软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2021,32(12):3698−3709 [doi: 10.13328/j.cnki.jos.006127]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

开源社区评审过程度量体系及其实证研究*

蒋 竞, 吴秋迪, 张 莉

(北京航空航天大学 计算机学院,北京 100191)

通讯作者: 张莉, E-mail: lily@buaa.edu.cn

摘 要: 在开源社区中,不同开发人员提交的代码水平参差不齐,需要代码评审检查提交代码质量.决策者是代码评审的关键人物,审核提交代码,发现软件缺陷.代码评审情况会对开源软件质量产生影响,因此需要建立评审过程度量体系,了解代码评审情况,促进提高开源软件项目质量.现有的软件过程度量方法主要考虑提交代码和评审评论活动,缺乏考虑决策活动,难以充分度量人员的评审行为.引入决策者因素,提出了一个开源社区评审过程度量体系,包括评审活动指标和人员分布指标.评审活动指标包含评审次数、评审信息长度、评审代码改动行数以及评审时间.人员分布指标主要考虑改动者、评论者和决策者的比例和数量.然后,收集了 3 个热门开源项目数据,分析评审过程度量指标与软件缺陷数量的关系.通过实证研究分析发现:决策者数量,少改动、少评论、少决策者的比例等决策者相关指标和软件缺陷数量中等正相关.同时,与不考虑决策者的度量体系进行对比分析,发现含有决策者的度量体系与软件缺陷的相关性更高.实证研究结果验证了评审过程度量体系的有效性,说明增加决策者相关指标的必要性.

关键词: 开源社区;代码评审;度量体系;决策者;软件缺陷

中图法分类号: TP311

中文引用格式: 蒋竞,吴秋迪,张莉.开源社区评审过程度量体系及其实证研究.软件学报,2021,32(12):3698−3709. http://www.jos.org.cn/1000-9825/6127.htm

英文引用格式: Jiang J, Wu QD, Zhang L. Open source community review process measurement system and its empirical research. Ruan Jian Xue Bao/Journal of Software, 2021,32(12):3698−3709 (in Chinese). http://www.jos.org.cn/1000-9825/6127.htm

Open Source Community Review Process Measurement System and Its Empirical Research

JIANG Jing, WU Qiu-Di, ZHANG Li

(School of Computer Science and Engineering, Beihang University, Beijing 100191, China)

Abstract: In the open source community, the code level of different developers varies, and code reviews are required to check the quality of the submitted code. Decision makers are the key persons in the code review, auditing the submitted code and finding software defects. Code reviews affect the quality of open source software. Therefore, it is necessary to establish code review process measurement system, understand the code review situation, and promote the quality of open source software projects. Existing software process measurement methods mainly consider code submission and review comments, but lack of consideration for decision making activities, and it is difficult to fully measure the review behavior. This study considers decision-maker factor, and proposes an open source community's review process measurement system, including evaluation activity indicators and personnel distribution indicators. Review activity indicators include numbers of review, length of review information, number of lines that code changes, and review time. The personnel distribution indicators mainly consider the proportion and number of modifiers, commenters, and decision makers. Then, this study collects data from three popular open source projects and analyzes the relationship between evaluation process metrics and the number of software defects. Through empirical research and analysis, it is found that the number of decision-makers, the proportion of

* 基金项目: 科技创新 2030——"新一代人工智能"重大项目(2018AAA0102304); 国家自然科学基金(61672078); 中央高校基本科研业务费(YWF-20-BJ-J-1018)

Foundation item: National Key Research and Development Program of China (2018AAA0102304); National Natural Science Foundation of China (61672078); Fundamental Research Funds for the Central Universities (YWF-20-BJ-J-1018)

收稿时间: 2020-02-27; 修改时间: 2020-07-11; 采用时间: 2020-08-07

decision-makers with few changes, few comments, and few decision-makers are moderately positively correlated with the number of software defects. At the same time, compared with the measurement system without the decision maker, it is found that the measurement system with the decision maker has a higher correlation with software defects. The results of the empirical study verify the effectiveness of review process measurement system, and illustrate the necessity of adding relevant indicators for decision makers.

Key words: open source community; code review; measurement system; decision maker; software defect

开源软件特有的开发模式激活大众创新潜力、提高创新效率.开源社区里的开发人员可以下载源码,并在本地代码库进行修改,协同进行软件开发.但是,不同开发人员提交的代码水平参差不齐,代码风格也不尽相同,需要代码评审检查提交代码质量[1].决策者是代码评审的关键人物,审核开发人员提交的代码,决定是否将其添加到源代码库.此外,开发人员也可以对代码改动和评审过程发表评论.代码评审要综合考虑代码质量、文档质量、测试结果、需求测试结果等因素.代码只有通过评审后,才能集成到源代码库中;未通过评审的代码会被开源项目拒绝,不会影响到开源软件的质量.代码评审能够提前发现项目存在的缺陷,减少了返工时间以及测试时间,保证开源软件质量.因此,需要建立评审过程度量体系,了解代码评审情况,发现容易出现软件缺陷的高风险软件模块,促进提高开源软件项目质量.

为了解软件开发情况,研究人员提出了过程度量方法.Bird 等人[2]提出了一种基于开发过程度量体系,分析不同人员在同一模块的代码贡献量,通过过程度量体系评估这些人员的开源软件开发过程,来分析开源开发过程与软件缺陷的关系.然而,他们的工作只针对了开源软件开发过程,并没有考虑评审过程.Thongtanunam 等人结合开发代码贡献量和评审评论贡献量,建立了一种结合评论的评审过程度量方法[3].然而,他们只考虑评审过程中的评论活动,忽略决策活动.在代码评审里,决策者才是起关键作用的角色,直接决定了代码改动是否能够加入源代码库里.

