软件学报 ISSN 900-9825, CODEN RUXUEW
Journal of Software, [doi: 9.13328/j.cnki.jos.000000]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-9-62562563

软件开发活动数据质量问题的研究*

涂菲菲, 周明辉

(北京大学 信息科学与技术学院,北京 100871)

通讯作者: 周明辉, E-mail: zhmh@pku.edu.cn

摘 要: 问题追踪系统和版本控制系统等软件开发支持工具已被广泛应用于开源和商业软件的开发中,产生了大量的数据,即,软件开发活动数据.软件开发活动数据被广泛应用于科学研究和开发实践,为智能化开发提供支持.然而,数据质量对相关的研究和实践有重大影响,却还没有得到足够的重视.为了能够更好地警示数据使用者潜在的数据质量问题,本文通过文献调研和访谈的方式,总结出了 9 种数据质量问题,覆盖了数据产生、数据收集和数据使用三个不同的阶段.进一步地,本文提出了相应的方法帮助发现和解决数据问题.发现问题指加强对数据上下文的理解和通过统计分析及数据可视化发现潜在的数据质量问题.解决问题指利用冗余数据或者挖掘用户行为模式进行修正.

关键词: 数据质量;数据产生;数据收集;数据应用;问题追踪数据;版本控制数据

中图法分类号: TP311

中文引用格式: 涂菲菲,周明辉.软件开发活动数据质量问题的研究.软件学报. http://www.jos.org.cn/900-9825/ 0000.htm

英文引用格式: Tu FF, Zhou MH. Study of quality in software development activity data. Ruan Jian Xue Bao/Journal of Software, 2018 (in Chinese). http://www.jos.org.cn/900-9825/0000.htm

Study of quality in software development activity data

TU Fei-Fei, ZHOU Ming-Hui

(School of Electronics Engineering and Computer Science, Peking University, Beijing 90871, China)

Abstract: Software development tools, such as Issue tracking system (ITS) and version control system (VCS), are widely used in the intelligent development of open source software and commercial software. When using these tools to assist software development, they produce substantial amount of data, which is called software development activity data. Data quality has attracted more and more attention with increasingly rich software activity data sources and their wide uses. Faithfully, data is the basis of intelligent development. Data quality has influence on research and practice. To remind data users of latent data quality problem of software developement activity data, we indicate three aspects that may have data quality problems through literature review and interview with data users. The data quality problems arose from three phases, i.e., data production, data collection, and data use. Next, to improve the data quality of software development activity data, we propose several recommendations that could be taken into consideration, including finding data quality problems and solving data quality problems. First of all, researchers should have a clear understanding of the context of data. Next, they may use statistical analysis and data visualization to find latent data quality problems. Finally, they can try to correct the particular problems by redundant data or to improve data quality by user behavior analysis.

Key words: data quality; data production; data collection; data use; issue tracking data; version control data

软件开发支持工具(例如,问题追踪系统,版本控制系统)已被广泛应用于开源和商业软件的开发中.这些软

* 基金项目: 国家自然科学基金(00000000, 00000000);

Foundation item: National Natural Science Foundation of China (00000000, 00000000); State Key Laboratory for Novel Software Technology (Nanjing University)开放课题 (KFKT00000000)

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

件开发支持工具在使用过程中产生了大量的数据,这些数据统称为软件开发活动数据.软件开发活动数据覆盖范围非常广,例如,代码提交[1],功能需求的讨论[2],开发者在集成开发环境中的操作步骤[3]等.软件开发活动数据被广泛应用于软件开发过程中的各种决策,例如,比较不同软件方法之间的效果[1],估算开发者的生产效率[4],预测软件质量[5].

软件开发活动数据的广泛使用使其数据质量受到了越来越多的关注.如果软件开发活动数据存在质量问题,这可能使得基于问题数据的方法、软件产生的结果存在偏差,甚至无效.例如,Kim 等人[6]在利用问题追踪数据智能化预测任务完成时间的工作中,将问题报告被标记为“已解决”的时间点视为该任务完成的时刻.然而,Zheng 等人[7]发现任务完成的时间存在问题——开发者在完成任务后可能并不会及时将问题报告标记为“已解决”,而是在之后清理问题追踪系统时,通过脚本进行批量处理,即,“任务完成时间”并不能真正代表任务被解决的时刻.因此,Kim 等人的结果存在偏差.

