次要评论、主要决策者且者对项目的质量有不良影响。

Njj：Cocos2d\_x项目当中，曲线上升趋势不明显，而且置信区间在扩大，说明预测结论不可靠。其他预测上升的项目中，njj的结果却没有统计学意义。其他结果有统计学意义的EV的预测图方向均是下降的，并且置信区间较小。elasticsearch的预测图中，虽然预测图上显示是下降的，但是odds ratio表中查到elasticsearch项目中njj的增长导致的odds ratio值变动为0%，说明这样的下降趋势不明显，我们不接受这样的预测结果。所以，我们可以认为，次要改动、主要评论、主要决策者对项目质量有积极作用。

Nnj：roslyn的置信区间过宽，趋势不明显，结果不予采用。Coreclr与kubernetes的预测图像方向较为明确，置信区间窄，结果予以采用。所以我们有理由认为，次要改动、次要评论、主要决策者对项目质量有积极作用。

Jjn：Coreclr项目的置信区间非常宽，说明预测结果可信度不高。同时，ansible的预测图中，曲线下降趋势明显的部分置信区间宽，结果可信度不高；置信区间窄的部分下降趋势非常不明显，所以，我们缺乏足够证据证明主要开发、主要评论和次要决策者对软件项目的质量有何种影响。

Jnn：预测图中，prestashop、roslyn、cocos2d\_x的预测图置信区间比较窄。至于其他项目，在ansible项目中，预测趋势是下降的，并且随着jnn值的增长置信区间在缩小；elasticsearch项目的预测图则中包含增长和下降两种趋势，其中增长的趋势置信区间较窄，下降的趋势置信区间较宽，我们暂不采信这个预测结果；kubernetes项目中也包含下降和上升两个趋势，下降趋势的置信区间虽然很窄，但是下降趋势非常不明显，后来的上升趋势虽然明显但是区间急剧变宽，这个预测结果我们暂不采信。所以综上所述，我们暂时认为，主要开发、次要评论、次要决策者对软件质量有一定的积极作用。

Njn：ansible和coreclr呈献下降趋势明显的部分置信区间太大，而置信区间较窄的部分下降趋势不明显。所以上述预测图的预测结果我们不予采用。而另外几个项目中，cocos2d\_x虽然区间在扩大，但是区间总体跨度还是很窄的，结果比较可信。Corefx和elasticsearch的预测图区间虽然较窄，但是跨度相比cocos2d\_x还是较宽。Roslyn的区间窄，并且趋势更加明显。所以我们比较倾向于相信，次要改动、主要评论、次要决策的用户对项目质量有积极影响。

Nnn：在baystation12、roslyn、prestashop三个项目中，预测图为增长趋势但是置信区间非常大，增长趋势也不是很明显，所以我们不轻易采信这样的预测结果。coreclr没有单调趋势并且前者是先减后增，后者先增后减，所以这里出现混淆，两者结论皆暂不予采用。而corefx项目当中，我们认为其增长趋势较为明显，置信区间也比较窄。至此，我们认为，次要改动、次要评论、次要决策者对项目的质量有不良影响。

#### 其他环境变量

如同5.4.1的研究方法，我们对其他的EV进行分析。

|  |  |  |
| --- | --- | --- |
| EV（环境变量） | 在哪些项目中该EV进入模型拟合步骤，并且其结果有统计学意义 | 参考其他项目的其他EV的预测图 |
| Contributor | - | coreclr的nnn, corefx的integrator, roslyn的integrator |
| Author | cocos2d\_x | coreclr的integrator, corefx的integrator, roslyn的integrator |
| Commentor | - | coreclr的nnn, corefx的integrator, roslyn的integrator |
| Integrator | cocos2d\_x, coreclr, corefx, kubernetes, prestashop, roslyn | - |
| Churn | coreclr, kubernetes, roslyn | cocos2d\_x的author |
| Entropy | coreclr, prestashop, roslyn | - |
| Top\_iso | corefx, roslyn | - |
| Top\_rso | cocos2d\_x, coreclr, corefx | - |
| Top\_tco | ansible, cocos2d\_x | coreclr的integrator |
| Size | cocos2d\_x, coreclr, corefx, kubernetes, roslyn | - |

