软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, [doi: 10.13328/j.cnki.jos.000000]

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

代码文件贡献组成模式的分析*

谭 鑫[1], 林泽燕[1], 张宇霞[1], 周明辉[1]

[1](北京大学 信息科学技术学院,北京 100871)

通讯作者: 周明辉, E-mail: zhmh@pku.edu.cn

摘 要: 软件开发过程中,同一代码文件经常由多名开发者共同开发和维护,各个开发者向文件贡献了不同的代码量,使之形成特有的贡献组成.代码文件的贡献组成是否合理直接影响开发者的任务分配,进而影响软件质量和开发效率.对于不同类型的代码文件,如何刻画并确定其合理的贡献组成模式,成为一个亟待解决的问题.首先,本文基于代码所有权,从贡献组成的集中度、复杂度和稳定性三个维度出发,提出刻画贡献组成的三个量度.其次,本文以 OpenStack 的核心项目 Nova 为研究案例,在其版本控制数据上建立贡献组成的量度,总结了 12 种通用文件类型,归纳出 3 种贡献组成模式.最后,本文结合邮件以及面对面访谈的方式,验证了量度的有效性以及贡献组成模式的合理性,并从贡献组成的角度,对软件开发过程给出了一些指导性建议.

关键词: 贡献组成;量度;代码所有权;数据分析;软件质量

中图法分类号: TP311

中文引用格式: 谭鑫,林泽燕,张宇霞,周明辉.基于量度的代码文件贡献组成模式的分析.软件学报. http://www.jos.org.cn/1000-9825/ 0000.htm

英文引用格式: Tan X, Lin ZY, Zhang YX, Zhou MH. The Analysis of Code Files' Contribution Composition Pattern. Ruan Jian Xue Bao/Journal of Software, 2017 (in Chinese). http://www.jos.org.cn/1000-9825/0000.htm

The Analysis of Code Files' Contribution Composition Pattern

TAN Xin[1] LIN Ze-Yan[1] ZHANG Yu-Xia[1] ZHOU Ming-Hui[1]

[1](School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

Abstract: In the process of software development, the one code file is often developed and maintained by more than one developer, and each developer contributes different amount of code to the file, which forms a unique contribution composition. Whether the contribution of the code file is reasonable or not directly affects the task allocation, and then affects the quality of software and development efficiency. For different types of code files, how to measure and determine the contribution composition of the code file becomes an urgent problem to be solved. Firstly, based on the code ownership, from the three dimensions of the contribution composition: concentration, complexity and stability, this paper establishes a set of metrics to describe the contribution composition of the code file. Secondly, we choose Nova (one of the OpenStack' core projects) as a case study. Based on its' version control data and the metrics put forward, a measure of contribution composition is established, and 12 common file types are summarized, showing 3 contribution composition patterns. In the end, we verify the validity of metrics and the rationality of contribution composition patterns by combining mail and face-to-face interviews, and gives some instructive suggestions for software development process.

Key words: contribution composition; metric; code ownership; data analysis; software quality

由于人们对于软件的功能需求日益丰富,软件的规模越来越大,结构越来越复杂.许多大规模复杂软件,例如 OpenStack, Linux Kernel 等都是由分布在全球各地的成千上万的开发者协同开发的.同一模块甚至同一个代码文件也往往是由多名开发者同时开发进行[1,2].开发者参与需求分析、架构设计及具体开发实现等,他们对代码文件的理解,包括功能定义、架构设计思想、代码主体逻辑等,形成了其关于该代码文件特定的知识和经验[3].随着时间的推

* 基金项目:国家基础研究资助项目: 2015CB352200;国家自然科学基金 61432001,61421091, 61690200
收稿时间: 0000-00-00; 修改时间: 0000-00-00; 采用时间: 0000-00-00; jos 在线出版时间: 0000-00-00
CNKI 在线出版时间: 0000-00-00

移,软件需求会发生变更,其他的开发者会根据新需求,结合自己的理解,对代码进行修改.开发者的知识经验通过这些开发活动逐渐沉淀在代码文件中,形成其特有的贡献组成. 在这种协同开发模式下,可以最大限度地发挥集体智慧的优势,对软件不断进行更新、优化和发展.加之,由于协同开发模式大多提供了支撑工具, 如:项目主页, 代码库, 缺陷追踪系统, 邮件列表或者论坛, Wiki, 持续集成环境等, 使得开发人员的活动可以被有效的记录, 因此, 其所产生的海量数据为数据驱动的智能化软件开发打下了基础. 如何利用这些大数据来优化软件开发过程, 成为一个亟待解决的问题.

对于代码文件来说,合理的贡献组成意味着合理的任务分配.一个功能复杂的模块往往需要多名开发者共同开发,因而一个代码文件往往会涉及多名开发者[4].如果开发者众多,则会导致人员沟通困难,责任分散,代码风格不一致等诸多问题,使得软件质量低下[5].相反,若代码文件开发任务集中在某一个人身上,当这名开发者因为某些原因退出时,则会导致该代码文件难以继续开发和维护.代码文件的贡献组成复杂多样,为了详细探究不同代码文件合理的贡献组成模式,首先需要建立一套有效的量度刻画代码文件的贡献组成.通过贡献组成量度的建立,对于理解软件开发过程,挖掘合理的贡献组成模式至关重要.

软件工程领域,已有的关于贡献组成的相关研究大多是研究代码所有权,例如:探究开源软件项目代码所有权的分布情况[4],将代码所有权作为影响因子用于缺陷预测等[6,7,8],缺乏对不同类型文件贡献组成的详细探究(例如:核心代码文件与配置文件贡献组成的差异性).考虑到已有研究工作的不足,本文的研究目标主要包括以下两个问题:

1)如何刻画代码文件的贡献组成?