决策者是代码评审过程中的关键人物,以往的研究没有考虑决策者的活动.本文引入决策者因素,提出了一个开源社区评审过程度量体系,从多个指标的角度分析评审过程与软件缺陷数量的关系.首先建立度量体系的度量指标,分别是评审活动指标和人员分布指标.评审活动指标包含评审次数、评审信息长度、代码改动行数以及评审时间.人员分布指标里考虑改动者、评论者和决策者的比例和数量.然后收集 3 个热门开源项目数据,分析评审过程度量指标与软件缺陷数量的关系.实证研究发现:一些度量指标和软件缺陷数量至少中等正相关,例如决策者数量,少改动、少评论、少决策者的比例等.这些指标的数值越大,软件缺陷数量越多.同时,与不考虑决策者的度量体系进行对比分析,发现含有决策者的度量体系和软件缺陷的相关性更高.实证研究结果验证了本文提出评审过程度量体系的有效性,说明增加决策者相关指标的必要性.

本文的主要贡献包括:

(1) 首次引入决策者因素,考虑决策者在评审过程的重要作用,提出了一个开源社区评审过程度量体系;

(2) 实证研究发现:和现有的度量体系相比,发现含有决策者的度量体系和软件缺陷的相关性更高.

本文第 1 节介绍研究背景.第 2 节介绍相关研究工作.第 3 节介绍评审过程的度量指标.第 4 节进行实证研究,爬取真实数据,计算得出评审过程的各种指标,最后分析评审过程度量指标与软件缺陷的关系.第 5 节讨论有效性和论文启示.第 6 节总结全文.

1 背景介绍

Github 是一个著名的开源软件平台[4−8].Github 可以看做一个开源代码库,同时也能作为库的版本控制系统,因此有 900 多万的用户选择在 Github 上面进行软件开发.现如今,Github 已经成为了开发人员进行开源软件开发的首选.我们的所有研究也都是基于 Github 的.

如图 1 所示,以 GitHub 为例的开源软件平台代码评审[9−12]主要包括以下步骤.

(1) 开发人员可以拷贝开源项目的代码,建立副本,并在副本上进行代码修改.但是如果他们想向源代码库提交自己的代码,就需要发出一条贡献请求,其中包括要修改的代码.贡献请求就是开发人员提交的一组代码和文本描述.开发人员如果想将修改的代码整合到开源项目时,可以向开源项目提交贡献

请求.这些提交了贡献请求的开发人员被称为改动者;

(2) 这种贡献请求是公开的,任何人都可以对贡献请求发表评论.发表评论的人员被称作评论者;

(3) 决策者是可以评审代码、修改项目代码库的核心开发人员.决策者对贡献请求进行检查,并且可以参考评论里的意见,对贡献请求的质量进行评估.如果通过,则这段代码可以提交到源代码.在评审结束后,决策者关闭贡献请求.

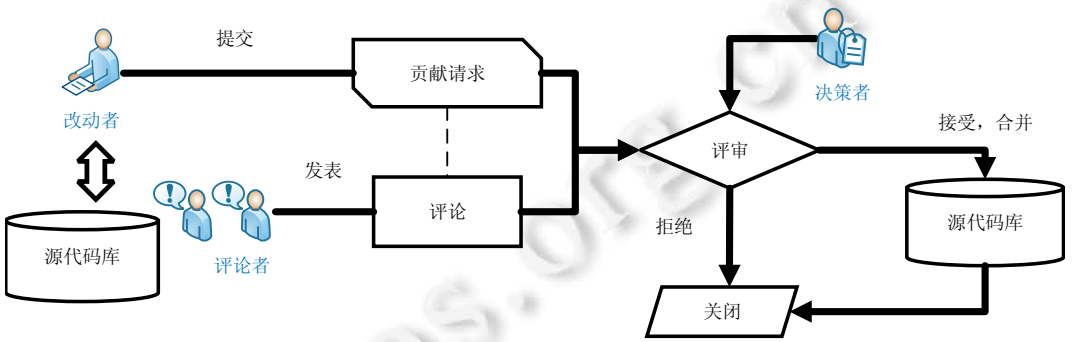

Fig.1 GitHub review process

图 1 GitHub 评审流程

从 Github 的评审流程可以看出:代码评审能够对提交的代码进行评估和改进,发现提交代码的缺陷.因此,代码评审对软件质量有着直接的影响.研究代码评审过程,能够更好地管理开源软件质量,减少软件缺陷.以往的研究[2,3]都没有考虑评审过程中决策者的作用,然而在评审过程中,决策者是代码评审的关键人物,只有经过决策者审核通过的代码才能加入到源代码库里.因此,本文的开源社区评审过程度量体系会考虑决策者因素.

2 相关研究

为了解软件开发情况,研究人员提出了过程度量方法.首先,Bird 等人[2]提出了一种基于软件开发的过程度量体系,通过计算开发人员在文件内编写的代码更改的比例,计算文件内开发人员的代码贡献量,来度量开发人员的软件开发过程.然后,根据过程度量体系的指标分析开发过程与软件缺陷的关系.然而,他们的工作只针对了开发人员在软件开发过程中的表现,并没有考虑评审过程对于软件缺陷的影响.其次,Thongtanunam 等人[3]研究了 QT 和 OpenStack 平台里改动者和评论者在评审过程中的贡献与分布.他们发现:许多评论者在传统指标里被定义为次要贡献者,但是在考虑了评论因素的指标里却被定义为主要贡献者.在无缺陷的文件里,有着相当大比例的开发者在传统指标里的贡献量比较低,而在引入评论因素的指标里贡献量比较高.他们的研究表明:传统的贡献量指标会忽略评论贡献,然而无缺陷的软件却拥有更多的高评论贡献量开发人员,说明传统的贡献量指标不足够.但是 Thongtanunam 等人只计算了评论者和改动者对于软件缺陷的影响,缺乏考虑评审过程的重要角色决策者.在开源软件开发过程中,决策者发挥重要作用,决定代码改动是否能否集成到开源项目.现有文献[2,3]缺乏考虑决策者,难以充分度量评审过程.本文引入决策者因素,分析度量指标与软件缺陷数量的关系.