Contributor：我们通过观察，发现coreclr当中，曲线方向不明确；roslyn当中曲线增长趋势不明显并且随着曲线的升高，置信区间在不断变大，说明支持曲线增长趋势的数据不够充足，我们不倾向于采信这样的预测结果。虽然corefx的图像比较明确，但是这不能成为contributor对项目质量影响的充足证据。所以我们无法认定贡献者的数量对项目质量是否有影响。

Author：我们发现，这些预测图中除了corefx的增长趋势较为可信之外，其他的预测图中曲线的置信区间非常宽，我们不倾向于采信这样的预测结果。所以，我们也无法判定代码改动者的数量对项目质量有什么样的影响。

Commentor：相关预测图的结果如author和contrubutor一样，我们依然无法的出比较准确的结论。

Integrator：排除掉coreclr、kubernetes、roslyn这三个预测图趋势不明、置信区间过大的项目后，剩下的三个项目的图像都比较可信。在这种情况下，我们更倾向于相信当决策者的数量上升的时候，项目出现缺陷的可能性会上升。

Churn：我们发现，所有的预测曲线的增长趋势不明显，并且置信区间急剧扩张，所以我们不能肯定模块中代码改动的总行数的多少对项目质量会造成何种影响。

Entropy：coreclr的预测图方向不明确，prestashop和roslyn的区间虽然较窄但是二者曲线方向完全相反。所以我们无法判定代码改动在模块中的分布情况对项目质量的影响。

Top\_iso： coreclr和kubernetes的图像趋势不明显且置信区间过大，我们不认为其预测结果有可信度；其他图像具有比较高的可信度并且趋势较为明显。可以看出，除了cocos2d\_x的项目图像都呈现上升趋势，我们有理由相信，在一个模块中决策比例最高的用户参与的决策的比例越大，项目质量越有可能受到不良影响。

Top\_rso：除了coreclr的图像上升下降趋势不明，cocos2d\_x和corefx的图像都具有较为明显的下降趋势并且置信区间较窄。由此我们有相信，在一个模块中评论比例最高的用户所做出的评论的比例越大越有利于阻止代码缺陷的产生。

Top\_tco：ansible和coreclr中，图像趋势相反，且都不明显并且置信区间非常宽；cocos2d\_x的图像则兼具上下趋势，所以我们很难判断，在一个模块当中做出最大比例的代码改动的用户改动代码的比例对项目质量的影响。

Size：Cocos2d\_x和kubernetes的预测曲线兼具上下行的部分，所以趋势不明。随着size值的变大，其他项目的预测曲线虽然都是上升趋势的，但是上升的趋势并没有特别明显，而且置信区间也随之变大。所以我们并没有充分的理由相信，模块中经过改动的文件的总代码行数越大，项目的就越有可能产生缺陷。

## 研究总结

通过指标设计、原始数据爬取、环境变量以及反应变量计算、模型拟合、具体项目研究、统计结果分析的全过程之后，我们着重总结了八种角色在各个模块中占据的比例对模块质量的影响。

### 观察结果

下表总结了本次研究中我们对八种角色对软件模块质量的影响的观察结果：

|  |  |  |
| --- | --- | --- |
| role | | 影响 |
| p1 | jjj | 主要改动、主要评论、主要决策者对项目质量有积极作用。 |
| p2 | jnj | 不明。 |
| p3 | njj | 次要改动、主要评论、主要决策者对项目质量有积极影响。 |
| p4 | nnj | 次要改动、次要评论、主要决策者对项目质量有积极影响。 |
| p5 | jjn | 不明。 |
| p6 | jnn | 主要开发、次要评论、次要决策者对软件质量有积极影响。 |
| p7 | njn | 次要改动、主要评论、次要决策者对项目质量有积极影响。 |
| p8 | nnn | 次要改动、次要评论、次要决策者对项目质量有不良影响。 |

--标红的EV能够对质量能够产生比较重要影响。

1. 改动、评论、决策三种行为当中，只有一种行为是主要行为而另外两种是次要行为的，对项目质量有积极影响。

仅仅主要做出改动行为的用户，他们可能对项目有更加深入的了解，所以他们所做代码改动更有品质保障。评论行为的用户的行为则应证了前人的研究当中关于软件评审行为的研究结论。