2)不同类型的代码文件贡献组成是否有差异?合理的贡献组成模式是什么?

文章整体的研究思路如下.首先,本文从贡献组成的集中度、复杂度和稳定性三个维度出发,提出刻画贡献组成的三个量度.其次,将量度用于 Nova 项目,在其版本控制数据上建立贡献组成的量度,总结了 12 种通用文件类型,归纳出 3 种贡献组成模式.最后,结合邮件以及面对面访谈的方式,验证了不同类型代码文件贡献组成的合理性以及相关量度的有效性.本文的主要贡献有:1)从贡献组成的集中度、复杂度、稳定性出发,建立了一组量度刻画代码文件的贡献组成.2)探究了不同类型代码文件合理的贡献组成模式,总结了 12 种文件类型,归纳出 3 种贡献组成模式.3)针对不同类型的代码文件,从贡献组成角度,对软件开发过程给出了一些指导性建议.为帮助复制本文工作以及应用本文所提量度和方法,我们梳理了相关数据和脚本,并提供了下载链接[1].

本文剩余部分将按以下方式展开:第 1 节主要介绍相关工作研究现状;第 2 节介绍本文的研究方法以及数据来源;第 3 节从三个维度出发,建立相关量度刻画贡献组成;第 4 节基于上一节的量度,以 OpenStack 的 Nova 项目为研究对象,探究不同类型文件的贡献组成模式;第 5 节,通过邮件以及面对面访谈的方式,验证了量度的有效性以及贡献组成模式的合理性;文章最后对本文工作进行总结并展望未来工作.

1 相关工作

近年来许多研究表明,人员因素是影响软件质量的关键因素之一[9-14].代码文件的贡献组成从其开发者人员构成的角度,考虑了开发任务在不同开发者之间的分配情况,以及责任划分.软件工程领域,关于贡献组成的相关研究,大多围绕代码所有权[4,6,7,8,15].通过代码所有权可以反映出特定时段开发者对某一特定模块的贡献比例以及责任程度.

Mockus 等人探究了开源项目的代码所有权分布情况,发现核心开发者通常会同时参与多个代码文件的开发,同一代码文件也会涉及多名开发者.尽管代码所有权并没有给他们任何特殊的变更控制权,但是会增加其在社区的声望[4].Rahman 和 Devanbu 在多个开源项目上研究了代码所有权与开发者经验对软件质量的影响,发现关系紧密的代码大多来源于同一开发者,代码文件的质量与开发者经验密切相关[16].Bird 等人在其工作中定义了代码所有权的相关量度,并分析了其与软件缺陷的关系[7].他们根据开发者的代码所有权大小,将开发者分为该模块的主要贡献者和次要贡献者.其研究结论表明,代码所有权量度与软件缺陷数量具有很强的相关性,当一个模块的次要开发者越多

[1] https://github.com/SunflowerPKU/The-Analysis-of-Code-Files-Contribution-Composition-Pattern

时,其可能出现的缺陷数量也会越多.Foucault 等人在开源项目上重复了 Bird 的相关工作,发现代码所有权与缺陷数量的相关性并不是很强[6].Meneely 等人基于 Linux 内核项目,探究了对于单一模块开发者人数与缺陷数的关系,并未考虑开发者的贡献比.他们发现当一个代码文件超过 9 个人贡献时,出现缺陷的概率增加了 16 倍[17].在软件开发方法方面,极限编程作为一个轻量级的、灵巧的软件开发方式,信奉集体代码所有权,即开发者在任何时候都可以进行代码修改,没有开发者对任何一个特定的模块或技术单独负责,每个人都可以参与任何其它方面的开发[18].然而已有的研究并未证明该方式对于复杂大型软件项目开发的合理性[7].

上述研究都未考虑到不同文件类型贡献组成的差异性.为了深入理解代码文件的贡献组成,探究不同文件合理的贡献组成模式,本文首先以代码所有权为基础,提出了一组量度刻画代码文件的贡献组成.其次,将相关量度用于 Nova 项目,总结出了 12 通用文件类型,归纳出 3 种贡献组成模式.最后通过邮件以及面对面访谈的形式对相关进行了验证.

2 研究方法以及数据来源

2.1 研究方法

对于代码文件来说,什么样的贡献组成是合理的？为了解决这一问题,首先必须要使得“贡献组成”这一抽象概念具体化,即:建立一组合理的量度刻画代码文件的贡献组成.为了达到这一目的,本文从贡献组成与软件质量的关系出发,结合已有文献以及软件开发过程,从贡献组成的集中度、复杂度、稳定性三个维度，建立贡献组成的相关量度.其次,为了探究不同类型代码文件的贡献组成的差异性,挖掘合理的贡献组成模式,本文选择 Openstack 的核心项目 Nova 作为案例研究,以其 8 个成熟版本的版本控制数据作为实验数据,在其上建立相关量度.进而,我们根据代码文件的功能将其划分为 12 种通用类型.通过实验数据,我们发现不同类型的代码文件贡献组成存在一定的差异性,从而归纳出了 3 种贡献组成模式.最后,为了探究不同文件类型相应的贡献组成模式的合理性,我们以邮件以及面对面访谈的形式进行了验证.文章的研究思路如图 1 所示.