贡献请求的代码评审近年来得到了广泛的关注.首先,一些学者研究贡献请求的决策者或者评论者推荐方法.莫纳什大学夏鑫等人提出了一种结合文件路径和文本描述的决策者推荐方法[13].Kim 等人提出了基于主题模型的决策者推荐方法.Thongtanunam 等人提出了基于文件路径的决策者推荐方法[14].国防科技大学的余跃、王怀民等人分析了开源软件平台的历史数据,建立了评论关系网络,对贡献请求推荐合适的评论者[15−17].中国科学院的卢松等人提出了基于时间和影响力因子的评论者推荐方法[18].其次,一些研究人员对大众贡献评审结果展开研究.Weigerber 等人发现,修改量小的代码更容易通过评审[19].Bosu 等人发现,项目核心人员提交的代码质量更高,更容易通过评审[20].Gousios 等人提出一种基于随机森林的贡献结果预测方法[21].Tsay 等人结合社交因素和技术因素预测评审结果[22].虽然这些论文研究问题和本文不同,但是都关注代码评审的贡献请求,研究思路

和指标值得本文借鉴.

3 评审过程度量体系

代码评审是保证开源软件质量的重要环节.本文研究评审过程度量体系,对开源项目的评审过程进行评估.参考以往论文[3],本文提出两个方面的指标:首先是评审活动指标,用来评估评审活动的情况,包括评审时间、评审次数、评审信息长度、代码改动行数;然后是人员分布指标,反映改动者、评论者和决策者的分布情况.

项目里不同的文件代码规模悬殊,根据文件来计算度量指标差异会很大.根据文献[3]使用模块来确定统计对象,对同属于一个子文件夹下的文件计算指标.对于一个模块 M,$FSet_M$ 是模块 M 里的文件集合,$PSet_M$ 是改动过模块 M 的贡献请求集合,$DSet_M$ 是改动、评论或者决策过模块 M 的用户集合,其中,$CSet_M$ 是改动模块 M 的改动者集合,$RSet_M$ 是评论模块 M 的评论者集合,$ISet_M$ 是评审模块 M 的决策者集合.对于一个贡献请求 P_i($P_i \in PSet_M$),定义其评论条数为 $I(P_i)$,评审时间为 $T(P_i)$.对于一个文件 F_i($F_i \in FSet_M$),$a(F_i)$,$d(F_i)$,$m(F_i)$分别为文件 F_i 里增加、删除、修改的代码行数.对于一个用户 D_i($D_i \in DSet_M$),$CH(D_i,M)$为用户 D_i 对模块 M 做出的改动的行数,$RE(D_i,M)$是用户 D_i 在模块 M 里的评论总数,$PR(D_i,M)$为 D_i 在模块 M 里决策了的贡献请求数目.

3.1 评审活动

定义 1(评审次数). 评审次数是一个模块所经历的所有评审次数.对于一条被整合进入源代码库的贡献请求,它的代码改动必然涉及到某些模块.每有一条贡献请求改动了模块 M,就称模块 M 经历了一次评审.$PSet_M$ 是改动过模块 M 的贡献请求集合,$card(PSet_M)$是 $PSet_M$ 集合里的贡献请求数目,就是模块 M 的评审次数.评审次数 N 为

$$N=card(PSet_M) \tag{1}$$

定义 2(评审信息长度). 评审信息是评审过程中的评论条数.在评审一个贡献请求的时候,人们可以对其提出自己的批注和意见.评审信息反映改动的代码是否经过了大量讨论.对于一个模块的评审信息长度,定义为这个模块中每条贡献请求的评论条数之和.M 是选定的一个模块.则评审信息长度 L 为

$$L=\sum_{P_i \in PSet_M} I(P_i) \tag{2}$$

定义 3(代码改动行数). 代码修改最重要的特征之一,就是代码改动的行数,包括代码的增添、删减和修改行数.这些属性是一个代码改动核心的特征,通过代码的增添、删减和改动行数,可以清楚地知道软件在一段时间内究竟有多少改动.一个模块的代码改动行数 $CH(M)$,就是这个模块中所有文件的增添、删除和修改的代码行数:

$$CH(M)=\sum_{F_i \in FSet_M} a(F_i)+d(F_i)+m(F_i) \tag{3}$$

定义 4(评审时间). 一个贡献请求从创建开始,一直到评审结束持续的时间.评审时间以模块为单位,计算一个模块里贡献请求的评审时间之和.M 是选定的一个模块.则评审时间 T 定义为

$$T=\sum_{P_i \in PSet_M} T(P_i) \tag{4}$$

3.2 人员分布

定义 5(改动者数量). 假如一个文件里面,提交代码修改的人数越多,那么提交的代码风格可能差异越大.找出模块的贡献请求集合中所有的改动者集合,计算集合里的改动者数目.改动者数量 C 为

$$C=card(CSet_M) \tag{5}$$

定义 6(评论者数量). 在评审过程中,评论者对评审结果有直接影响.评论者可以对代码修改提出自己的意见,比如指出代码修改里面的不足,而这些评论可以作为后面决策工作的参考.找出模块的贡献请求集合中所有的评论者集合,计算集合里的评论者数目.评论者数量 R 为

$$R=card(RSet_M) \tag{6}$$

定义 7(决策者数量). 决策者是能够直接决定代码修改是否能整合进源代码库的人员,决策者是直接决定评审结果的人员.根据模块的贡献请求集合计算所有的决策者数目.决策者数量 I 为

$$I=card(ISet_M) \tag{7}$$

在 Patanamon 的论文[3]里,他们提出了贡献量这一说法.贡献量就是评估开发人员给项目做的贡献多少,比如作者的贡献量就是修改代码的多少、评论者贡献量就是评论数目的多少.代码贡献量为大型软件系统中的模块建立了一系列的责任链.虽然之前的工作揭示了代码贡献量、评论贡献量与软件缺陷之间的联系,但这些启发式仅依赖于代码更改的作者和评论者.除了编写代码更改和评论之外,还有决策者对模块做出重要贡献.决策者可以决定代码修改是否能被整合进源代码库中,作为评审活动中重要的一环,将决策者也加入到贡献量计算中.提出了 3 种贡献量的计算方式.

- 代码贡献量

作者是最基本的一种开发人员,给项目添加代码改动的开发人员就是作者.作者提交的代码改动是评审的基础和对象.而代码贡献量就是评估作者对项目的贡献.

定义 8(代码贡献量). M 是选定的一个模块,那么用户 D_i 在模块 M 中所占的代码贡献量 $TCO(D_i,M)$为

$$TCO(D_i,M)=\frac{CH(D_i,M)}{\sum_{D_i\in DSet_M}CH(D_i,M)} \tag{8}$$

- 评论贡献量

评论者虽然不能直接影响评审结果,但是评论者的评论可以作为决策者提供参考.评论数目反映一条代码修改是否经历了大量的讨论.