虽然已有一些研究工作注意到了数据质量问题[7,17,18],但文献中往往只隐式地提及这些数据质量问题,缺乏对这些数据质量问题进行综合分析的工作.并且,数据质量问题并没有引起足够的重视,多数工作并不会提及其数据处理的细节,例如数据获取来源,数据质量情况,数据预处理步骤等.为了揭示潜在的数据质量问题,以更好地帮助软件开发活动数据的应用,本文主要研究回答以下两个问题:

1） 软件开发活动数据可能存在哪些质量问题?

2） 如何发现和解决软件开发活动数据的质量问题?

文章通过文献调研和访谈的方式进行分析,总结出 9 种潜在的数据质量问题,覆盖了数据产生、数据收集、数据使用三个阶段.并提出方法以帮助对数据问题的发现和修正.

本文将按以下方式组织:第 1 节主要介绍相关背景;第 2 节介绍本文的研究方法;第 3 节从数据质量问题覆盖的三个阶段出发,对软件开发活动数据的质量问题进行详细介绍;第 4 节总结了发现和解决软件开发活动数据质量问题的方法;文章最后对本文工作进行总结并展望未来工作.

1 背景介绍

常见的软件开发活动数据包括问题追踪数据和版本控制数据.

问题追踪数据是指在项目开发过程中出现的各种缺陷以及新增加的功能需求等的解决过程留下的数据,每个缺陷或者功能需求被称为一个问题(issue).每个问题都包含一些特定信息,如谁发现的这个问题,这个问题被分配给谁来解决,这个问题的当前状态,这个问题的优先级等等.问题报告的数量能在一定程度上反映代码的质量,问题报告的解决速度能反映开源项目的活跃程度等,因此问题追踪数据对于分析软件项目的最佳实践具有重要意义.

版本控制数据是指代码库每一次变更的历史数据,每次代码的更新都被称为一个代码提交(commit).每次代码提交记录了修改的原因,谁做了修改,修改了什么.版本控制数据不仅记录了当前的代码库,还记录了整个代码库演化的过程,这些数据对于分析项目的历史、当前状态以及未来都非常有价值.

1.1 问题追踪数据

问题追踪系统主要用于帮助开发人员追踪软件缺陷和需求,目前最常见的问题追踪系统有 Bugzilla 和 JIRA 等.

对于每个问题报告,问题追踪系统记录了报告的基本信息和报告的历史活动两部分.第一部分是报告的基本信息,主要包括序列号、报告标题、报告人(Reporter)、问题描述(Description)、报告当前的状态(Status)、处理意见(Resolution)、问题产生的环境等信息,如图 1 所示.

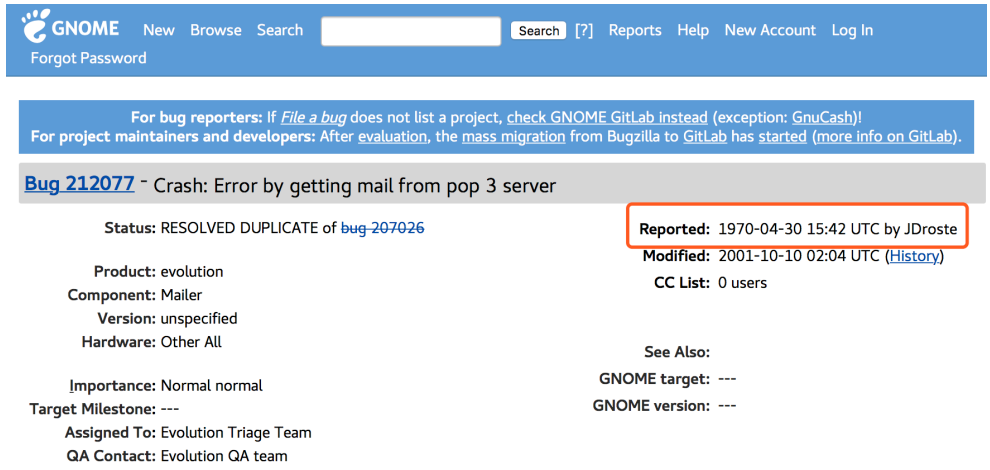
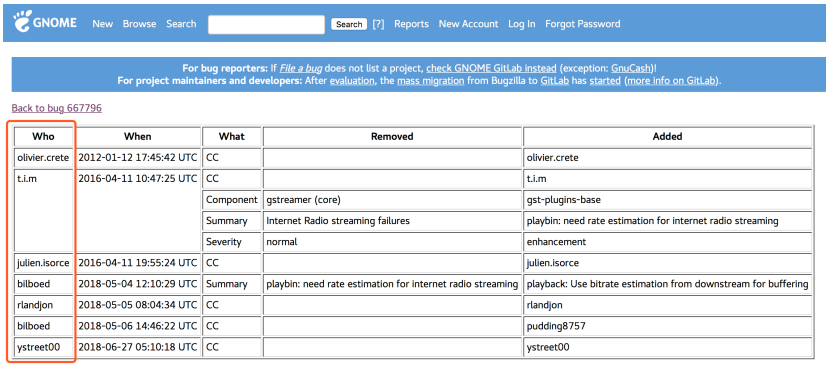