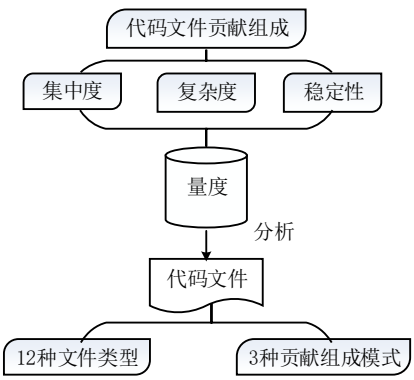

Fig.1 The framework of code file contribution composition research

图 1 代码文件贡献组成研究框架

2.2 数据来源

为了探究不同类型代码文件合理的贡献组成模式,本文选择了 OpenStack 的核心项目 Nov 作为案例研究,该项目主要负责 OpenStack 的计算功能.实验数据包括 Nova 从 Folsom 到 Mitaka 共 8 个成熟版本的版本控制日志数据,时间跨度为 2012 年 4 月到 2016 年 4 月.之所以选择 OpenStack 的核心项目 Nova 作为案例研究,主要是由于:1)贡献组成相对复杂,研究意义大.开源软件相对于传统软件开发模式更加灵活,志愿者可以根据兴趣对不同模块进行贡献,因而也导致其贡献组成的不确定性很强[2].而 OpenStack 作为开源项目的典型代表之一,其开发者众多,人员流动

性强,针对这种复杂的情况探究代码文件的贡献组成,使得本文的研究结果更有价值.2)发布周期稳定.OpenStack 的发布周期基本稳定,大约每 6 个月会有一个新版本的发布,这使得我们探究贡献组成随时间的变化情况成为了可能.3)提供了易于获取的标准化开发历史数据.目前 OpenStack 项目托管在 GitHub[1]平台上,使用的版本控制系统是 Git.版本控制系统中的日志数据记录了每一次代码提交的相关信息,主要内容有提交者,提交时间,提交编号,提交的注解,涉及到文件及对应的修改等,如图 2 所示.OpenStack 作为新兴活跃的开源社区,其官方网站可以非常方便的下载相关的软件开发活动数据,本文使用的数据是 OpenStack 官方提供的经过提取处理的代码提交日志数据的 MySQL 形式[2].4)Nova 项目复杂度高且持续活跃.Nova 项目是 OpenStack 中开发者最多(历史贡献人数:近 900 名),且持续活跃的软件项目,因而我们将其作为案例研究.

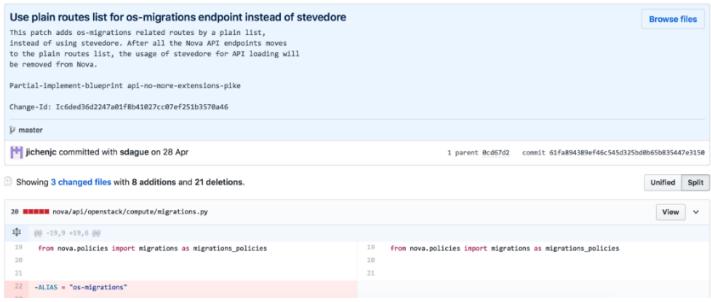

Fig.2 A log record of OpenStack on GitHub

图 2 GitHub 上 OpenStack 项目的一条日志记录

3 量度设计

为了深入理解代码文件的贡献组成情况,首先必须要建立一组合理的量度刻画代码文件的贡献组成.为了使量度能充分反映贡献组成,且具有应用价值,本文从其与代码文件质量关系的角度出发,结合已有文献基于代码所有权,进行量度设计.代码文件的贡献组成取决于其对应开发者的贡献度,即代码所有权.当开发任务集中于某位开发者时,代码文件的逻辑、编码风格更容易保持一致,但与此同时,当该名开发者突然离开时,该代码文件会出现无人监管状态;当开发任务过于分散,涉及多名开发者时,由于不同开发者对需求具有自己独特的理解,使得代码的整体逻辑和风格很难保持一致,而且也会导致沟通效率低下,责任分散;另外,开发者人员流动是否稳定,也是影响代码文件质量的关键因素之一.为了尽可能准确的反映贡献组成的相关特征且考虑到以上因素,本文基于代码所有权从贡献组成的集中度、复杂度和稳定性三个维度对软件制品的贡献组成进行刻画.

为了使量度在实际生产中具有更好的指导意义,我们基于文件粒度,设计了相关量度.考虑到文件粒度层次的原因是:开发者的知识经验直接体现为代码文件,最终表现为功能丰富的软件制品.而衡量开发者对某一模块的贡献度是基于版本控制系统中的提交记录数,这些提交记录是在文件级别上的,因此,从文件粒度层次对软件制品贡献组成量度进行设计是非常合理的.

首先,我们基于代码所有权,定义了一些基本符号和基本量度,这些符号和量度在下文中会用到.表 1 列举了下文所提出量度所涉及的一些基本概念以及符号表示.其中,代码所有权($Ownership_p^i$)是用来描述开发者对某一模块的贡献程度以及责任度,与开发者针对该模块的提交次数直接相关.如果某位开发者对该代码文件拥有较高的代码所有权，该名开发者一般能清楚该文件的代码逻辑,需求等,能对该文件后续的变更做出决策,当出现缺陷时,能够立即定位缺陷并对其进行修复.借鉴 Bird 等人的工作,开发者 p 对文件 i 的代码所有权表示如下:

[1] https://github.com/openstack

[2] http://activity.openstack.org/dash/browser/data_sources.html

$$Ownership_p^i = \frac{Commits_p^i}{TotalCommits^i}$$

本文使用开发者版本控制系统中的提交记录来衡量贡献度,忽略修改量(比如:添加或删除的代码行数).修改量与编程语言以及开发者的代码风格密切相关,因而即使完成同一任务,不同的实现方式代码量的差异也会非常大.而且代码行粒度太小,一行代码并无具体实际含义.而版本控制系统中的一条提交记录通常情况下与一条修改请求相对应,具体来说可能是增加新功能或修复缺陷[19].而且由于提交记录是在文件级别上的,所以当考虑具体的某条提交记录时,只需考虑其所涉及修改的文件数,不必考虑文件的代码行数.因而,使用版本控制系统中的提交记录数能更好的反映开发者的贡献度.另外,需要特别指出的是,当统计某位开发者的提交次数时,我们仅仅考虑对于源文件的修改,忽略非代码文件(比如:图片、配置文件等).

Table 1 Basic concepts and symbolic representations

表 1 基本概念以及符号表示

符号表示	含义
$Ownership_p^i$	开发者 p 对文件 i 的代码所有权大小
$Commits_p^i$	开发者 p 对文件 i 的提交次数
$TotalCommits^i$	文件 i 的总提交次数
$TotalContributors^i$	文件 i 的贡献者总人数

注:表中所有统计量都对应某一时间段

3.1 贡献组成的集中度

贡献组成集中度意在描绘软件制品的贡献组成在多大程度上是由某个特定的开发者集中贡献的.贡献组成集中度越高,说明该模块的开发任务越集中于某一位特定的开发者,也就是说,该名开发者完成了这个模块的主要开发任务.贡献集中往往有利于保持软件整体思路与代码文件风格的一致性,使该模块处于核心开发者的引导状态,降低缺陷的产生概率.另一方面,集中度越高则开发者的责任越明确,当该模块出现问题时,可以有效的定位开发者,更好的进行缺陷修复者推荐.相反集中度越低,则任务分配越分散,所涉及的开发者数量越多,因此会导致团队沟通困难,责任不明确,拖慢开发进度等问题.然而,需要特别指出的是,当开发任务主要集中于某位特定开发者时,当该名开发者突然离开,则可能导致代码文件处于无人监管的状态.

本文基于代码所有权定义贡献组成的集中度,对其进行量化度量有助于更好地理解文件的开发组织现状,也可结合历史的演化,发现其开发组织的规律,更好地进行后续代码文件的开发和改进.借鉴 Bird 等人的工作,我们定义如下量度来体现贡献组成的集中程度.

- 量度 1:源文件 i 的代码所有权:代码文件 i 的贡献者中代码所有权的最大值.

$$OWNERSHIP^i = \max\left(Ownership_p^i\right)$$

3.2 贡献组成的复杂度

贡献组成的集中性只考虑了高贡献比例开发者的信息(即其所占的代码所有权),而未考虑到所有开发者对软件模块的贡献组成情况.随着软件规模日益庞大,软件结构日益复杂,一个软件模块通常是由多人协同开发.多个开发者的知识经验以及他们对软件需求的理解交织在一起,共同对软件的功能和质量产生影响.因而有必要对软件模块的所有开发者的贡献进行刻画,从而反映贡献组成的整体情况.

贡献组成的复杂度体现的是代码文件的贡献来自不同开发者的不确定程度,反映了代码文件中所有开发者融合在一起的信息量.贡献组成越复杂,所包含的信息量越多,越难确定代码文件的开发者来源.在这里,本文基于代码所有权.并结合香农熵的定义刻画贡献组成的复杂度.香农熵描述的是事件的不确定性程度.在信息世界里,熵值越大,传递的信息量越多,熵值越小,则所包含的信息量越少.整个信息源的不确定性通过该信息源所包含的多个随机事件的不确定性的期望来表征.因而,香农熵定义如下:$H(x) = E[I(i)] = -\sum_{i=1}^{N} p_i \log_2 p_i$,其中$p_i$表示事件 i 发生的概率.其中,$I(i) = -\log_2 p_i$表示随机事件 i 的不确定度,也称为自信息量.软件制品的代码所有权,可以理解为贡献来源

于某位开发者的可能性.根据香农熵的定义,基于代码所有权,代码文件贡献组成的复杂度可以定义如下.

- 量度 2:代码文件 i 贡献组成的复杂度:表示代码文件 i 的贡献来源的不确定程度,用其所有开发者的代码所有权的自信息量的平均值表征.

$$ENTROPY^i = -\sum_{p=1}^{N} Ownership_p^i \, log_2 Ownership_p^i$$

其中,$\text{Ownership}_\text{p}^\text{i}$表示开发者 p 在代码文件 i 上的所有权,即某个提交属于开发者 p 的概率,N 表示贡献者人数.根据熵的定义,$\text{I(p)} = -\log_2\text{Ownership}_\text{p}^\text{i}$ 即为确定文件中的一个提交属于开发者 p 所需要的信息量.ENTROPY^i即为确定文件 i 中所有代码的贡献来源所需要的总信息,需要的信息越多,说明贡献组成越不确定,越无序.ENTROPY^i的取值范围为[0, $\log_2$N],当只有一个开发者完成文件所有的提交贡献时,则贡献组成可以确定全来自于一个开发者,ENTROPY^i为 0;当所有开发者均匀地对文件 i 进行提交贡献,即 $\text{Ownership}_\text{p}^\text{i}$=1/N, p = 1, 2,…, N, 则很难确定文件中的某一个提交来自于哪位开发者,贡献组成不确定性极高,此时ENTROPY^i为 $\log_2$N.