定义 9(评论贡献量). M 是选定的一个模块,那么用户 Di 在模块 M 中所占的评论贡献量 $RSO(D_i,M)$为

$$RSO(D_i,M)=\frac{RE(D_i,M)}{\sum_{D_i\in DSet_M}RE(D_i,M)} \tag{9}$$

- 决策贡献量

决策者是可以把代码修改整合进入源代码库的内部人员.过去的论文都没有讨论过决策者的作用,然而决策者作为直接评审的人员,肯定是不容忽视的.

定义 10(决策贡献量). M 是我们研究的一个模块,那么用户 D_i 在模块 M 中所占的决策贡献量 $ISO(D_i,M)$为

$$ISO(D_i,M)=\frac{PR(D_i,M)}{\sum_{D_i\in DSet_M}PR(D_i,M)} \tag{10}$$

各种类型的人员并不是完全独立的,评审过程中的开发人员可能同时扮演几种角色.为了准确地分析评审过程里的人员组成,需要根据不同的贡献量将人员进行划分,并将模块中不同类型人员的比例作为度量指标.文献[2,3]以 0.05 为阈值,按照贡献量划分开发人员的角色.类似的,本文也按贡献量来划分主要、次要贡献者.在一个模块中,如果一个人某个方面的贡献量比例大于等于 0.05,则称他是这个方面的主要贡献者.根据人员在改动、评论或者决策的贡献量比例来给他们划分角色.例如一个人 $TCO\geqslant 0.05$,$RSO\leqslant 0.05$,$ISO\geqslant 0.05$,那么他是主要代码改动贡献者、次要评论贡献者、主要决策贡献者.由此可以将用户分为 8 类,并计算这 8 类用户在模块里的比例作为度量指标.

- 多改动、多评论、多决策者的比例:$TCO\geqslant 0.05$,$RSO\geqslant 0.05$,$ISO\geqslant 0.05$ 的人员比例;
- 多改动、少评论、多决策者的比例:$TCO\leqslant 0.05$,$RSO\geqslant 0.05$,$ISO\geqslant 0.05$ 的人员比例;
- 少改动、多评论、多决策者的比例:$TCO\geqslant 0.05$,$RSO\leqslant 0.05$,$ISO\geqslant 0.05$ 的人员比例;
- 少改动、少评论、多决策者的比例:$TCO\leqslant 0.05$,$RSO\leqslant 0.05$,$ISO\geqslant 0.05$ 的人员比例;
- 多改动、多评论、少决策者的比例:$TCO\geqslant 0.05$,$RSO\geqslant 0.05$,$ISO\leqslant 0.05$ 的人员比例;
- 多改动、少评论、少决策者的比例:$TCO\leqslant 0.05$,$RSO\geqslant 0.05$,$ISO\leqslant 0.05$ 的人员比例;
- 少改动、多评论、少决策者的比例:$TCO\geqslant 0.05$,$RSO\leqslant 0.05$,$ISO\leqslant 0.05$ 的人员比例;

- 少改动、少评论、少决策者的比例:$TCO \leqslant 0.05, RSO \leqslant 0.05, ISO \leqslant 0.05$ 的人员比例.

表 1 中给出了评审过程度量指标的定义和描述,这些度量指标从评审活动和人员分布两方面诠释了软件评审过程中的多维属性.

Table 1 Review process metrics

表 1 评审过程度量指标

指标		描述
评审活动	评审次数	一个模块的贡献请求数量
	评审信息长度	一个模块里贡献请求的评审信息条数之和
	代码改动行数	一个模块代码增加、删除和修改的行数
	评审时间	一个模块里贡献请求的评审时间之和
人员分布	改动者数量	一个模块中的改动者者人数
	评论者数量	一个模块中的评论者人数
	决策者数量	一个模块中的决策者人数
	多改动、多评论、多决策者的比例	$TCO \geqslant 0.05, RSO \geqslant 0.05, ISO \geqslant 0.05$ 的人员的比例
	多改动、少评论、多决策者的比例	$TCO \leqslant 0.05, RSO \geqslant 0.05, ISO \geqslant 0.05$ 的人员的比例
	少改动、多评论、多决策者的比例	$TCO \geqslant 0.05, RSO \leqslant 0.05, ISO \geqslant 0.05$ 的人员的比例
	少改动、少评论、多决策者的比例	$TCO \leqslant 0.05, RSO \leqslant 0.05, ISO \geqslant 0.05$ 的人员的比例
	多改动、多评论、少决策者的比例	$TCO \geqslant 0.05, RSO \geqslant 0.05, ISO \leqslant 0.05$ 的人员的比例
	多改动、少评论、少决策者的比例	$TCO \leqslant 0.05, RSO \geqslant 0.05, ISO \leqslant 0.05$ 的人员的比例
	少改动、多评论、少决策者的比例	$TCO \geqslant 0.05, RSO \leqslant 0.05, ISO \leqslant 0.05$ 的人员的比例
	少改动、少评论、少决策者的比例	$TCO \leqslant 0.05, RSO \leqslant 0.05, ISO \leqslant 0.05$ 的人员的比例

4 实证研究

为了验证度量体系的有效性和实用性,同时为了进一步研究评审过程指标与软件缺陷的相关性,本文采用 Github 的真实数据来进行实证研究.如图 2 所示,参考 Patanamon 的论文[23],本文把数据分成 3 部分:上一个时间段、当前时间段、下一个时间段.每个时间段都是 6 个月.上一个时间段的数据用来计算之前的软件缺陷;当前时间段的数据用来计算当前的评审过程指标;下一个时间段的数据用来计算评审后的软件缺陷.然后,基于度量评估体系对原始数据进行加工处理,计算得出各种指标,最后使用相关性分析方法研究评审过程度量指标与软件缺陷的关系.相关系数的绝对值越大,说明度量指标和软件缺陷数量越相关,度量指标越有效.