Fig.1 Gnome 项目的 212077 号问题报告的基本信息

图 1 The basic information of issue report of #212077 in Gnome

问题追踪数据具有一个特点：问题报告的信息有可能随着时间而发生改变.例如一个问题被报告到问题追踪系统之后,会被其他人评论,会被调整优先级,在问题解决之后其状态会被标记为“已解决”状态,甚至在关闭之后被重新打开.这些操作都会使该问题报告的信息发生变动.

因此,问题报告的第二部分是报告的历史活动,如图 2 所示.表中的每一行记录了对报告的一次修改,记录的内容包括:修改人(Who)、修改时间(When)、修改域(What)、修改前的值(Removed)和修改后的值(Added).



Who	When	What	Removed	Added
olivier.crete	2012-01-12 17:45:42 UTC	CC		olivier.crete
t.i.m	2016-04-11 10:47:25 UTC	CC		t.i.m
		Component	gstreamer (core)	gst-plugins-base
		Summary	Internet Radio streaming failures	playbin: need rate estimation for internet radio streaming
		Severity	normal	enhancement
julien.isorce	2016-04-11 19:55:24 UTC	CC		julien.isorce
bilboed	2018-05-04 12:10:29 UTC	Summary	playbin: need rate estimation for internet radio streaming	playback: Use bitrate estimation from downstream for buffering
rlandjon	2018-05-05 08:04:34 UTC	CC		rlandjon
bilboed	2018-05-06 14:46:22 UTC	CC		pudding8757
ystreet00	2018-06-27 05:10:18 UTC	CC		ystreet00

Fig.2 Gnome 项目的 667796 号问题报告的历史活动

图 2 The activity history of issue report of #667796 in Gnome

1.2 版本控制数据

版本控制系统的核心要求是能够方便团队协同开发以及记录历史版本,目前最常见的版本控制系统有 Git,Mercurial,SVN,和 CVS 等.

版本控制系统在代码提交日志里记录了开发者的每次代码提交信息.如图 3 所示,开发者的一次代码提交,一般包括如下信息：提交者,作者,代码提交的时间,代码修改的时间,对该次提交的注解,修改的文件以及对每个文件修改的内容.

```
commit 628e366df11c0a61487522ec1d4bca5c77fe9083
Author:     Andreas Gruenbacher <agruenba@redhat.com>
AuthorDate: Mon Jun 4 07:56:51 2018 -0500
Commit:     Bob Peterson <rpeterso@redhat.com>
CommitDate: Mon Jun 4 07:56:51 2018 -0500

    gfs2: Iomap cleanups and improvements

    Clean up gfs2_iomap_alloc and gfs2_iomap_get.  Document how
    gfs2_iomap_alloc works: it now needs to be called separately after
    gfs2_iomap_get where necessary; this will be used later by iomap write.
    Move gfs2_iomap_ops into bmap.c.

    Introduce a new gfs2_iomap_get_alloc helper and use it in
    fallocate_chunk: gfs2_iomap_begin will become unsuitable for fallocate
    with proper iomap write support.

    In gfs2_block_map and fallocate_chunk, zero-initialize struct iomap.

    Signed-off-by: Andreas Gruenbacher <agruenba@redhat.com>
    Signed-off-by: Bob Peterson <rpeterso@redhat.com>

 fs/gfs2/bmap.c  | 205 ++++++++++++++++++++++++++++++++-------------------------
 fs/gfs2/bmap.h  |   6 +-
 fs/gfs2/file.c  |   6 +-
 fs/gfs2/inode.c |   4 --
 4 files changed, 126 insertions(+), 95 deletions(-)
```

Fig.3 Linux kernel 项目中的一次代码提交

图 3 Commit log in Linux kernel

2 研究方法

问题追踪数据和版本控制数据记录了软件从需求到代码实现的过程,是软件开发中非常重要的数据.本文研究主要针对这两类数据.