3.3 贡献组成的稳定性

在软件开发过程中,开发者的人员流动现象普遍存在,项目文件中的开发者经常会由于工作变换,内部人员调整或者其它个人原因而离开,对应地,也时不时有新的开发者加入到项目的开发当中.这些过程都会导致代码文件的开发者发生变动,其贡献组成也会有较大的改变,可能导致出现遗留代码、部分知识损失、以及对文件进行尚不成熟的开发而给文件带来质量风险.

本文中前两个维度都是从代码所有权的角度刻画软件制品的贡献组成,主要描述了开发者对软件模块的贡献度,并未考虑到开发者人员调度在不同周期间的变化情况.考虑到这一点,我们设计了第三个维度:贡献组成的稳定性.贡献组成的稳定性意在体现开发者的人员调度情况.稳定的贡献组成说明该模块开发者数量平稳,人员变更不频繁,则文件的开发风格、代码组织方式、算法思路等都不会发生太大的变动,开发者可以按照原先的理解及经验对文件进行开发贡献;而不稳定的人员组成说明人员调度频繁,则在开发过程中可能会引入更多的不确定性.软件制品贡献组成的稳定性可以定义如下.

- 量度 3:代码文件 i 的人员调度情况:表示相对于上一周期新加入以及离开的开发者人数.

$$STABILITY_t^i = New\_Developers_t^i + Left\_Developers_t^i$$

其中,$\text{New\_Developers}_\text{t}^\text{i}$,表示不在上一个周期参与贡献而在当前周期参与贡献的开发者人数,即: $\text{New\_Developers}_\text{t}^\text{i} = \text{count(p)}\ (\text{p} \in \text{Contributors}_\text{t}^\text{i} \wedge \text{p} \notin \text{Contributors}_{\text{t}-1}^\text{i})$; $\text{Left_Developers}_\text{t}^\text{i}$表示在上一个时间周期参与贡献而不在当前周期参与贡献的开发者人数,即:$\text{Left_Developers}_\text{t}^\text{i} = \text{count(p)}\ (\text{p} \notin \text{Contributors}_\text{t}^\text{i} \wedge \text{p} \in \text{Contributors}_{\text{t}-1}^\text{i})$.

4 代码文件贡献组成模式

通过第 3 章节,我们已经可以回答文章开头提出的第一个问题:刻画代码文件的贡献组成.接下来,我们就要通过这组量度度量代码文件的贡献组成,并探究不同类型的代码文件合理的贡献组成模式.本文选取了 Nova 从 Folsom 到 Mitaka 共 8 个成熟版本的版本控制日志数据来进行数据验证,时间跨度为 2012 年 4 月到 2016 年 4 月.Nova 项目旨在为计算资源提供大规模、可扩展的按需自助服务访问.代码文件数达 4517,在该时间周期内的总贡献者人数为 1107,提交总数为 19081.有研究发现,在软件项目中 60%以上的缺陷只涉及到项目中 1%以下的项目文件[20],因而本文实验将目标文件定位为项目中的活跃代码文件.实验从所有代码文件中选取提交量最多的前 10%的文件做为目标文件进行案例分析.

我们基于 Nova 的活跃文件,对上一节给出的三个量度分别进行计算.通过度量值,我们发现不同类型的代码文件的贡献组成分布存在明显的差异.因而,我们根据文件的功能结合代码文件的开发者贡献组成,对 Nova 项目的代码文件进行了大致的划分,总结出了 12 类常见的具有代表性的文件类型,如表 2 所示.接下来,从贡献组成的集中度、复杂度、稳定性三个维度,对 nova 项目代码文件的贡献组成情况进行分析,我们发现这 12 类文件整体上呈现出三种贡献组成模式,这三种贡献组成模式的具体分布如表 3 所示.

从实验结果可以看出,这三种模式存在明显的差异.模式一的特征是贡献组成分散、复杂度高、人员变动大.符合模式一的文件的共同特点是:一般与多个文件或模块进行交互、参与修改的人数多,修改频繁.例如:实现系统核心功能的文件,这类文件涉及到系统核心部分的实现,在整个系统中具有非常重要的地位.由于系统在不断演化,系统功能是否足够强大直接取决于核心功能模块,因而该类文件的修改频率非常高.另一方面,这类文件一般功能结构非常复杂,涉及的其它模块或文件众多,因而会产生许多关联修改.在 Nova 项目中,libvirt 是 Nova 使用的核心虚拟化技术,代码文件 nova/virt/libvirt/driver.py 负责对底层虚拟化的管理,该文件非常复杂,代码量达近 8000 行,因而修改频率相当高,所涉及的历史贡献者有 308 位.相对于核心功能文件,还有一类文件,这类文件逻辑结构并不复杂,但是其所关联的文件众多且被经常修改.例如:异常处理文件、活跃文件的测试文件等,当其所关联的文件发生修改时,这些文件也通常需要被修改,因而贡献组成分散、复杂度高、人员变动大.

