软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.005222]

针对 GitHub 开源软件开发过程中影响因素的相关性分析*

杨 波[1,2,4+], 于 茜[1,2], 张 伟[3], 刘 超[4]

[1](北方工业大学 计算机学院,北京 100144)

[2](大规模流数据集成与分析技术北京市重点实验室,北京 100144)

[3](中国科学研究院 软件研究所,北京 100190)

[4](北京航空航天大学 计算学院,北京 100191)

Influence Factors Correlation Analysis in GitHub Open Source Software Development Process*

YANG Bo[1,2,4+], YU Qian[1,2], ZHANG Wei[3], LIU Chao[4]

[1](College of Computer Science, North China University of Technology, Beijing 100144, China)

[2](Beijing Key Laboratory on Integration and Analysis of Large-scale Stream Data, Beijing 100144, China)

[3](Institute of Software Chinese Academy of Sciences, Beijing 100190, China)

[4](School of Computer Science and Engineering, BeiHang University, Beijing 100191, China)

Abstract: More than 12 million open source software in GitHub so far. There are many existing studies about development process impact factors in GitHub open source software. However, it's lack of correlation analysis between Influence factors. Many impact factors are put forward (e.g., problem solve speed, problem increase speed) through analysis the GitHub open source software development process. This paper proposes an approach to analysis the correlation of between these factors. It's proving the existence of certain factors among some relevance after experiments. This paper also gives some suggestions on GitHub open source software development process according to the experiment.

Key words: GitHub; impact factors; correlation analysis; open source software; data mining

摘 要: 截至到目前为止,在 GitHub 开源软件托管平台上面的项目超过 1200 万,现有很多研究对 GitHub 开源软件的开发过程中的影响因素进行了分析,缺乏对影响因素间的相关性进行研究.本文通过分析 GitHub 开源软件的开发过程,提出了问题解决速度、问题增加速度等影响因素,并对这些影响因素间的相关性进行了分析.经过实验证明了有些影响因素之间存在一定的相关性.同时根据实验的结果还给出了针对 GitHub 开源软件开发过程的一些建议.

关键词: GitHub;影响因素;相关性分析;开源软件;数据挖掘

中图法分类号: TP301 **文献标识码**: A

开源软件指的是允许用户基于 OSI 列出的开源协议,在协议许可的范围内自由使用、修改软件源代码,且

* Supported by the National Natural Science Foundation of China under Grant No.6152011 (国家自然科学基金)
收稿时间: 2016-05-08; 修改时间: 2016-07-15; 采用时间: 2016-12-22; jos 在线出版时间: 2017-02-20
CNKI 网络优先出版: 2017-02-20 15:14:45, http://www.cnki.net/kcms/detail/11.2560.TP.20170220.1514.031.html

可以将软件源代码与其他软件代码进行结合使用的一种软件形式[1].开源软件的兴起与日益活跃有力地促进了软件以及相关技术的进步.开源软件发展至今已取得巨大成功,这些成功从对 WEB 服务器、移动操作系统和数据库等相关技术的发展中得到了验证[2].社交化开发平台 GitHub 是众多开源软件的社区之一,它不仅允许个人和组织创建和浏览代码库,而且还提供社区化软件开发的功能,包括允许用户关注其他用户、组织、仓库的动态,跟踪仓库代码的改动、Bug 和评论等,GitHub 同时也提供供仓库使用的 Wiki,以及使用 Git 进行项目协同开发[3].截止到目前为止,GitHub 上已拥有超过 1200 万个开源项目,且这个数目仍在不断增长[4].对开发过程中的影响因素及其相关性分析,在一定程度上可以揭示 GitHub 开源软件发展水平的高低与项目进展的好坏.

影响 GitHub 开发过程的因素很多,其中包括开发者能力水平的高低、开发者人数多少、开源软件的问题解决快慢和用户参与的激励等.这些因素中有一部分是商业行为,比如大型软件公司在背后的支持等,有一部分虽然可能是商业行为在开发过程中的间接体现,但却存在一些客观的数据,比如开发者人数多少、开源软件的问题解决快慢等.现有针对 GitHub 开发过程中的影响因素分析的研究将这些商业行为和客观的数据夹杂在一起考虑,这使得对影响 GitHub 开源软件项目开发过程的因素中的商业成分和非商业化成分难以进行区分,并让分析 GitHub 开源软件开发过程的影响因素之间的相关性具有不确定性.

本文从开源软件开发过程中产生的数据进行分析的角度出发,对影响 GitHub 开源软件开发过程的因素进行了分析,提出了问题(issue)解决速度、提交(commit)增加速度、问题(issue)增加速度、请求合并(merge)增加速度、问题评论(comment)速度、合并评论速度、解决问题总量和对代码变化速度等影响因素,并对这些因素之间存在的相关性进行了分析.第 1 节介绍针对 GitHub 开源软件的相关研究,其中重点介绍了影响 GitHub 开源软件开发过程的研究的相关方法.第 2 节给出了本文所提出影响因素的相关假设,并详细描述了影响 GitHub 开源软件开发过程中的影响因素.第 3 节详细阐述影响开源软件开发过程中的影响因素相关性分析方法.第 4 节利用给出的分析方法,进行实验并对实验结果进行详细分析.最后在第 5 节进行总结,并对后续工作进行展望.