至于主要做决策行为的用户，他们往往是项目中最重要的人员，他们深刻地了解这个项目。他们专注于决策而不轻易做出改动和评论行为，这样有助于他们集中精力去管理为数众多的贡献请求以确保软件项目的品质。但是由于5.3段落所述，他们在模块贡献者中所占比例对项目质量的影响程度有限。所以，GitHub项目的开发者们不应该将项目品质保障的希望全部寄托在这些人身上，进而无限制地贡献存在缺陷隐患的代码改动。

1. 三种行为当中具有两种主要行为的用户当中，有主要改动行为的用户对项目质量的影响结果不明确，次要改动、主要评论、主要决策者对项目质量有积极影响。

三种行为中具有两种主要行为、并且改动行为为主要的用户，他们如果以主要参与者的身份参与了其他任一行为，他们对项目质量的影响就难以判断了。这样的原因可能是因为具有了主要的改动和另一行为的用户的特点较为复杂，他们在实际项目开发当中牵涉的事务可能比较繁杂。而次要开发、主要评论、主要决策的用户则对项目质量有积极影响，说明决策者如果少开发而多从事评审行为，还是有助于保证项目质量的。

1. 三种行为都是主要或次要的用户，前者对项目质量有积极影响，后者则对项目有较为明显的不良影响。

三种行为都是次要行为的用户，可能对项目的了解不深、较少参与到项目的评论当中。虽然他们并不贡献大量的贡献请求，但是他们一旦贡献了代码改动，就是缺陷隐患较大的改动。所以他们对项目造成的影响比较明显。

而对于其他的EV，我们能够得到如下的观察结果：

1. 代码改动本身的一些属性，例如模块中被改动的文件的总代码行数、代码改动行数总和、改动在模块中的分布情况、参与改动和评论的用户的数量、贡献者总数等，对项目质量的影响不明确。但是决策者数量上升却导致了项目出现缺陷的可能性上升。

我们认为出现这种情况的可能原因是具有决策权力的人如果为数太多，可能会导致决策意见分散，做出有害于项目质量的决定。

1. 在一个模块中决策比例最高的用户参与的决策的比例越大，项目质量越有可能受到不良影响；而一个模块中评论比例最高的用户所做出的评论的比例越大，越有利于阻止代码缺陷的产生。

我们认为出现这种情况的可能原因是：在一个模块中决策比例最高的用户是主要的决策者，这样的用户拥有极大评判贡献请求的权利，这样的人越主动，可能会导致其他的决策者越不主动，从而这样的用户倾向于独自判断贡献请求的质量。他们由此很难对海量的贡献请求一一做出正确决断。而在一个模块中评论比例最高的用户如果表现地越积极，越有助于项目开发的参与者（包括改动代码、评论和决策者）做出最有利于保证项目质量的行为。可能由于他们的积极行动，代码改动中的缺陷更容易地被察觉。

### 对项目开发和管理的建议

结合6.1的观察结果，我们得出了以下有利于保证GitHub开源项目代码质量的建议：

1. 贡献代码改动的GitHub用户，应当提高自己对于代码品质的要求，并减少贡献请求的数量，这将有助于决策者能够做出最正确的决策。
2. 项目决策者如果专注于贡献请求的评审，项目质量能够得到更大的保障。
3. 项目的决策者应当对不经常贡献代码、不常参与评论的普通用户所做出的贡献请求保持谨慎的态度，应较为严格地审查其代码改动的品质，考虑其引入缺陷的风险。
4. 在维护一个项目的过程中，决策者应当合理分配代码评审工作，避免一人专断的情况；同时也不宜过多决策者参与决策，以避免决策意见的分散、让低质量贡献请求成为漏网之鱼。

## 研究中存在的问题以及未来展望

本次研究当中存在一些问题可能会对所得结论有一定的影响。现特总结如下：

1. 我们在选择项目的时候，对项目本身的特性没有特别的关注，而仅仅是将其作为无差别的一般软件看待。例如某个项目是由什么语言所写，是用来满足什么需求的项目，我们并没有特别注意。如果能够有机会的话，我们会对项目本身的特性进行筛选，以求达到最一般化的研究结论。
2. 本文当中统计结果分析部分，存在一些根据经验判断的结果，有可能在不同的解读。

## 附录

按照repo来放图片。