为了对数据质量问题进行全面的总结,我们首先在 DBLP 数据库中进行检索,时间范围是 2000 年到 2017 年,检索时采用的英文关键词包括“bug report”,“issue report”,“code repository”,“software repository”,共检索到 501 篇文章;然后,对检索出的论文,通过人工审查方式移除掉与研究问题无关的论文,并通过查阅相关论文的参考文献和相关研究人员发表的论文列表来进一步识别出遗漏的论文.

进一步地,我们基于自身的研究和实践经验,对 Gnome、Mozilla 和 Linux kernel 数据进行分析,观察可能的错误所在,并跟研究社区进行互动以理解数据问题存在的广度与深度.

在调研的过程中,我们发现许多的工作并不会清楚的说明论文中使用的数据集的来源,对数据进行的预处理也很少提及,例如,Tamrawi 等人[19]在任务分配的工作中,并没有提及如何建立邮箱地址和独立开发者之间的关联关系.另外,在调研的过程中,我们去除了“问题报告错误分类”这一类已经非常有影响力的工作.我们更多关注的是,在软件开发活动数据的分析中,容易忽略的数据质量问题.

3 数据质量问题

本章节回答第一个研究问题：软件开发活动数据可能存在哪些质量问题?

数据质量问题可能产生于与数据有关的任何阶段,其产生的原因也多种多样.在使用软件开发支持工具的过程中,工具记录了与开发活动相关的数据,这个过程为数据的产生过程.数据使用者通过各种方式,将软件开发支持工具记录的数据收集到本地,这个过程为数据的收集过程.数据使用者基于自身对数据的理解,利用数据解决各种研究问题,这个过程为数据的使用过程.

如图 4 所示,我们根据问题可能产生的时间点,即,数据产生阶段,数据收集阶段,和数据使用阶段,将数据质

量问题分为三类.数据产生阶段的数据质量问题包括前文提到过的“任务完成时间”的陷阱,问题追踪数据的创建时间错误,和版本控制数据中的时间问题.数据收集阶段的数据质量问题包括爬取数据不完整问题,和隐私保护和安全保护等导致的数据不完整问题.数据使用阶段的数据质量问题包括未来数据泄露问题[22],邮箱地址的问题[9],版本控制数据中关于作者与提交者的问题.

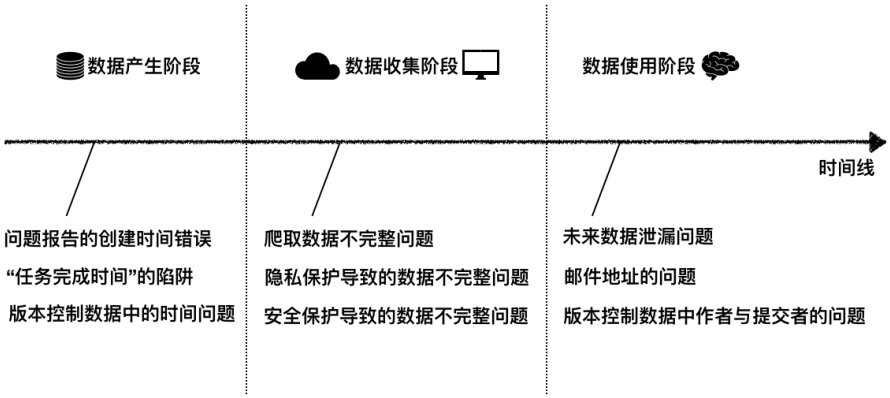


Fig.4 The three phases of data quality problems of software developemt activity data

图 4 软件开发活动数据质量问题产生的三个阶段

不同阶段的数据问题具有不同的特性.数据产生阶段的问题通常是因为软件开发流程(支持工具或基础设施)等所造成的数据使用者无法控制的问题.数据收集阶段的问题多数可以通过优化数据收集方法而得到解决或者部分解决,属于部分可控.数据使用阶段的问题在本文中多指“数据误用问题",即数据使用者忽略了数据本身的某些特征,误用数据导致问题.这类问题往往较为隐蔽不易发现,需要数据使用者对数据有深刻的理解.