1 相关研究及研究动机

分布式版本控制系统的出现,催生了开源软件并促进了其发展[4].开源软件与传统软件在很多方面存在不同[5,6]. 针对开源软件的各种特征,很多学者进行了研究.Mockus 等人研究了开发者是如何在大型开源软件开发中进行交互的[7].Krishanmurthy 对 Sourceforge.net 上前 100 个开源软件的分析,得出了开源软件开发与管理人员的组成情况[8].Hars 等人开源软件得以发展的内在动力进行了研究,得出的结论是内部动力包括利他主义、开源社区的身份认同感；外部动力是贡献者看好开源软件未来的发展以及个人需求[9].Lakhani 等人从贡献者所需的激励出发,指出开源软件开发过程中贡献者的激励来自与参与讨论的乐趣和对软件进行改进的需求[10].Sohn 等人利用传统软件质量评价体系 ISO/IEC9126,对开源软件的质量因素对用户使用的影响进行了分析,结果表明功能性、效率、共享性等质量因素对用户使用有显著的直接影响,软件移植性、可靠性和可维护性等质量因素对用户使用有间接影响[11].Subraniam 等人研究了开源软件的成功因素,指出开源软件的成功因素包括两类：非时变变量与时变变量,非时变包括编程所使用语言、采用的许可证类型等,时变变量包括软件项目的开发阶段和上一个时期的用户数等[12].Evangelos 等人调研了有些开源软件最终失败的原因,指出使用者的参与以及核心贡献者间的沟通好坏是开源软件成功与否的关键[13].由于 GitHub 的 pull-request 的特性,使得其开源软件的开发过程的研究呈现出自身的特征.Dabbish 等人发现 GitHub 的透明性使得贡献者更好的管理项目的开发[14].Peterson 等人发现开源软件的开发过程与传统软件的开发过程有很多相似之处,它们的不同之处在于开源软件开发过程中有更快的修改和更新次数[15].Gousios 等人对基于推送的软件开发模型进行调研,指出基于这样的开发模型(以 GitHub 为例)有很多优势,比如使得贡献者之间的沟通更加频繁等[16].Minghui Zhou 等人分析了新加入的开发人员成为长期贡献者的影响因素,他们从多个方面对开发人员作出的贡献进行了分析,实验结果显示,新加入的开发者在加入项目的第一个月,表现会非常活跃,且他们会更愿意对问题进行评论.另外,环境对新加入的开发者也影响很大,如果其所在的环境很活跃,凝聚力很强,开发者成为长期贡献者的概率会相应增加[17]. 等 Jason Casebo 等提出了基于作者熵（author entropy）的方法,用来量化开发人员对文件的贡献,

结果发现开发人员对同一文件的贡献越平均,则文件的作者熵越大.另外,结果显示当两位开发人员修改同一个文件时,大文件和小文件相比更倾向于拥有主导者.除此之外,Jason Casebo 等人还指出非主导者对文件的修改主要包括如下几种情况：接口修改、缺陷修复、代码格式的修改即输出信息的修改[18].

从上述研究可以看出,许多研究者都关注到开源软件开发过程中的影响因素,但却忽略了开源软件开发过程中影响因素的相关性分析. 而影响因素间的相关性分析可以帮助开源软件的贡献者更好的参与到开发与维护中去,同时能够在一定程度上提高开源软件开发的效率与质量.比如是否可以通过调节开源软件开发过程中的某个或某几个影响因素,来达到加快开源软件开发过程中问题解决的速度？类似这样的问题成为本文研究的主要动机：为了让开源软件的开发过程更快更好地进行,我们可以调节哪些影响因素,怎样去调整？鉴于此,本文特提出如下研究问题：

1. 影响 GitHub 开源软件开发过程中的主要因素之间是否存在相关性？如果存在,其相关性是怎样的？
2. 如何利用 GitHub 开源软件开发过程中影响因素的相关性,来更好地指导开源软件的开发？

针对上述研究问题,本文在接下来的内容中对此进行详细的描述.

2 GitHub 开源软件开发过程的影响因素

GitHub 开源软件开发过程中会产生大量的数据,这些数据能在一定程度上反映出影响开发过程的因素.为了获得这些数据并进行挖掘,需要分析 GitHub 开源软件的开发过程.一般而言,GitHub 开源软件的开发过程包含若干个工作流程,一个典型的 GitHub 工作流如图 1 所示,其中包括讨论问题、指定事务、执行事务、审查事务和问题解决后迭代五个步骤.讨论问题步骤中主要是许多开发人员就一个新的功能或需要修改的功能进行沟通,并最终确定要增加或修改什么样的功能；某次讨论结束后,为专门为需要增加或修改的功能创建一个事务；接下来是实现或修改该功能；功能完成之后会推送到远程进行审查；审查通过后会将代码合并到主分支.对 GitHub 开源软件开发过程中的工作流进行分析,可以找到影响 GitHub 开源软件开发过程带来影响的因素.接下来给出 GitHub 工作流的形式化定义.

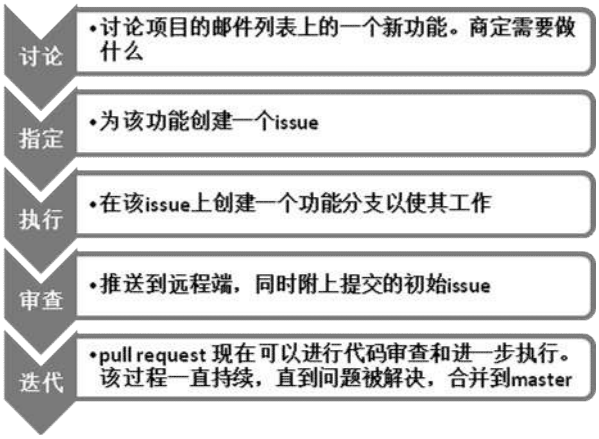


Fig.1 Workflow of GitHub
图 1 GitHub 工作流

定义 1：GitHub 工作流(GHW)

GitHub 工作流可以表示为一个四元组 GHW={S,P,M,N},其中 S 表示提交,可以表示为三元组 S={s_num,comment_commit,author},其中 s_num 表示提交次数, comment_commi 表示的是对提交的评论, author 表示的是提交的作者.

P 表示讨论的问题,可以表示为三元组 P={des,comment_problem,sta},其中 des 表示问题的描述, comment_problem 表示对问题的评论, sta 表示的是问题的状态.

M 表示请求合并,可以表示为四元组 M={ add_merge,del_merge,comment_merge, merge_id},其中 add_merge

表示合并时增加的代码行数, del_s 表示合并时删除的代码行数, comment_merge 表示的是对合并的评论, merge_id 表示的是请求合并的编号.

N 表示贡献者的人数.针对 S,P,M 和 N 的进一步分析,可以细化 GitHub 工作流中所包含的内容,接下来给出 S,P,M 和 N 中相关概念的定义.