Table 2 The file types of Nova

表 2 Nova 项目的文件类型

文件	特点
活跃文件的测试文件	负责测试的目标文件相对复杂,代码行数多,修改频率高
异常处理文件	提供了处理程序运行时出现的任何意外或异常情况的方法,通常与多个模块交互
权限管理文件	定义了不同场景下用户的操作权限,以及系统权限
核心接口文件	系统核心模块接口文件,提供复杂多样的接口实现,经常与系统其它模块进行交互
实现系统核心功能的文件	系统核心功能的实现,调用关系复杂,逻辑复杂,代码量非常大,修改频率高
模块的功能实现文件	负责某一模块的功能实现,功能相对独立
复杂模块的功能实现文件	系统某一模块的功能实现,该模块相对独立,但逻辑复杂,代码量大
模块接口文件	负责该模块向外提供接口的实现
模块测试文件	负责单一模块的测试任务
模块配置文件	负责单一模块的配置任务
非功能实现文件	未涉及系统的功能实现,逻辑接口简单,例如依赖库定义
i18n 文件	国际化文件,一般是系统的语言包

Table 3 The contribution composition pattern of different file types

表 3 不同文件类型的贡献组成模式

贡献组成模式	特征	度量值	文件类型
模式一	贡献组成分散 复杂度高 人员变动大	0WNERSHIP<=0.2 3=<ENTROPY 14=<STABILITY	实现系统核心功能的文件
			复杂模块的功能实现文件
			活跃文件的测试文件
			异常处理文件
			权限管理文件
			核心接口文件
模式二	贡献组成较集中 复杂度较高 人员变动较大	0.2<OWNERSHIP<=0.7 2<=ENTROPY<3 5=<STABILITY<14	模块的功能实现文件
			模块接口文件
			模块测试文件
			模块配置文件
模式三	贡献组成集中 复杂度低 人员变动稳定	0.7<OWNERSHIP<=1 0<=ENTROPY<2 0=<STABILITY<5	非功能实现文件
			i18n 文件

注:表中不同模式的度量值的取值范围需要根据具体项目进行调整

模式二的特征是贡献组成较集中、复杂度较高、人员变动较大.项目中有相当数量的文件都属于这种模式.这类文件一般从属于某一模块,功能相对独立,开发者相比模式一较为稳定.例如:模块的功能实现文件,这类文件主要

负责系统某一模块的功能实现,功能较集中,是该模块的核心文件.如 Nova 项目中 nova/conductor/manager.py ,该文件负责数据库查询.在 Nova 项目中,数据库操作都是通过 conductor 类实现的,主要是为了避免计算节点直接访问数据库,增加系统的安全性.该文件定义了许多数据库访问的方法,这些方法负责建立本机与数据库的连接.当数据库发生修改时,该文件通常也会发生修改,因此修改频率较高.但是相比模式一,其功能相对集中,因而代码文件的贡献组成相对稳定.

模式三的特征是贡献组成集中、复杂度低、人员变动稳定.符合这种模式的代码文件产生缺陷的可能性相对最低,这类文件的共同特点是:基本不涉及项目的功能实现,代码量少,逻辑结构简单.例如:非功能实现文件,如:项目的策略文件,该类文件定义了项目中的某些规则,例如用户名,密码标准,图片大小等.这类文件一般结构简单,代码量少.而且由于这些规则一般会比较稳定,因而单一发布周期内文件修改频率非常低.还有一类文件也符合模式三,例如:i18n 文件,该文件里面存放着系统的区域语言设置,可以使系统支持国际化信息显示.在 Nova 项目中,nova/locale/de/LC_MESSAGES/nova.po 就属于这类文件.

需要特别指出的是,尽管从实验结果可以看出这 12 类文件整体上呈现三种贡献组成模式,然而,仍有相当数量文件未能被正确划分,比如:nova 项目中的 requirements.txt,该文件属于非功能实现文件,负责记录所有依赖包及其精确的版本号.尽管该文件逻辑简单,但是其所涉及的贡献者多样,人员流动性很大.之所以不能做到所有文件精确划分,考虑到以下几方面原因:1)文件类型的定义存在一定的交叉,比如:模块的功能实现文件与系统核心功能实现文件;2)度量值是连续的,处在边界值附近的很容易被划分为另一种贡献组成模式;3)代码的贡献组成是动态变化的,受很多因素影响,比如:项目的计划的变更、项目的进度、开发者自身原因等.尽管由于以上几方面因素，导致不能将所有文件的贡献组成模式精确划分，但必需要指出的是:从文件类型的角度探究贡献组成,可以看出其总体上存在一定的规律,对于理解不同类型的文件的贡献组成模式,发现潜在风险文件很有价值.

5 有效性验证

为了验证量度的有效性以及实验结果的合理性,从而更好地理解代码文件的开发者贡献组成对实际开发活动的影响.这一节主要是从定性的角度去探究本文提出的量度的价值,以及不同文件类型的贡献组成模式的合理性,通过与实践中的开发者进行邮件交流以及面对面访谈的方式.以下介绍邮件访谈方法的各方面信息.

访谈主题:主要针对量度的意义,文件分类的有效性,贡献组成模式的合理性展开讨论.

访谈方式:邮件交流,面对面访谈.

访谈对象:邮件交流的对象,近期在 Nova 项目中的活跃开发者、核心开发者;访谈对象,有多年开发经验的开源项目贡献者.

访谈内容:通过向实践者描述开发者贡献组成的各个量度的定义,展示基于 Nova 项目进行案例分析,所得出的不同文件类型的开发者贡献组成模式,从而获得他们对于开发者贡献组成的看法.

访谈结果分析:该部分是将访谈内容进行话题筛选、主题提取后得到的分析结果,主要从两个角度阐述实践者们的观点.