这 9 个数据质量问题有些源自数据经验,有些是从文献中归纳总结的.为了使本文更具实践价值,对于每一个数据问题,我们列出其影响的项目.如果数据问题源于数据经验,那么其影响的项目就是指 Gnome、Mozilla、Linux kernal；如果数据问题源于文献调研,那么除了 Gnome、Mozilla、Linux kernal,其影响的项目还包括相应文献中涉及的项目.表 1 列出了这 9 个数据质量问题的来源和影响的项目.

Table 1 The 9 data quality problems of software developemt activity data

表 1 软件开发活动数据的 9 个数据质量问题

质量问题产生的阶段	软件开发活动数据质量问题	信息来源	受影响的项目
数据产生阶段	问题报告的创建时间错误	数据经验	Gnome
	“任务完成时间”的陷阱	[7]	Mozilla
	版本控制数据中的时间问题	数据经验	Linux kernel
数据收集阶段	爬取数据不完整问题	[9]	Mozilla, Gnome
	隐私保护导致的数据不完整问题	[4, 9]	Gnome
	安全保护导致的数据不完整问题	[9]	Mozilla
数据使用阶段	未来数据泄露问题	[22]	Mozilla, Eclipse, Gnome
	邮箱地址的问题	[11, 22]	Mozilla, Eclipse, Gnome
	版本控制数据中关于作者与提交者的问题	[11, 12]	Gnome

3.1 数据产生阶段的数据质量问题

数据产生阶段的数据质量问题主要是指,记录在软件开发支持工具中的数据并不能真实地反映真实的开发活动.这些有问题的数据产生的原因多种多样,可能是软件在记录过程中出现了问题,也可能是由于特殊的软件开发实践.

对于许多工作来说,时间是一个非常重要的研究因素[6,9,15,16].直观上,相较于其他文本数据而言,数据中记录的时间的准确性相对较高.因为时间的记录形式简单,通常为数字;含义明确,表示了某项活动发生时的系统时间.而且时间通常是由软件自动记录,不涉及人工输入.但是我们所发现的数据产生阶段的数据质量问题全部与时间相关.

3.1.1 问题报告的创建时间错误

当问题报告被创建时,问题追踪系统自动记录了该问题报告的创建时间.如图2所示,在Gnome项目中,有一些问题报告的创建时间为1970年.这显然是错误的时间数据,因为Gnome基金会最初建立于2000年[1],不可能在1970年报告问题.这类问题的出现可能是源于软件记录的错误.软件系统中记录的时间通常与UNIX时间戳对应,而UNIX时间戳是从1970年1月1日开始计算.因此当软件系统发生错误时,记录的错误时间通常在1970年附近.

3.1.2 版本控制数据中的时间问题

前文已经介绍,版本控制系统中存在代码作者和代码提交者两种不同的角色.代码作者指实际写代码的人.代码提交者指将写好的代码提交到版本控制系统中的人.根据活动发生的时间看,代码提交的时间不会早于写代码的时间.如果代码提交者同时也是代码作者,那么代码提交的时间可能与写代码的时间相同,或者代码提交的时间晚于写代码的时间.如果代码提交者和代码作者不同,那么代码提交的时间晚于写代码的时间.在Linux kernel中,存在一些代码提交数据,它们的代码提交者和作者不同,但是代码提交时间和写代码时间相同.根据前面的分析,这明显是存在问题的数据.那么在根据时间戳追溯代码变更的过程时,这个问题就会造成困扰.

3.1.3 “任务完成时间”的陷阱

“任务完成时间”的陷阱就是前文中举例的问题.有许多工作利用问题报告的“任务完成时间”来预测一个新的问题需要的修复时间.计算方法为,一个问题报告的“任务完成时间”(即问题报告被标记为“已解决”的时间)与该问题报告的创建时间之间的时间差,即为修复该问题所需的时间.Kim等人的工作就是通过这种计算方法来预测任务完成的时间.但是,Zheng等人发现,如果将把问题报告标记为“已解决”的人视为解决该问题的人,那么按照每人每天来统计工作量,则会发现有的人一天内解决了超过200个问题.据估计,每人每天解决9个问题即为上限.一人一天内解决200个问题,这大大超过了人的能力限制.因此,这明显是有问题的数据.Zheng等人发现,开发者在完成任务后可能并不会及时将问题报告标记为“已解决”,而是在之后清理问题追踪系统时,通过脚本进行批量处理.因此,“任务完成时间”并不能真正代表任务被解决的时刻.而基于问题时间数据的研究结果产生了偏差.