定义 2：提交总量(S_ALL)

提交总量 S_ALL 指的是仓库历史提交次数之和.S_ALL 可以由如下公式计算：

$$S\_ALL = \sum_{i=0}^{n} s\_num_i \tag{1}$$

i 表示的是序号, s_num_i 表示的是第 i 次提交的次数, n 表示的是最后一次提交的序号.

定义 3：问题总量(P_ALL)

问题总量 P_ALL 指的是仓库历史问题个数之和.P_ALL 可以由如下公式计算：

$$P\_ALL = \sum_{i=0}^{n} p\_num_i \tag{2}$$

i 表示的是序号, p_num_i 表示的是第 i 次问题个数, n 表示的是最后一次问题的序号.

定义 4：解决问题总量(P_SOL_ALL)

解决问题总量 P_SOL_ALL 指的是在某个时间点,解决仓库历史问题个数之和. P_SOL_ALL 可以由如下公式计算：

$$P\_SOL\_ALL = \sum_{i=0}^{k} p\_sol\_num_i \tag{3}$$

i 表示的是序号, $\sum_{i=0}^{k} p\_sol\_num_i$ 表示的是第 i 次解决问题个数, k 表示的是最后一次解决问题的序号.

定义 5：问题的评论速度(P_COM_VEL)

问题评论速度 P_COM_VEL 指的是仓库在某个时间段（记为 d ）内,平均每天问题评论的个数.P_COM_VEL 可以由如下公式计算：

$$P\_COM\_VEL = \frac{comment\_p\_all}{d} \tag{4}$$

定义 6：问题解决速度(P_SOL_VEL)

问题解决速度 P_SOL_VEL 指的是仓库在某个时间段（记为 d ）内,平均每 d 天解决的问题个数.P_SOL_VEL 可以由如下公式计算：

$$P\_SOL\_VEL = \frac{p\_sol\_all}{d} \tag{5}$$

p_sol_all 表示的是解决问题总量, d 表示的是时间段内包含的天数.

定义 7：请求合并增加速度(M_ADD_VEL)

请求合并增加速度 M_ADD_VEL 指的是仓库在某个时间段（记为 d ）内,平均每天增加的请求合并次数.M_ADD_VEL 可以由如下公式计算：

$$M\_ADD\_VEL = \frac{m\_add\_all}{d} \tag{6}$$

m_add_all 表示的是在时间段内总的请求合并的总次数, d 表示的是时间段内包含的天数.

定义 8：合并速度(M_VEL)

合并速度 M_VEL 指的是仓库在某个时间段（记为 d ）内,平均每天成功合并的次数.M_VEL 可以由如下公式计算：

$$M\_VEL = \frac{m\_sus\_all}{d} \tag{7}$$

m_sus_all 表示的是在时间段内总的成功合并的总次数, d 表示的是时间段内包含的天数.

定义 9：贡献者人数(N)

贡献者人数 N 指的是仓库历史上做出贡献者的总人数.N 可以由如下公式计算：

$$N = \sum_{j=0}^{n} a_j \tag{8}$$

j 表示的是仓库历史中每一天的序号,从 0 开始, a_j 表示的是第 j 天成为贡献者的人数, n 表示的是统计的仓库历史中的最后一天.

定义 10：代码变化速度(C_VEL)

代码变化速度 C_VEL 指的是仓库在某个时间段（记为 d ）内,平均每天变化的代码行数.C_VEL 可以由如下公式计算：

$$C\_VEL = \frac{c\_all}{d} \tag{9}$$

c_all 表示的是代码总的变化量, d 表示的是时间段内包含的天数.

3 GitHub 开源软件开发过程中影响因素相关性分析

3.1 影响因素相关性分析框架

GitHub 开源软件开发过程影响因素的相关性分析框架图如图 2 所示,从图 2 可以看出,影响因素相关性分析主要包括数据收集、采集影响数据、影响因素相关趋势分析和影响因素相关性分析四个主要的步骤.面对数量庞大的 GitHub 开源软件仓库,首先需要对这些仓库进行查询,然后对符合要求的仓库按照时间进行数据的抓取, GitHub 开源软件开发过程中的数据采集有多种方法,比如网络爬虫和 GitHub API 等,本文采用的是调用 GitHub API 的方式来进行数据采集,该方法通过拼接 URL,发送 http 请求,返回 HTTP 响应内容,并对正文中的 Json 格式数据进行解析,然后显示结果或存入本地.

在抓取到数据后,还需要对数据进行预处理；经过预处理后的数据会保存到本地,成为影响分析的初始数据；根据此前所提到的影响因素,对其中的数据进行采集,能够得到影响因素数据；对影响因素数据进行相关

趋势分析,可以得到相应的趋势图.从趋势图中可以看到影响因素之间存在的大致的相关性.

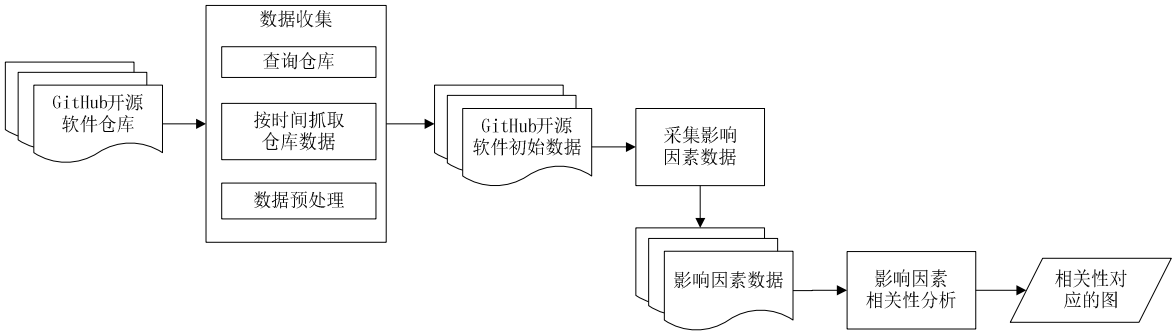


Fig.2 Impact factors correlative analyze framework
图 2 影响因素相关性分析框架

3.2 影响因素之间的相关性分析

GitHub 开源软件的开发过程的影响因素有很多,考虑到影响因素之间相关性的复杂性,且由于影响因素之间可能存在线性相关或非线性相关,因此仅考虑两种影响因素之间的相关性以及相关性的趋势图. 所考虑相关性的两种影响因素如下：

（1）贡献者人数 N 与问题解决速度 P_SOL_VEL

在传统的软件开发中,贡献者人数越多,问题解决的速度一般越快.这样类似的结论是否在 GitHub 开源软件的开发过程中是否也存在？对贡献者人数 N 对问题解决速度 P_SOL_VEL 的影响的分析有助于回答这个问题.