具体来讲,本文从两个方面研究度量指标和软件缺陷的相关性.首先,计算当前时间段的代码评审度量指标和下一个时间段软件缺陷数量的关系,来研究当前的评审过程是影响下一个时间段的软件缺陷.比如:当前时间的评审时间更长,下一个时间段的软件缺陷是否更少.然后,计算上一个时间段的软件缺陷数量与当前时间段的评审过程度量指标的相关性,研究之前存在的软件缺陷是否会对当前的代码评审过程造成影响.比如:上一个时间段软件缺陷数量多的模块,是否会在当前代码评审的时间更长.

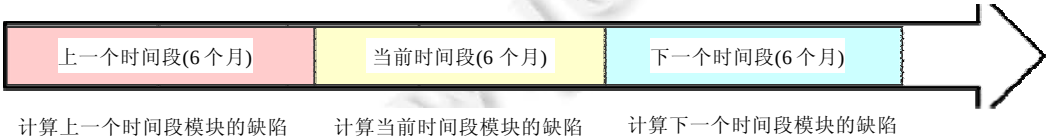

Fig.2 Data partition

图 2 数据划分

4.1 研究过程

4.1.1 项目选择

根据 Github 上项目的关注人数和贡献请求数,本文选取 3 个热门项目:xbmc,roslyn,elasticsearch.xbmc (https://github.com/xbmc/xbmc)是一款优秀的免费开源媒体中心软件,可以播放几乎所有的音频和视频格式.roslyn(https://github.com/dotnet/roslyn)是微软开发的一个开源的.NET 编译器,支持 C#和 Visual Basic.elasticsearch(https://github.com/elastic/elasticsearch)是一个基于 Lucene 的搜索服务器,提供了一个分布式的搜索

引擎.这些热门项目关注人数多,提交的贡献请求也多,分析结果才具有统计意义.

4.1.2 数据收集

采用的是 Github 自带的 API 进行数据获取.爬取的内容包贡献请求信息,这里指贡献请求编号、代码提交编号、创建时间、提交时间以及贡献请求下面的评论信息等.这些数据包含了一个贡献请求的所有信息,也可以将贡献请求与代码提交联系起来.接下来爬取代码提交信息,包括改动行数、文件名、改动者、决策者、评论信息、对应的贡献请求编号.这些数据文件包含了代码提交的所有信息,可以知道每条代码提交对应的模块.最后收集各种评论信息,包括了评论对应的贡献请求编号、评论者、评论条数.表 2 统计了 3 个项目的数据.本文将数据集上传,以供研究者参考(https://github.com/wqdbuaa/Dataset).

Table 2 Project data statistics

表 2 项目数据统计

项目名称	贡献请求数	贡献请求评论数	改动者数	评论者数	决策者数
xbmc	2 355	11 082	218	288	85
roslyn	3 863	17 999	147	198	81
elasticsearch	3 423	10 731	473	415	133

4.1.3 软件缺陷

软件缺陷又被称作 BUG,即为计算机软件或程序中存在的某种破坏正常运行能力的问题、错误,或者隐藏的功能缺陷.缺陷的存在,会导致软件产品在某种程度上不能满足用户的需要.一个软件的质量是跟它的软件缺陷相挂钩的.软件缺陷多,说明这个软件质量差.本文采用的是 Ray 在其论文[24]中使用的一种启发式的方法来对缺陷进行识别.找出一个贡献请求的提交信息,利用自然语言处理把提交信息进行分词,最后得到的词汇集合与错误词集进行匹配[25].如果提交信息里面包含错误关键词,那么可以认为这条贡献请求有缺陷,该条贡献请求所涉及的文件也视为有缺陷[24].Ray 在文献[24]中使用的错误关键字为 error,bug,fix,issue,mistake,incorrect,fault,defect,flaw,patch,本文采用同样的错误关键词.

4.1.4 相关性分析

本文采用 Spearman 系数检验来分析度量体系指标与软件缺陷的相关性.Spearman 系数检验能够衡量两个变量之间的依赖性,利用单调的方程来评价变量之间的相关性.如果两个变量完全单调相关,那么相关系数为+1 或者–1.其中,+1 表示正相关,–1 表示负相关,0 表示完全不相关.若相关系数在 0.7 到 1.0 之间,为强相关;0.3 到 0.7 为中等相关;0 到 0.3 为弱相关[26].相关性系数的绝对值越大,说明相关性越大、度量体系指标越有效.为了评估 Spearman 系数的显著性,计算 p 值(p value)来分析结果是否显著有效.使用不同个数的*来表示显著性值的大小等级:***代表 p 值<0.001,即极其显著的统计学差异;**代表 p 值<0.01,即有显著统计学差异;*代表 p 值<0.05,即有统计学的差异;没有任何*表示 p 值>0.05,即缺乏统计学的差异.

4.2 研究结果

4.2.1 当前时间段的代码评审过程和下一个时间段软件缺陷的关系

为了研究开源社区评审过程度量体系的有效性,本节分析当前时间段的度量指标和下一时间段软件缺陷的关系.首先,根据当前时间段的数据计算了当前时间段的项目中每个模块在度量体系里的各个指标;然后,通过错误词集匹配,计算下一个时间段每个模块中的缺陷数目;最后,通过 Spearman 系数检验来计算当前时间段的度量指标和下一时间段软件缺陷的相关系数,得出表 3.如果 3 个项目的相关性结论一致,那么在第 6 列标出具体的结论.如果 3 个项目的相关性不一致,那么结论是不确定.

从表 3 的数据可以看出:在 Xbmc 和 Roslyn 项目里,评审活动指标(评审次数、评审信息长度、评审时间、代码改动行数)与软件缺陷的相关系数都在 0.4 到 0.7 之间,即中等正相关;elasticsearch 项目中,评审活动指标(评审次数、评审信息长度、评审时间、代码改动行数)与软件缺陷的相关系数都在 0.7 以上,呈高等正相关.这说明度量体系中的评审活动指标与软件缺陷存在较强的相关性.当前时间段的评审次数越多、评审信息长度越大、评审时间越长、代码改动行数越多,下一个时间段的模块缺陷越多.在计算相关系数时,本文还计算了 p 值,

分析结果是否显著.表 3 的结果表明:大多数 p 值都小于 0.001,结果具有显著的统计学意义.