3.2 数据收集阶段的数据质量问题

数据收集阶段的数据质量问题主要是指,数据使用者收集到的数据与记软件开发支持工具产生的数据不完全一样,在这一节中列举的数据质量问题主要是指数据不完整.无论是通过爬虫从网站上爬取数据,还是通过API或者某些工具同步数据,或者是直接从官方链接下载整理好的数据集,收集到的数据都可能会出现不完整的问题.

3.2.1 爬取数据不完整问题

在官方没有提供数据集或者没有可用API下载数据的情况下,数据使用者们往往需要自己编写爬虫来爬取数据.这就很容易遇到爬取数据不完整的情况[9].主要是两点原因,网络问题以及反爬虫机制.网络问题主要是指网页加载慢,网络延迟或者丢包等问题.反爬虫机制是指被爬取的网站采取的防御措施.因为这两点原因,当使用爬虫爬取数据时,数据往往不完整.

3.2.2 隐私保护导致的数据不完整问题

开源社区为了保护贡献者的隐私,会采取一些列措施使得外部人员无法批量获取贡献者名单和邮箱地址.如图2所示,在Gnome社区中,如果不登录账号,则无法看到贡献者完整的邮箱地址.[4]由于无法获得完整的邮箱

[1] https://en.wikipedia.org/wiki/The_GNOME_Project

地址,只有开发者的昵称,那么就丧失了邮件地址的域名这部分信息,而这部分信息往往包含了的背景信息,例如公司,学校等.因此可能会对开发者的背景分析造成困扰.

3.2.3 安全保护导致的数据不完整问题

Zhu 等人的工作[9]提供了多个版本的 Mozilla 问题追踪数据,其中 2011 和 2012 两个版本为通过爬虫爬取的数据.2013 的版本为 Mozilla 官方提供的数据.通过三个版本的数据比较,他们发现有些问题报告 Mozilla 社区并不会开放出来,因为这些问题可能涉及到 Mozilla 的核心安全.当确认这些报告并不会产生安全隐患后,这些报告会重新开放出来.这种数据问题对于研究“安全”相关问题的人来说非常重要,因为这些曾经被隐藏起来的问题是真正涉及到安全的问题,是非常具有研究价值的对象.

3.3 数据使用阶段的数据质量问题

数据使用阶段的数据质量问题是指,数据使用者由于对数据产生的上下文不了解,而基于数据建立了错误的假设.

3.3.1 未来数据泄露问题

问题报告的属性一直在随着问题修复的进程而被不断修改.但是许多问题追踪数据的使用者并没有清楚地认识到这一点,并且没有仔细区别实验中所使用的数据是否与研究问题的应用场景所匹配.因此,可能产生在研究实验中,错误的使用了来自“未来”的信息[22].例如在 Sun 等人[13]的研究中,他们使用问题报告的标题进行重复报告检测.然而,问题报告的标题会不断修改.他们在实验中使用的问题报告的标题已经是经过反复修改的.但是重复报告检测的应用场景,主要是发生在问题报告创建之初.因此,应用问题报告创建时的原始标题来检测该报告是否重复更为合适.这种数据使用问题主要是因为没有理解“问题报告的属性一直在随着问题修复的进程而被不断修改”这一事实,忽略了数据产生的上下文,对数据建立了错误的假设.这种错误使用未来数据的问题也普遍存在于数据挖掘领域,也称为数据泄露问题,是数据挖掘的十大错误之一.