考虑到开发人员,尤其是经验丰富的开发者,本身工作的任务量就非常大,闲暇时间非常宝贵.因而,我们没有采用大规模调研的方式,而是针对少量的特定受访者,例如:我们选择了 Nova 项目近期提交数前一百名的开发者,最终收到了 16 份有效回复.关于面对面访谈,我们采访了在 Nova 项目参与度较大的三个公司(华为、VMware、Redhat)的四位资深开发人员(OpenStack 的开发经验都在三年以上),每位受访者的受访时间控制在三十分钟左右.尽管这部分人员较少,但是由于访谈的目的是对问卷结果的补充,因而也获得了很多非常有价值的信息.在访谈和邮件交流的设计中,我们充分借鉴相关调研方法[21],希望在有限的受访者中得到较高的有效回复率.例如:1)我们针对每位受访者设计了个性化的问卷;2)尽可能简要清晰的描述我们的调查问题,尽量少使用晦涩的专业术语,问卷采用封闭式与开放式结合的方式;3)所有回复确保匿名;4)问卷整体的浏览及回答时间控制在十分钟之内.

5.1 量度的价值

对于开发者贡献组成的三个维度,实践者给出了他们的看法.1)贡献组成的集中度:在项目实践里,有时会需要

通过查找特定文件的提交记录数来确定文件的核心开发者,通过该量度可以很容易的定位出项目中的核心开发者,有利于帮助开发者快速找到文件的专家,来进行重要缺陷的修复或做提交代码审查,减轻实践中的工作量;对于新进的开发者来说,可以帮助他们找到资深开发者,从而节省沟通时间;另外,可以定位到每个文件的“负责人”,当代码有任何提交申请或出现缺陷时可以及时通知到“负责人”,让其能清楚掌握文件的动态.2)贡献组成的复杂度:通过该量度可以实时的反映代码文件所处的开发状态,方便开发者以及项目的管理者了解哪些文件逐步趋于稳定,哪些文件处于活跃状态;另外,根据不同文件所处的开发状态,可以对应地调整代码审查的力度;3)贡献组成的稳定性:该量度可以反映出代码文件贡献者的稳定性.当有大量新开发者加入可能意味着文件正在完成新功能或做较大改动.项目管理者可以明确了解到这种变动,从而做好与开发者间的交流工作以使开发更好开展.

5.2 贡献组成模式合理性

实践者指出,从文件类型的角度,探究开发者贡献组成模式是非常有意义的.有助于开发者实时了解文件的贡献组成状态,探究不同文件的合理贡献组成方式,进而识别风险文件.实践者也基本赞同本文对文件类型的划分结果,且表示分析的贡献组成模式基本符合实际开发.

6 建议

通过问卷和访谈，针对三种贡献组成模式,实践者也给出了相关的建议.1)贡献组成模式一:这种模式下,代码文件的贡献者非常多,容易导致文件的思维逻辑不一,代码风格不一致,常常难以维护.另外,由于缺乏核心贡献者,人员流动大,使文件处于无领导状态,导致文件的风险性很高,所以应该特别关注.对于实现系统核心功能的文件 、复杂模块的功能实现文件等,这类文件逻辑复杂、代码量大,一旦出现问题会直接导致系统不能正常工作.由于该文件重要性极高,针对这类文件最好由大于一名的核心开发者进行开发和维护,这样在核心开发者离开时,不会出现无监管状态. 对于异常处理文件、权限管理文件这类文件修改频率高,贡献者众多,但是由于其逻辑简单,因而即使贡献组成非常分散、人员变动大,也不会对文件质量产生很大的影响.2)贡献组成模式二:这种模式下,代码文件的贡献者相对较多,所有贡献者的贡献度存在一定的差异性,存在核心开发者.这类文件一般属于功能模块文件,相对于模式一,人员相对稳定,虽然修改频率较高,但是由于贡献较集中,因而代码质量能得到一定程度的掌控.文件风险一般.3)贡献组成模式三:这种模式下,文件的代码所有权非常高,贡献者差异非常大,贡献者相对稳定.这类文件一般逻辑简单,代码量非常少,文件的风险等级最低.

另外,关于识别风险文件,实践者指出:文件的风险控制主要针对模式一.当某类文件的贡献组成集中度非常小,复杂度很高,稳定性很低时,特别是功能非常复杂、核心度高的代码文件.这类文件处于高风险状态,应该着重关注;当某类文件的贡献组成变化非常大时,应该重点关注.比如:当前周期相对于上一周期,贡献者人数突增,或突然下降.实践者也提到,严格的代码审查机制会在一定程度上削弱开发者贡献组成对代码文件质量的影响.

7 总结与展望

随着代码文件的开发演化,开发者会根据自己的兴趣结合软件需求对不同的模块进行贡献.合理的开发者贡献组成意味着合理的开发任务分配和人员调度,有利于代码文件的高效高质开发;而开发任务的分配不合理及开发者调度问题可能引起文件中的开发者贡献组成出现冲突,导致文件在开发过程中会出现较多缺陷,拖慢开发进度,增加开发负荷,对文件质量带来风险.通过实现以数据为驱动的智能化软件开发,利用软件开发过程所产生的海量开发者行为数据,从开发者贡献组成的角度,实时了解代码文件的开发状态,针对不同类型的代码文件,探究合理的贡献组成模式,定位风险文件,对于软件项目的健康发展意义重大.

为了探究不同类型代码文件合理的贡献组成模式,本文首先从贡献组成的集中度、复杂度、稳定性三个维度出发,设计了相关量度刻画代码文件的贡献者组成情况.其次,以 Nova 作为案例研究,选择其版本控制系统的日志数据作为实验数据,分析开发者贡献组成的分布情况.实验结果显示,不同类型的代码文件的开发者贡献组成呈现出一定的差异性.通过对代码文件的功能分析结合其贡献组成情况,我们总结出了 12 种通用文件类型,3 种开发者贡献组成

模式.为了进一步,探究量度的价值,验证开发模式的合理性,我们通过邮件和面对面访谈的形式,获得实践者对于贡献组成的认识,以及对于不同类型代码文件的合理贡献组成模式的看法.文章最后也对不同类型的代码文件的贡献组成给出了一些指导性建议.