在对贡献者人数 N 对问题解决速度 P_SOL_VEL 的影响进行相关性分析时,以贡献者人数 N 为横坐标,问题解决速度 P_SOL_VEL 为纵坐标.将得到的影响因素数据中的贡献者人数 N 和问题解决速度 P_SOL_VEL 在坐标轴上进行映射,看两者是否存在相关性.

（2）问题增加速度 P_ADD_VEL 与请求合并增加速度 M_ADD_VEL

通过考虑问题增加速度 P_ADD_VEL 对请求合并增加速度 M_ADD_VEL 之间相关性,可以看出是否问题增加的频率越快,是否会导致请求合并增加的速度越快.

在对问题增加速度 P_ADD_VEL 与请求合并增加速度 M_ADD_VEL 进行相关性分析时,以问题增加速度 P_ADD_VEL 为横坐标, 请求合并增加速度 M_ADD_VEL 为纵坐标.将得到的影响因素数据中在坐标轴上进行映射,看两者是否存在相关性.

类似地,本文还考虑以下影响因素之间的相关性：

（3）问题评论速度 P_COM_VEL 对问题解决速度 P_SOL_VEL

（4）解决问题总量 P_SOL_VEL 与代码变化速度 C_VEL

另外,本文还提出了影响因素的相关性评价指标,用来衡量因素之间存在的相关程度.

4 实验

4.1 实验目的

针对此前提出的影响开源软件开发过程中影响因素的相关性分析三个研究问题,实验目的可以总结为：验证 GitHub 开源软件开发过程中影响因素存在相关性,且能够用图表进行直观表征；在实验目的达到的前提下,能够给出利用 GitHub 开源软件开发过程中的影响因素的相关性,给出指导开源软件更好地开发的一些建议.

4.2 实验工具与数据集

为了方便实验,实现了一个 GitHub 开源软件开发过程数据收集与分析工具 Ghda.本实验选取了 1000 个开源软件[20],这些开源软件大小规模有 2000KB 的,也有多达 640138KB 的大型程序,收集这些开源软件的数据的时间间隔最短为 1 年,最长的接近 6 年.表 1 是部分开源软件的列表:

Table 1 List of open source softwares

表 1 开源软件列表

软件名称	简介	大小	收藏	关注	拷贝	创建日期	采集截至日期
AFNetworking	IOS 框架	640138	17913	1407	5299	2011-06-01	2015-06-10
bottle	Web 框架	9373	2962	219	598	2009-07-01	2015-05-27
GPUImage	图像处理库	52907	8858	362	2154	2012-02-13	2015-05-27
pop	动画引擎	2836	11455	872	1617	2014-03-31	2015-06-10
Mantle	建模框架	64338	6762	350	795	2012-09-05	2015-05-27
paramiko	Python 模块	8445	1796	166	525	2009-02-02	2015-05-27
ActionBarSherlock	ActionBar 库	18403	6869	845	3989	2011-03-08	2014-12-20
cancan	Ruby 类库	8361	5858	182	809	2009-11-17	2013-09-06

表分为 8 列,其中软件名称指的是所使用开源软件的名称描述;简介指的是该开源软件所具备的主要功能描述;大小指的是开源软件的规模,以 KB 为单位;收藏指的是 GitHub 上获得称赞(star)的次数;关注指的是 GitHub 上被关注的(watch)次数;拷贝表示 GitHub 上 fork 的次数;创建日期表示仓库在 GitHub 上的创建日期;采集截至日期表示收集该仓库在 GitHub 上的某一个时间节点.

4.3 评价指标

1.影响因素的相关性(Impact Factors Correlation, IFC)

影响因素的相关性 *IFC* 用来衡量因素之间存在的相关程度,其值可以用可以由如下公式计算:

$$IFC = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{n\sum_{i=1}^{n}(x_i-\bar{x})^2.\sum_{i=1}^{n}(y_i-\bar{y})^2}} \tag{10}$$

IFC 是按积差方法计算,以两因素与各自平均值的离差为基础,通过两个离差相乘来反映两因素间的相关程度. *IFC* 的值越大,表示两因素间的相关程度越高;反之则表示两因素间的相关程度越低. $IFC>0$ 表示两因素之间存在正相关, $IFC<0$ 表示两因素之间存在负相关, $IFC=0$ 表示两因素之间不存在线性相关, $IFC=1$ 表示两因素之间存在完全线性相关.

2. 积极影响(Positive Impact, PI)

影响因素的相关性 *IFC* 虽然能够衡量因素间的相关程度,但因素间是否存在积极影响却还需要一定阈值来限定,一般来说,当满足不等式 $0.5<IFC<=1$ 时,可以认为两因素间存在积极影响.

3. 消极影响(Negative Impact, NI)

与积极影响类似,因素间是否存在消极影响也需要一定阈值来限定,一般来说,当满足不等式 $-1<IFC<-0.5$ 时,可以认为两因素间存在消极影响.

4. 难以确定影响(Uncertain Impact, UI)

当满足不等式 $-0.5\le IFC<=0.5$ 时,可以认为两因素间难以确定是否存在影响.

4.4 实验结果及分析

实验 1:典型项目的影响因素间的相关性关系分析

为了初步看出本文所提的几个影响因素间的相关性,特对上述 8 个项目的相关性进行了直观的图形展示,

分别如图 3、图 4、图 5 和图 6 所示,需要注意的是,本文所提到的速度中所取的时间间隔 d 为周.