3.3.2 邮箱地址的问题

在软件数据仓库挖掘中,一个邮箱地址通常代表了一个开发者.但是,一个邮箱地址并不是单纯的对应一个开发者,而是存在“多对一”和“一对多”的关系.一个邮箱地址对应多个开发者,这种情况是指“一个邮箱地址”是公共邮箱(邮件列表地址)或者代表了特殊标识,例如,platform-help-inbox@eclipse.com 和 nobody@mozilla.org.前者可以被看作是问题报告分发过程中的 HUB,而后者则是代表了目前这个问题报告没有指派,任何人都可以尝试解决这个问题.在研究问题报告分配任务时,如果目的是将问题报告分配给具体的真实开发者,那么“多对一”将引入噪音.多个邮箱地址对应一个开发者,这是因为开发者拥有多个账户.这些账户的使用场景可能不同,例如,私人邮箱和公司邮箱;或者开发者所属单位发生了变化,即开发者跳槽了,因此拥有多个账户.在研究开发者工作效率或者开发者网络时,“一对多”将使结果产生偏差.

3.3.3 版本控制数据中关于作者与提交者的问题

在版本控制系统的提交日志中,代码提交者(committer)和代码作者(author)代表了两种不同的身份.Git 将代码提交者和代码作者分别用 committer 和 author 记录.但是 SVN 和 CVS 并没有区分这两种不同的身份.在 SVN 和 CVS 中,只存在 author 这个域,但是 author 实际记录的却是代码提交者.因此,如果按照 Git 的习惯去理解 SVN 和 CVS 的数据,以为 author 记录的是代码作者,那么就对数据有了误解,产生了错误的假设.

4 数据质量问题的发现和修正

本章节回答第二个研究问题:如何发现和解决软件开发活动数据的质量问题?

4.1 问题的发现

在前面的分析中,可以看出,有的数据质量问题很容易理解,比较容易发现,例如问题报告的创建时间错误,通过常识和统计分析及数据可视化可以发现.但有的数据质量问题比较隐蔽,例如未来数据泄露问题,需要对数据上下文有清楚的理解.

4.1.1 理解数据上下文

对于数据产生的上下文的理解是应用数据的基础.在 3.3 节中,我们反复强调了理解数据上下文的重要性.对数据理解不重复或者存在误解,都会对研究结果产生影响.Mockus[14]认为,理解数据产生的上下文应至少包含以下几点:1)数据产生的方式,例如,数据是由系统自动产生还是人工输入,是否存在缺省值;2)数据是否存在修改/删除/过滤/缺损的情况,以及这些被修改/删除/过滤/缺损的数据对结果是否有影响.

以“任务完成时间”的陷阱为例.在一般情况下,问题报告是在开发者解决了对应的问题后,手动将问题报告关闭并标记为“已解决”.但是 Zheng 等人发现,一天之内解决上百个问题报告,这已经不是正常情况下的处理流程.据他们发现,这种现象一般是由脚本批处理造成的.有三个原因导致了这种特殊的问题报告处理流程.第一,问题报告由其他系统进行管理,两边系统进行数据同步,批量关闭了报告.第二,某个时间点集中清理问题追踪系统中遗留的长期未解决的问题,将这些报告批量关闭.第三,当版本控制系统中的问题修复了,其关联的问题报告可能会被批量关闭.这三种情况下,数据都是系统自动处理,而非人工手动输入.这就是因为对数据产生的方式理解不充分而产生的数据质量问题.

4.1.2 统计分析及数据可视化

发现问题的一个有效手段是统计分析.最简单地,统计数据的平均值、中位数、最大值、最小值、四分位数,就可以对数据的分布有一个大概的认识.在这个过程中,如果有分布非常不均衡的现象,一些突出的问题已经可以被识别出来,例如,邮箱地址问题中的一对多问题.此外,还可以通过拟合一些概率分布模型来发现数据中的异常值.以“任务完成时间”的陷阱为例,Zheng 等人通过泊松分布定位出了异常的“任务完成时间”.

数据可视化是数据分析中一个有效的技术手段[20].当数据以直观的图形形式展示在分析者面前时,分析者往往能够一眼洞悉数据背后隐藏的信息并转化知识以及智慧[21].通过数据可视化,可以对数据的整体走向有一个清楚的认识.对于其中的异常点也能够轻而易举的捕捉到.例如,在揭示“任务完成时间”陷阱的过程中,Zheng 等人通过数据可视化展示出了该数据质量问题日益严重的发展趋势.另外,在可视化问题追踪数据的过程中找到了 1970 年这个异常点,因此发现了问题报告的创建时间错误.