本文的工作目前还存在一些不足,未来将在以下几个方面进行继续探索:

1. 从模块或功能点出发,探究开发者的贡献组成.在访谈的过程中,有实践者指出,在实际开发中,社区更关注的是模块或功能点,因为单一代码文件并没有实际的含义.因此,后续应从不同粒度层次出发,探究开发者的贡献组成情况,从而给开发者和社区管理者提供更多有价值的参考信息.
2. 挖掘更多实践关注的维度信息.本文主要是从贡献组成的集中度、复杂度、稳定性三个维度出发,探究代码文件的贡献组成情况.为了能帮助项目更好地了解代码文件的开发现状,尽早发现潜在问题,后续还需要再进行深入调研,挖掘实践中开发者关注的信息,提供更多有价值的、能在实践中帮助项目开发的维度.
3. 探究不同类型的软件项目的开发者贡献组成情况.本文选择了 OpenStack 的核心项目 Nova 作为案例分析,尽管实践者指出实验得出的文件类型以及贡献组成模式具有一定的通用性,为了更好地指导实际开发活动,后续应关注不同类型的软件项目,探究开发者贡献组成情况.

References:

[1] Cataldo M, Wagstrom P A, Herbsleb J D, et al. Identification of coordination requirements: implications for the Design of collaboration and awareness tools[C]//Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work. ACM, 2006: 353-362.

[2] Ma X, Zhou M, Mei H. How developers participate in open source projects: a replicate case study on JBossAS, JOnAS and Apache Geronimo[C]//Proceedings of the 1st International Workshop on Replication in Empirical Software Engineering Research. 2010.

[3] Rigby P C, Zhu Y C, Donadelli S M, et al. Quantifying and mitigating turnover-induced knowledge loss: case studies of chrome and a project at avaya[C]//Proceedings of the 38th International Conference on Software Engineering. ACM, 2016: 1006-1016.

[4] Mockus, Audris, Fielding, et al. Two case studies of open source software development: Apache and Mozilla[J]. Acm Transactions on Software Engineering & Methodology, 2002, 11(3):309-346.

[5] Pendharkar P C, Rodger J A. An empirical study of the impact of team size on software development effort[J]. Information Technology & Management, 2007, 8(4):253-262.

[6] Foucault M, Falleri J R, Blanc X. Code ownership in open-source software[C]// International Conference on Evaluation and Assessment in Software Engineering. ACM, 2014:39.

[7] Bird C, Nagappan N, Murphy B, et al. Don't touch my code! :examining the effects of ownership on software quality[C]// ACM Sigsoft Symposium and the, European Conference on Foundations of Software Engineering. ACM, 2011:4-14.

[8] Greiler M, Herzig K, Czerwonka J. Code ownership and software quality: a replication study[C]// Mining Software Repositories. IEEE, 2015:2-12.

[9] Bird C, Nagappan N, Gall H, et al. Using Socio-Technical Networks to Predict Failures[J]. Proceedings of the IEEE International Symposium on Software Reliability Engineering, 2009.

[10] Cataldo M, Wagstrom P A, Herbsleb J D, et al. Identification of coordination requirements: implications for the Design of collaboration and awareness tools[C]// Anniversary Conference on Computer Supported Cooperative Work. ACM, 2006:353-362.

[11] Nagappan N, Murphy B, Basili V. The influence of organizational structure on software quality[C]// ACM/IEEE, International Conference on Software Engineering. IEEE, 2008:521-530.

[12] Pinzger M, Nagappan N, Murphy B. Can developer-module networks predict failures?[C]// ACM Sigsoft International Symposium on Foundations of Software Engineering. ACM, 2008:2-12.

[13] Cockburn A, Highsmith J. Agile Software Development: The People Factor[M]. IEEE Computer Society Press, 2001.

[14] Matsumoto S, Kamei Y, Monden A, et al. An analysis of developer metrics for fault prediction[C]// International Conference on Predictive MODELS in Software Engineering, Promise 2010, Timisoara, Romania, September. DBLP, 2010:1-9.

[15] Bird C, Nagappan N, Devanbu P, et al. Does distributed development affect software quality? An empirical case study of Windows Vista[J]. Communications of the Acm, 2009, 52(8):85-93.

[16] Rahman F, Devanbu P. Ownership, experience and defects: a fine-grained study of authorship[C]// International Conference on Software Engineering. ACM, 2011:491-500.

[17] Meneely A, Williams L. Secure open source collaboration: an empirical study of linus' law[C]// ACM Conference on Computer and Communications Security. ACM, 2009:453-462.]

[18] K. Beck and C. Andres. Extreme Programming Explained: Embrace Change. Addison-Wesley Reading, MA, 2005.

[19] Zhou M, Mockus A. Developer fluency: achieving true mastery in software projects[C]// Eighteenth ACM Sigsoft International Symposium on Foundations of Software Engineering. ACM, 2014:137-146.

[20] Mockus A, Hackbarth R, Palframan J. Risky files: An approach to focus quality improvement effort[C]//Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, 2013: 691-694.

[21] E. Smith, R. Loftin, E. Murphy-Hill, C. Bird, and T. Zimmermann, “Improving developer participation rates in surveys,” in CHASE, May 2013, pp. 89–92.