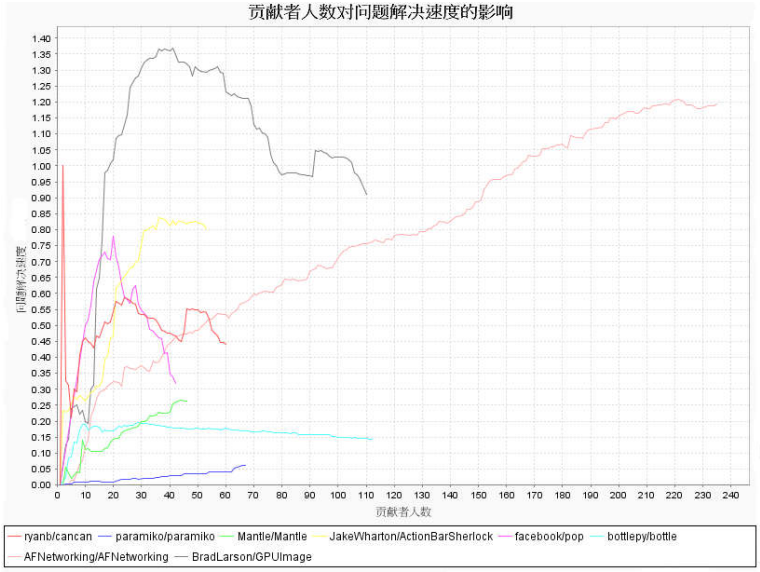


Fig.3 The influence of the number of contributors on the speed of problem solving

图 3 贡献者人数对问题解决速度的影响

贡献者人数对问题解决速度之间的相关性如图 3 所示,从图 3 可以看出,所有仓库对应的曲线的斜率都是先逐渐增加,然后又逐渐下降,部分仓库还出现了波峰与波谷.这表明在开发过程的前期,贡献者人数越多,问题解决速度越快;而到了一定阶段,贡献者人数的多少与问题解决速度之间不存在必然的关系.在这个阶段可能也是 GitHub 开源软件逐渐趋于成熟.

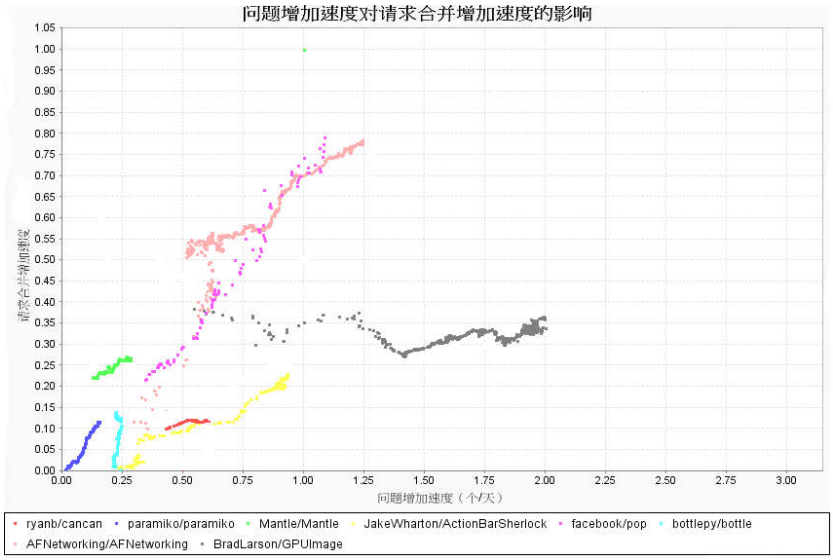


Fig.4 The influence of problem addition speed and asking merger speed

图 4 问题增加速度对请求合并速度的影响

问题增加速度与请求合并速度之间的相关性如图 4 所示,从图 4 可以看出,除了仓库 GPUImange 的问题增加速度增大时其请求合并速度略微有些下降之外,其他的 7 个仓库都存在问题增加速度越快,其请求合并速度也越快的现象.

问题评论速度与问题解决速度之间的相关性如图 5 所示,从图 5 可以看出,所有的 8 个仓库都存在问题评论速度越快,其问题解决速度也越快的现象.

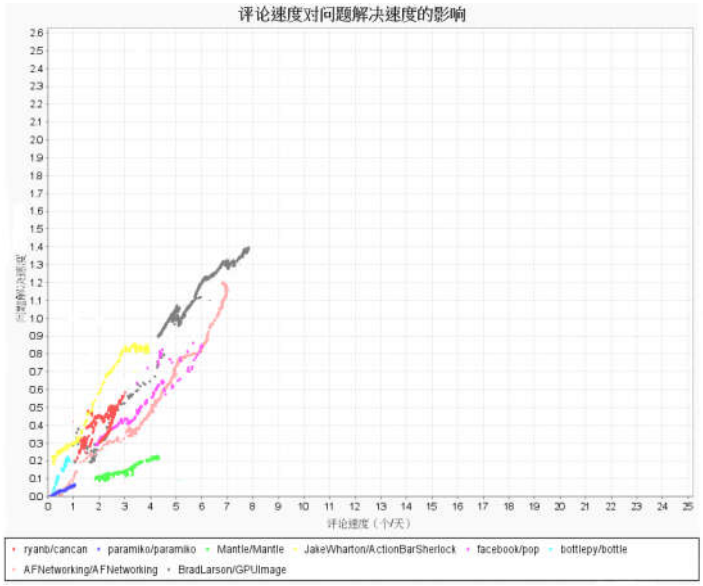


Fig.5 The influence of problem comment speed and problem resolve speed

图 5 评论速度对问题解决速度的影响

解决问题总量与代码变化速度之间的相关性如图 6 所示,从图 6 可以看出,所有的 8 个仓库都存在随着解决问题总量的增加,代码变化速度越慢的情况.