Table 3 Relationship between current time period index and next time period module defect

表 3 当前时间段指标和下一个时间段模块缺陷的相关系数

指标		Xbmc	Roslyn	elasticsearch	相关性
评审活动	评审次数	0.552***	0.639***	0.792***	不确定
	评审信息长度	0.450***	0.477***	0.707***	不确定
	评审时间	0.489***	0.487***	0.702***	不确定
	代码改动行数	0.416***	0.405***	0.568***	中等正相关
人员分布	改动者数量	0.440***	0.395***	0.651***	中等正相关
	评论者数量	0.414***	0.426***	0.633***	中等正相关
	决策者数量	0.411***	0.407***	0.656***	中等正相关
	多改动、多评论、多决策者的比例	0.132***	−0.262***	−0.177***	不确定
	多改动、少评论、多决策者的比例	0.116**	−0.013	0.263***	不确定
	少改动、多评论、多决策者的比例	0.136***	0.067**	0.240***	弱正相关
	少改动、少评论、多决策者的比例	−0.092***	−0.025	−0.005	弱负相关
	多改动、多评论、少决策者的比例	0.140***	0.094***	0.070**	弱正相关
	多改动、少评论、少决策者的比例	0.271***	0.261***	0.429***	不确定
	少改动、多评论、少决策者的比例	−0.073***	−0.290***	−0.316***	不确定
	少改动、少评论、少决策者的比例	0.418***	0.402***	0.585***	中等正相关

在人员分布指标中,改动者数量,评论者数量,决策者数量,少改动、少评论、少决策者的比例也和软件缺陷成中等正相关性,相关系数基本在 0.4 到 0.7 之间.另外,多改动、少评论、少决策者的比例和软件缺陷数量的相关性系数接近 0.3,接近中等正相关.

从表 3 可以看出:少改动、多评论、少决策者的比例与软件缺陷数量相关系数接近在−0.1 到−0.3 之间,表现为负相关性;而多改动、少评论、少决策者的比例与软件缺陷数量相关系数在 0.2 到 0.4 之间,接近中等正相关性.可以看出:开发人员并不只是通过代码改动来参与软件的开发,开发人员还可以通过评论和评审来对开源项目做贡献.多改动但是评论和决策贡献量小的开发者的比例,反而与软件缺陷成正相关;少改动但是评论和决策贡献量大的开发者的比例,与软件缺陷成负相关.如果度量体系不考虑评论者和决策者,那么少改动、多评论、少决策者的比例与软件缺陷的相关性就会被忽视掉,这说明本文在设计评估体系时考虑评审指标是合理的.

与现有工作[2,3]相比,本文提出的评审过程度量体系增加了决策者指标.为了分析决策者指标的有效性,本文分别统计只考虑改动者和评论者的度量体系、只考虑改动者的度量体系与软件缺陷数量的关系,结果详见表 4 和表 5.

Table 4 Relationship of modifier and commentator between current time period index and next time period module defect

表 4 只考虑改动者和评论者的当前时间段指标和下一个时间段模块缺陷的相关系数

指标		xbmc	Roslyn	elasticsearch	相关性
评审活动	评审次数	0.552***	0.639***	0.792***	不确定
	评审信息长度	0.450***	0.477***	0.707***	不确定
	评审时间	0.489***	0.487***	0.702***	不确定
	代码改动行数	0.416***	0.405***	0.568***	中等正相关
人员分布	改动者数量	0.440***	0.395***	0.651***	中等正相关
	评论者数量	0.414***	0.426***	0.633***	中等正相关
	多改动、多评论者的比例	0.105***	−0.254***	−0.187***	不确定
	多改动、少评论者的比例	0.266***	0.093***	0.327***	不确定
	少改动、多评论者的比例	−0.026	−0.279***	−0.303***	不确定
	少改动、少评论者的比例	0.050	0.339***	0.318***	不确定

Table 5 Relationship of modifier between current time period index and next time period module defect

表 5 只考虑改动者的当前时间段指标和下一个时间段模块缺陷的相关系数

指标		Xbmc	Roslyn	elasticsearch	相关性
评审活动	评审次数	0.552***	0.639***	0.792***	不确定
	评审信息长度	0.450***	0.477***	0.707***	不确定
	评审时间	0.489***	0.487***	0.702***	不确定
	代码改动行数	0.416***	0.405***	0.568***	不确定
人员分布	改动者数量	0.440***	0.395***	0.651***	不确定
	多改动者的比例	0.138***	−0.235***	−0.160***	不确定
	少改动者的比例	−0.136***	0.237***	0.161***	不确定

表 3 考虑了决策者的人员分布指标中,决策者数量,多改动、少评论、少决策者的比例,少改动、少评论、少决策者的比例与软件缺陷数量都接近中等正相关.相比于表 3,表 4 只考虑改动者和评论者的人员分布指标里,多改动、少评论者的比例,少改动、多评论者的比例,少改动、少评论者的比例都和软件缺陷数量接近弱相关,达不到中等相关;表 5 只考虑改动者的人员分布指标的相关性无法确定.本文提出的度量指标与软件缺陷数量存在较强的相关性.这说明了考虑决策者因素的度量体系的适用性和有效性.

4.2.2 上一个时间段的软件缺陷数量与当前时间段的评审过程的相关性

本文进一步研究当前时间段的评审活动是否会受上一个时间段模块缺陷的影响.探究对于上一个时间段有缺陷的模块,开发人员是否在当前时间段的代码评审中给了它们足够的关注和评审,来解决它们的遗留问题.首先,根据当前时间段的数据计算了当前时间段的项目中每个模块在度量体系里的各个指标;然后,通过错误词集匹配,计算上一个时间段每个模块中的缺陷数目;最后,通过 Spearman 系数检验来计算当前时间段的度量指标和上一时间段软件缺陷的相关系数,得出表 6.