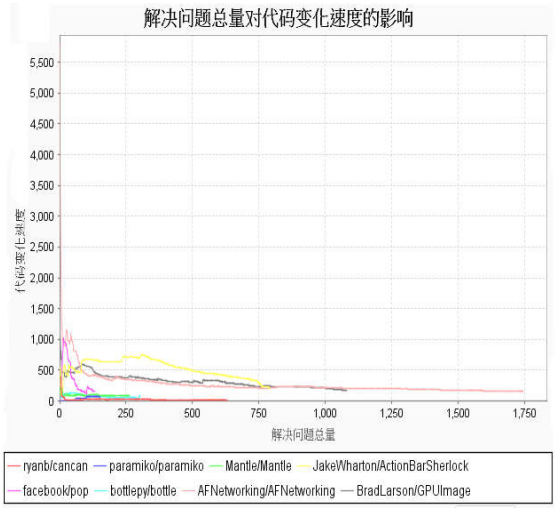


Fig.6 The influence of problem resolve number and code change speed

图 6 解决问题总量对代码变化速度的影响

为了进一步了解这些因素之间存在的相关性,本文还利用影响因素的相关性 *IFC* 用来衡量这些因素之间存在的相关程度.

实验 2：贡献者人数 N 与问题解决速度 P_SOL_VEL

实验的结果如图 7 所示,从图 7 可以看出,在所选的 1000 个项目中,贡献者人数与问题解决速度之间存在积极影响的项目个数接近 800 个,而存在消极影响的项目个数也超过了 200 个,难以确定影响的项目只有 3 个.

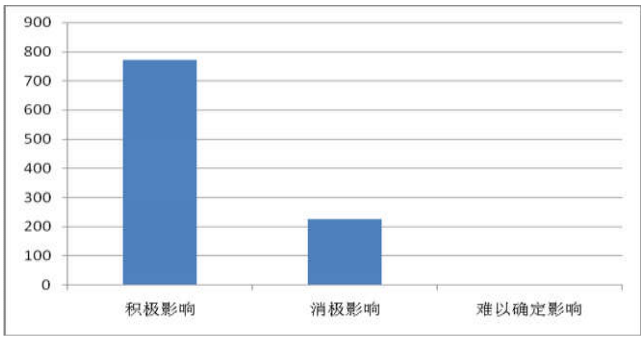


Fig.7 the relationship between the number of contributors and problem resolve speed

图 7 贡献者人数与问题解决速度之间的关系图

为了进一步分析贡献者人数与问题解决速度之间的相关性,对这 1000 个项目进行了第二次分析,分别从人数增长初期（25 人）,人数增长中期（100 人）和人数增长后期（200 人）来进行分析,得出的结果如图 8 所示. 从图 8 可以看出, 在人数增长初期, 贡献者人数与问题解决速度存在明显的积极影响；而随着项目开发过程往前推进,超过 200 个项目的贡献者人数与问题解决速度存在消极影响；到了人数增长后期,贡献者人数与问题解决速度之间的影响因素存在更多的不确定性.

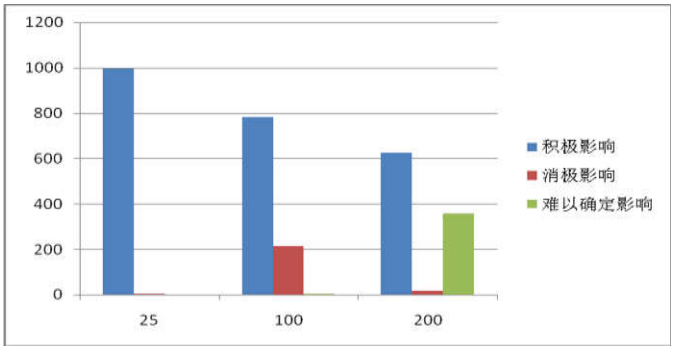


Fig.8 the relationship between contributor and problem resolve speed with contributor number

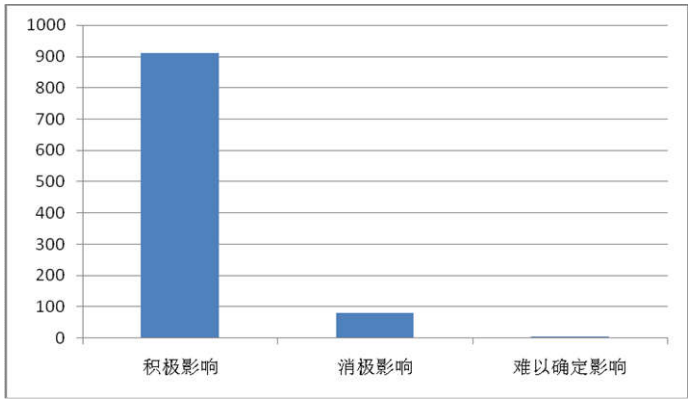
图 8 不同人数的贡献者人数与问题解决速度之间的关系图

根据上面的实验,再结合实际的 GitHub 的开发过程来对实验 2 的分析,从中可以看出,在项目开发的初期,贡献者人数较少,但问题解决的相对较快.随着项目的进行,虽然贡献者人数有所增加,但问题解决速度却并没有增加,而到了贡献者人数达到一定的规模,其问题解决速度却不一定会明显比此前快.原因可能是初期贡献者热情很高,项目也有很多有待完善的地方,随着项目逐渐成熟,贡献者热情有所降低,使得问题解决速度相对下降.

实验 3：问题增加速度 P_ADD_VEL 与请求合并增加速度 M_ADD_VEL

实验结果如图 9 所示,从图 9 可以看出,1000 个项目中超过 900 个项目的问题增加速度与请求合并速度存在积极影响,少于 100 个项目中的问题增加速度与请求合并速度存在消极影响,只有 6 个项目中的问题增加速度与请求合并速度之间的相关性难以确定.

根据上面的实验,再结合实际的 GitHub 的开发过程来对实验 3 的分析,从中可以看出,在项目开发的过程中,当问题增加的速度增加时,可能会导致参与开发的程序员积极性变高,从而使得请求合并增加速度加快.但也需要注意的是,这种现象不是出现在所有的项目中,在实验 3 中,大约有 9%的项目出现问题增加速度与请求合并速度存在消极影响或相关性难以确定.后续对这些项目做进一步的分析,发现其中大部分项目的开发人员较少,且问题增加相对缓慢,而请求合并增加数量也不多,因此导致这两者之间没有必然的相关性.



. **Fig.9** the relationship between problem addition speed and asking merger speed

图 9 问题增加速度与请求合并增加速度的关系图

实验 4：问题评论速度 P_COM_VEL 对问题解决速度 P_SOL_VEL

实验结果如图 10 所示,从图 10 可以看出, 1000 个项目中超过 990 个项目的问题评论速度与问题解决速度存在积极影响,少于 10 个项目的问题评论速度与问题解决速度存在消极影响,只有 2 个项目中的问题评论速度与问题解决速度之间的相关性难以确定.

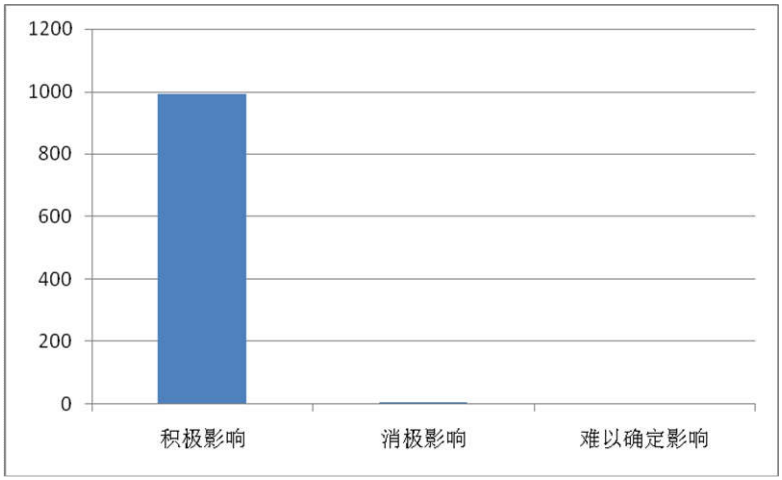


Fig.10 the relationship between problem comment speed and problem resolve speed

图 10 问题评论速度与问题解决速度的关系图

根据上面的实验,再结合实际的 GitHub 的开发过程来对实验 4 的分析,从中可以看出,在项目开发的过程中,当问题评论的速度增加时,表明参与开发的程序员对问题的关注度变高,从而在一定程度上使得问题解决速度加快.而对于还有存在消极影响或难以确定影响的 10 多个项目,本文也做了进一步的调研和分析,结果发现这些项目也存在开发人员数目较少,且代码更新的速度也相当慢,从而使得问题评论速度与问题解决速度之间没有必然的积极影响.

实验 5：解决问题总量 P_SOL_ALL 与代码变化速度 C_VEL

实验结果如图 11 所示,从图 11 可以看出, 1000 个项目中超过 980 个项目的解决问题总量与代码变化速度存在消极影响,少于 20 个项目的问题评论速度与问题解决速度存在积极影响,只有 5 个项目中的问题评论速度与问题解决速度之间的相关性难以确定.

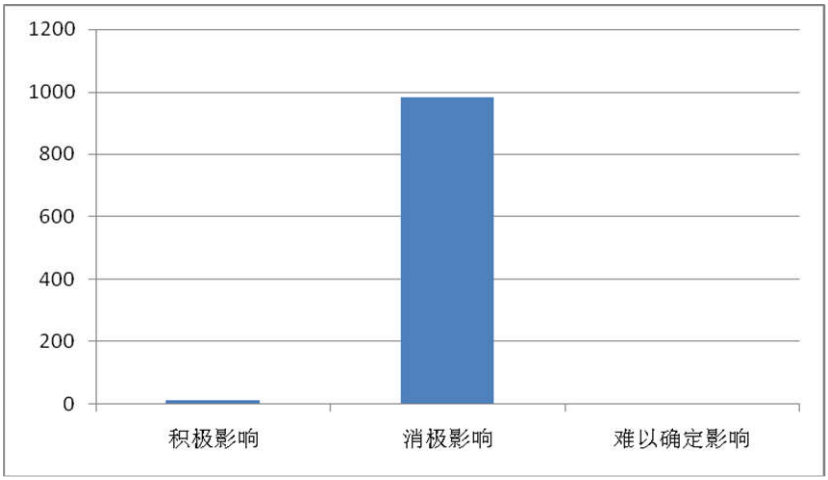


figure 8　the relationship between problem resolve number and code change speed

图 8 解决问题总量与代码变化速度的关系图

根据上面的实验,再结合实际的 GitHub 的开发过程来对实验 5 的分析,从中可以看出,在项目开发的过程中,随着解决问题总量的增加,代码越来越趋于稳定,代码变化的速度也相应变慢.而对于还有存在消极影响或难以确定影响的 20 多个项目,本文也做了进一步的调研和分析,结果发现这些项目也存在开发人员数目较少,且有些项目的代码基本上很长时间不再更新,从而使得解决问题总量与代码变化速度之间没有必然的相关性.

4.5 对研究问题的回答及对GitHub开源软件开发过程的几条建议

本文在此前提出了两个研究问题,在基于上述 5 个实验的基础上,这两个问题也有了相应的答案.影响 GitHub 开源软件开发过程中的主要因素之间有些确实存在一定的相关性,比如本文所提出的贡献者人数 N 与问题解决速度 P_SOL_VEL、问题增加速度 P_ADD_VEL 与请求合并增加速度 M_ADD_VEL、问题评论速度 P_COM_VEL 与问题解决速度 P_SOL_VEL 之间就存在一定的积极影响,解决问题总量 P_SOL_VEL 与代码变化速度 C_VEL 则存在一定的消极影响.

利用这些结论,可以对基于 GitHub 开源软件的开发过程给出几条建议,这些建议可以给开源项目的管理者或是参与项目开发的程序员.

1. 在 GitHub 开源软件开发初期,适量地吸引更多的贡献者来参与.这样做的好处是可以使得问题解决的速度在一定程度上加快,从而可能提高开发的效率.
2. 尽量提高提交问题的频率,这样或有助于请求合并增加的速度.这样做的好处是随着提交的问题的频率变快,请求合并的速度也有可能会加快,从而可能提高开发的效率.
3. 尽量提高问题评论的频率,这样或有助于问题尽快解决.这样做的好处是随着对问题评论的频率变快,问题解决的时间间隔有可能会减少,从而可能提高开发的效率.
4. 尽量多解决问题,这样会尽早使得代码趋于稳定.这样做的好处是随着对解决问题的数量越来越多,代码中存在的问题可能会越来越少,从而让开发过程变得更有效率.