Table 6 Relationship between current time period index and previous time period module defect

表 6 当前时间段的指标和上一个时间段模块缺陷的相关系数

指标		xbmc	roslyn	elasticsearch	相关性
评审活动	评审次数	0.611***	0.549***	0.648***	中等正相关
	评审信息长度	0.547***	0.465***	0.571***	中等正相关
	评审时间	0.559***	0.450***	0.559***	中等正相关
	代码改动行数	0.515***	0.463***	0.562***	中等正相关
人员分布	改动者数量	0.647***	0.653***	0.671***	中等正相关
	评论者数量	0.614***	0.607***	0.663***	中等正相关
	决策者数量	0.678***	0.647***	0.674***	中等正相关
	多改动、多评论、多决策者的比例	0.294***	−0.316***	−0.230***	不确定
	多改动、少评论、多决策者的比例	0.067	0.095***	0.281***	弱正相关
	少改动、多评论、多决策者的比例	0.167***	0.139***	0.229***	弱正相关
	少改动、少评论、多决策者的比例	−0.012	0.046	0.030	不确定
	多改动、多评论、少决策者的比例	0.107**	0.251***	0.042	弱正相关
	多改动、少评论、少决策者的比例	0.424***	0.435***	0.420***	中等正相关
	少改动、多评论、少决策者的比例	−0.110**	−0.463***	−0.368***	不确定
	少改动、少评论、少决策者的比例	0.603***	0.520***	0.621***	中等正相关

从表 6 的数据可以看出:3 个项目的评审活动指标(评审次数、评审信息长度、评审时间、代码改动行数)与软件缺陷的相关系数在 0.4~0.7 之间,呈现中等正相关.这说明度量体系中的评审活动指标与软件缺陷存在较强的相关性.上一个时间段缺陷越多的模块,在当前时间段的评审次数越多、评审信息长度越大、评审时间越长、代码改动行数越多.在计算相关系数时,本文还计算了 p 值,分析结果是否显著.表 6 的结果表明,大多数 p 值都小于 0.001.结果具有显著的统计学意义.

人员分布指标中,改动者数量,评论者数量,决策者数量,多改动、少评论、少决策者的比例,少改动、少评论、少决策者的比例也和软件缺陷成中等正相关性,相关系数在 0.4~0.7 之间.这说明本文的度量体系的决策者数量指标的有效性,在人员分布指标中考虑决策者数量是合理的.

为了研究是否考虑决策者,本文统计分别统计只考虑改动者和评论者的度量体系、只考虑改动者的度量体系与软件缺陷数量的关系,结果详见表 7 和表 8.

Table 7 Relationship of modifier and commentator between current time period index and next time period module defect

表 7 只考虑改动者和评论者的当前时间段指标和上一个时间段模块缺陷的相关系数

指标		xbmc	roslyn	elasticsearch	相关性
评审活动	评审次数	0.611***	0.549***	0.648***	中等正相关
	评审信息长度	0.547***	0.465***	0.571***	中等正相关
	评审时间	0.559***	0.450***	0.559***	中等正相关
	代码改动行数	0.515***	0.463***	0.562***	中等正相关
人员分布	改动者数量	0.647***	0.653***	0.671***	中等正相关
	评论者数量	0.614***	0.607***	0.663***	中等正相关
	多改动、多评论者的比例	0.174***	–0.273***	–0.249***	不确定
	多改动、少评论者的比例	0.359***	0.250***	0.339***	不确定
	少改动、多评论者的比例	–0.061	–0.420***	–0.358***	不确定
	少改动、少评论者的比例	0.156***	0.455***	0.385***	不确定

Table 8 Relationship of modifier between current time period index and next time period module defect

表 8 只考虑改动者的当前时间段指标和上一个时间段模块缺陷的相关系数

指标		xbmc	roslyn	elasticsearch	相关性
评审活动	评审次数	0.611***	0.549***	0.648***	中等正相关
	评审信息长度	0.547***	0.465***	0.571***	中等正相关
	评审时间	0.559***	0.450***	0.559***	中等正相关
	代码改动行数	0.515***	0.463***	0.562***	中等正相关
人员分布	改动者数量	0.647***	0.653***	0.671***	中等正相关
	多改动者的比例	0.165***	–0.198***	–0.213***	不确定
	少改动者的比例	–0.162***	0.198***	0.216***	不确定

表 6 里考虑了决策者的人员分布指标中,多改动、少评论、少决策者的比例,少改动、少评论、少决策者的比例与软件缺陷数量相关系数在 0.4 到 0.7 之间,呈现较强的正相关性;相比于表 6,表 7 只考虑改动者和评论者的人员分布指标里,多改动、少评论者的比例,少改动、多评论者的比例,少改动、少评论者的比例和软件缺陷数量的相关性无法确定;表 8 只考虑改动者人员分布指标的相关系数绝对值基本都小于 0.3.本文提出的度量指标与软件缺陷数量存在较强的相关性.这充分说明本文考虑了决策者因素的度量体系的有效性和合理性.

5 讨 论

5.1 有效性分析

本文分别从外部有效性和结构有效性的角度讨论对有效性的威胁.

- 外部有效性.本文只在 Github 上选取了部分项目作为数据集,因此,本文的结果可能不适用于所有开源社区.以后可以考虑分析更多 GitHub 的项目或者其他开源社区项目,进一步分析评审过程度量体系的有效性;
- 结构有效性.本文爬取的数据都是 Github 记录的代码评审活动,然而开发人员可能会通过其他方式进行代码评审,例如当面讨论或收发邮件.然而,这些方式都没有显式的记录和数据.因此,本文研究的是有相应的代码更改和评审记录的模块,包含项目中大部分的评审活动.另外,本文主要考虑代码评审过程度量指标.未来考虑尝试更多度量指标并分析其和软件缺陷的关系,比如考虑评估代码评审质量体系的方法以及相关指标.

5.2 启 示

表 4 中只考虑了改动者和评论者,没有考虑决策者,结果显示:多改动、少评论者的比例,少改动、多评论者

的比例,少改动、少评论者的比例都和软件缺陷数量接近弱相关,达不到中等相关;表 5 里只考虑改动者,人员分布指标与软件缺陷数量的相关性无法确定.而表 3 中,考虑了决策者因素的度量指标与软件缺陷数量存在较强的相关性.因此,在设计评审过程度量体系时,应该考虑决策者因素.

表 3~表 5 的结果显示,有的开发人员主要通过决策来参与软件开发.他们在传统的指标体系中被定义为少改动者、少评论者,和软件缺陷数量的相关性无法确定.但是在考虑了决策者因素后,他们反而因为多决策活动而成为了软件开发的主要贡献人员.因此,建议未来评审相关的研究考虑决策者和决策活动.