另外，本文也给出了一些实施的建议：

1. 对于吸引更多的贡献者来参与开发的问题，可以采取宣传、激励、寻求企业的支持等方法来吸引更多的贡献值参与，并能经常进行积极地沟通与协作，使贡献者之间能够很和谐地进行工作。对于经验非常丰富的贡献者，应该采取一定的奖励机制，使得项目能够更好更快地进行。
2. 有针对性地提高问题提交问题的频率,并且为贡献者创造不错的问题评论的氛围，让大家畅所欲言且能够集中焦点，这需要从管理学和心理学等多个角度来进行处理。
3. 对于解决问题的快慢的问题，可以研究开源软件尤其是基于 GitHub 的软件问题分配的各种策略，采

用更好的分配策略，将问题分配给更合适的贡献者进行修改。

4.6 威胁实验有效性因素

GitHub 开源软件开发过程中影响因素的相关性分析虽然得到了多条很有意思的结论,但考虑到所分析的开源软件数目还不够多,且没有考虑更多的影响因素,比如贡献者的水平、评论中是否包含情感等因素,因此,后续还需要进行更多的实验和考虑更多的影响因素来做进一步的验证.另外,本文只考虑两种影响因素之间的相关性,缺乏对多种影响因素之间相关性的考察,且目前的分析还没有进一步定量化,后续会在这些方面进行更深的研究.除此之外,本文所选择的实验程序还仅仅只是 1000 个开源程序,与 GitHub 超过 1200 万的项目相比依然是一个非常小的数目,因此也有可能存在一定的偶然性.

5 总结与展望

本文分析了 GitHub 开源软件开发过程中的影响因素,提出了问题解决速度、提交增加速度、问题增加速度、请求合并增加速度、问题评论速度、合并评论速度、解决问题总量和对代码变化速度等影响因素,并对这些因素之间存在的相关性进行了分析.通过对典型的 GitHub 开源软件进行实验,得出了一些对 GitHub 开源软件开发过程非常积极的结论.由于 GitHub 开源软件开发过程非常复杂,本文肯定还存在一些需要改进的地方.在后续的研究中,会考虑更多的影响因素和多种影响因素之间的相关性.

致谢 衷心感谢北京航空航天大学软件工程研究所开源软件库挖掘小组的成员在此前所做的工作,感谢在百忙之中对本文进行评阅的各位专家.

References:

[1] Chris DiBona,Sam Ockman,Mark Stone. Open Sources: Voices from the Open Source Revolution. O'Reilly Open Source,1999.

[2] Sen R,Singh S S,Borle S. Open source software success:Measure and analysis. Decision Support Systems, 2012,52(2):364-372.

[3] Scott Chacon. Pro Git[M]. Berkeley,CA,USA：Apress, 2009. 1-288

[4] C. Bird, P. C. Rigby, E. T. Barr, D. J. Hamilton, D. M. German, P. Devanbu. The promises and perils of mining Git. In MSR ’09: Proceedings of the 6th IEEE Intl. Working Conference on Mining Software Repositories, 2009,37(5):1-10.

[5] Minghui Zhou. Looking for micro-process in large-scale data. In Proceedings of the 2nd international workshop on Evidential assessment of software technologies (EAST '12). ACM, New York, NY, USA, 2012. 39-42. DOI=10.1145/2372233.2372245.

[6] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, Daniela Damian. The promises and perils of mining GitHub. In Proceedings of the 11th Working Conference on Mining Software Repositories (MSR 2014). ACM, New York, NY, USA, 2014. 92-101.

[7] A. Mockus, R. T. Fielding, J. D. Herbsleb. Two case studies of open source software development: Apache and Mozilla. ACM Trans. Softw. Eng. Methodol, 2002,11(3):309–346.

[8] Sandeep Krishnamurthy. Cave or Community?: An Empirical Examination of 100 Mature Open Source Projects. First Monday, 2002,7(6).

[9] A Hars, S Ou. Working for Free? Motivations of Participating in Open Source Projects. In HICSS '01: Proceedings of the 34th Annual Hawaii International Conference on System Sciences (HICSS-34),2001.

[10] Joseph Feller, Brian Fitzgerald, Scott A. Hissam, Karim R. huff. Why Hackers Do What They Do:Understanding Motivation and Effort in Free/Open Source Software Projects, 2003. 3-21.

[11] S.Sohn, M.Mok. A strategic analysis for successful open source software utilization based on a structural equation model[J]. The Journal of Systems and Software,2008,81(6):1014-1024.

[12] Subrama niam, R.Sen, M.Nelson. Determinants of open source software project success: A longitudinal study[J]. Decision Support Systems,2009,46(2):576-585.

[13] Evangelos Katsamakas, Nicholas Georgantzas. Why Most Open Source Development Projects Do Not Succeed?. In Proceedings of the First International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS '07). IEEE Computer Society, Washington, DC, USA,2007. 3-3.DOI=10.1109/FLOSS.2007.15 http://dx.doi.org/10.1109/FLOSS.2007.15

[14] L. Dabbish, C. Stuart, J. Tsay, J. Herbsleb. Leveraging transparency. IEEE Software,2013,30(1):37-43.

[15] K. Peterson. The github open source development process. Technical report, Mayo Clinic, May 2013.

[16] Georgios Gousios, Martin Pinzger, Arie van Deursen. An exploratory study of the pull-based software development model. In Proceedings of the 36th International Conference on Software Engineering (ICSE 2014). ACM, New York, NY, USA,2014. 345-355. DOI=10.1145/2568225.2568260 http://doi.acm.org/10.1145/2568225.2568260

[17] M. Zhou, A. Mockus. Who Will Stay in the FLOSS Community? Modeling Participant's Initial Behavior. in IEEE Transactions on Software Engineering,2015,41(1):82-99.

[18] Jason R. Casebolt, Jonathan L. Krein, Alexander C. Maclean, Daniel P. Delorey. Author Entropy vs. File Size in the GNOME Suite of Applications. 6th IEEE International Working Conference on Mining Software Repositories,2009. 91-94.

[19] https://github.com.