从表 3 看出,一些度量指标和软件缺陷数量至少中等正相关,包括评审次数,评审信息长度,评审时间,代码改动行数,改动者数量,评论者数量,决策者数量,少改动、少评论、少决策者的比例.这些指标的数值越大,软件缺陷数量越多.因此,开源项目需要重点观察少改动、少评论、少决策者的比例等指标高的模块,这些高风险模块往往有更多的软件缺陷.

6 总结与展望

在开源开发中,不同的开发人员的代码水平参差不齐.代码评审是保证开源项目质量的重要方式.本文提出了一个开源社区评审过程度量体系,包括评审活动指标和人员分布指标.该度量体系综合考虑人员在代码改动、评审决策和发表评论的活动.然后,本文分析了评审过程与软件缺陷之间的关系.通过与不考虑决策者的度量指标进行对比分析,验证了本文提出评审过程度量体系的有效性,说明了增加决策者相关指标的必要性.

References:

[1] McIntosh S, Kamei Y, Adams B, *et al*. The impact of code review coverage and code review participation on software quality. In: Proc. of the MSR. 2014.

[2] Bird C, Nagappan N, Murphy B. Don't touch my code! Examining the effects of ownership on software quality. In: Proc. of the 19th SIGSOFT/FSE ACM SIGSOFT Symp. on the Foundations of Software Engineering (FSE-19) and the 13rd European Software Engineering Conf. (ESEC 2011). 2011.

[3] Thongtanunam P, Mcintosh S, Hassan AE. Revisiting code ownership and its relationship with software quality in the scope of modern code review. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Software Engineering. 2016.

[4] Dabbish L, Stuart C, Tsay J, *et al*. Social coding in Github: Transparency and collaboration in an open software repository. In: Proc. of the ACM 2012 Conf. on Computer Supported Cooperative WorkACM. 2012.

[5] Gousios G, Zaidman A, Storey MA, *et al*. Work practices and challenges in pull-based development: The integrator's perspective. In: Proc. of the ICSE. 2015.

[6] Casalnuovo C, Vasilescu B, Devanbu P. Developer onboarding in GitHub: The role of prior social links and language experience. In: Proc. of the 2015 10th Joint Meeting on Foundations of Software Engineering. 2015.

[7] Tsay J, Dabbish L, Herbsleb J. Influence of social and technical factors for evaluating contribution in GitHub. In: Proc. of the 36th Int'l Conf. on Software Engineering. 2014.

[8] Gharehyazie M, Posnett D, Filkov V. Social activities rival patch submission for prediction of developer initiation in oss projects. In: Proc. of the ICSM. 2013.

[9] Zhou M, Mockus A. What make long term contributors: Willingness and opportunity in OSS community. In: Proc. of the ICSE. 2012.

[10] Jiang J, Feng FL, Lian XL, *et al*. Long-term active integrator prediction in the evaluation of code contributions. In: Proc. of the SEKE. 2016.

[11] Zanjani MB, Kagdi H, Bird C. Automatically recommending peer reviewers in modern code review. IEEE Trans. on Software Engineering, 2016,42(6):530−543.

[12] Gousios G, Pinzger M, van Deursen A. An exploratory study of the pull-based software development model. In: Proc. of the ICSE. 2014.

[13] Xia X, Lo D, Wang XY, *et al*. Who should review this change? Putting text and file location analyses together for more accurate recommendations. In: Proc. of the ICSME. 2015.

[14] Thongtanunam P, Tantithamthavorn C, Kula RG, *et al*. Who should review my code? A file location- based code-reviewer recommendation approach for modern code review. In: Proc. of the SANER. 2015.

[15] Yu Y, Wang HM, Yin G, *et al*. Reviewer recommender of pull-requests in GitHub. In: Proc. of the ICSME. 2014.

[16] Yu Y, Wang HM, Yin G, *et al*. Who should review this pull-request: Reviewer recommendation to expedite crowd collaboration. In: Proc. of the APSEC. 2014.

[17] Yu Y, Wang HM, Yin G, *et al*. Reviewer recommendation for pull-requests in GitHub: What can we learn from code review and bug assignment. Information and Software Technology, 2016,74:204−218.

[18] Lu S, Yang D, Hu J, *et al*. Code reviewer recommendation based on time and impact factor for pull request in github. Computer Systems & Applications, 2016,25(12):155−161 (in Chinese with English abstract).

[19] Weibgerber P, Neu D, Diehl S. Small Patches Get In! In: Proc. of the MSR. 2008.

[20] Bosu A, Carver JC. Impact of developer reputation on code review outcomes in OSS projects: An empirical investigation. In: Proc. of the ESEM. 2014.

[21] Gousios G, Pinzger M, van Deursen A. An exploratory study of the pull-based software development model. In: Proc. of the ICSE. 2014.

[22] Tsay J, Dabbish L, Herbsleb J. Influence of social and technical factors for evaluating contribution in GitHub. In: Proc. of the ICSE. 2014.

[23] Thongtanunam P, McIntosh S, Hassan AE, *et al*. Investigating code review practices in defective files: An empirical study of the QT system. In: Proc. of the 12th Int'l Working Conf. on Mining Software Repositories. 2015.

[24] Ray B, Posnett D, Filkov V. A large scale study of programming languages and code quality in GitHub. In: Proc. of the 22nd ACM SIGSOFT Int'l Symp. on Foundations of Software Engineering. 2014.

[25] Bosu A. Characteristics of the vulnerable code changes identified through peer code review. In: Proc. of the ICSE Companion 2014, 2014. 736−738.

[26] Ratner B. The correlation coefficient: Its values range between +1/−1, or do they? Journal of Targeting, Measurement and Analysis for Marketing, 2009,17(2):139−142.

附中文参考文献:

[18] 卢松,杨达,胡军,张潇.基于时间和影响力因子的 Github Pull Request 评审人推荐.计算机系统应用,2016,25(12):155−161.

蒋竞(1985－),女,博士,副教授,CCF 专业会员,主要研究领域为经验软件工程,开源软件,基于数据的分析与推荐.

张莉(1968－),女,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为软件建模与分析,需求工程,经验研究工程,软件体系结构.

吴秋迪(1996－),男,硕士生,主要研究领域为经验软件工程,开源软件.