4.2 问题的修正

在发现了软件开发活动数据的质量问题后,可以尝试通过两种方法对问题数据进行修正来降低数据质量问题带来的负面影响,即,利用冗余数据进行修正,和挖掘用户行为模式进行修正.所幸,丰富的软件开发活动数据资源使得这两种数据修正方式成为可能.这两种数据修正方法分别适用于不同的研究问题和背景.利用冗余数据进行修正的方法能够很好地适用于时间存在问题的数据,例如“任务完成时间”陷阱.挖掘用户行为模式进行修正适用于需要进行用户身份识别的问题,例如邮箱地址的问题.数据问题的修正方法与研究问题和背景密切相关,仔细分辨研究问题的动机、背景,了解数据产生的上下文,不仅可以发现问题,也是修正数据必不可少的步骤.

4.2.1 利用冗余数据进行修正

冗余数据是指,软件开发活动数据中与原数据用途、含义相近的数据.在软件开发活动数据中,一项开发活动可能在多个不同的系统或者相近的活动中留下痕迹.这些就是互为冗余的数据.特别强调,这里的冗余并非贬义词,而是指数据的意义相近(并非完全相同需要去除的冗余数据).

以“任务完成时间”的陷阱为例.冗余数据可能存在于同一个系统中,也可能存在于不同的系统中.“任务完成时间”的陷阱中,有问题的数据是问题报告被标记为“已解决”的时间.这个数据原本是想表示该问题被解决的时间.但是当它出现了问题,它就并不能起到应有的作用了.因此,从这个数据原本表示的含义出发,结合“已解决”时间产生的上下文,即,与该数据相关的软件开发活动,分别在同一个系统和不同的系统中寻找意思相近的数据.在同一个系统中,即问题追踪系统中,每一个问题报告的最后一条评论时间基本可以等视为问题被解决的时间.因为当一个问题被解决后,就很少会有人再去关注它,讨论它.那么就可以将原先错误的“任务完成时间”修改为更为准确的问题报告最后一条评论时间.在不同的系统当中,例如版本控制系统,与该问题报告对应的代码的提交时间可以等视为问题为解决的时间.因为通常情况下的处理流程就是,在代码库中提交了代码修改,然后

将问题报告标记为“已解决”.这些都是从数据的含义出发,理解了开发实践过程,找到的解决方法.

这种方法也可以应用到问题报告的创建时间的修正上,例如,可以用问题报告的第一条评论的时间或者第一次信息被修改的时候来预估问题报告的创建时间,利用冗余数据进行修正的方法在修正软件开发活动的时间上具有一定的通用性,因为软件开发活动是一系列连贯的活动,根据该项活动前后的活动数据,就可以对原来错误的数据进行修正.

4.2.2 挖掘用户行为模式进行修正

软件开发活动数据记录了软件开发过程中开发者的一系列活动.软件开发是一项连续的活动,每一个行为或活动不可能单独存在,也不会突然发生巨大的改变.我们假定一个用户的行为模式是特定的,那么我们可以从历史数据中挖掘出用户的行为模式,进而对问题数据进行修正.

这里以邮箱地址的问题为例.在解决“多对一”的问题时,除了根据用户名的命名规则和相似度进行识别,还可以通过挖掘用户行为模式进行修正.例如,在版本控制数据中,这些邮件地址的作者或者代码提交者如果有着相近或者相同的行为习惯,那么这些邮件地址就很有可能指向同一个人.在这个问题中,行为习惯包括,编写或者提交过的文件,提交日志的书写风格和工作时区.如果多个不同的账户修改同样的代码文件,那他们可能指向同一个人.不同的开发者有着不同的代码风格,也有着不同的代码日志风格,如果多个账户的代码提交日志的书写风格类似,那么他们可能指向同一个人.假定开发者的物理位置不会频繁发生改变,那么如果多个,例如,有相同书写风格的账户的物理位置相近,他们就更可能指向同一个人,而在版本控制数据中,该信息可以从时区信息中提取出来.

同样地,在识别“一对多”问题时,也同样可以通过挖掘用户行为模式定位出那些公共账号,例如工作时区.在开源开发中,开发者遍布世界各地.当一个邮件地址对应的时区信息经常发生变化,具有较大不确定性,那么该账号就很有可能是一个公共账号.

在发现数据问题后,首先应明确研究问题的动机、背景,结合数据产生的上下文,确定是否存在冗余数据,以及能否使用冗余数据进行修复.如果数据问题涉及开发者身份识别，则可以尝试通过挖掘用户行为模式来修正数据.

5 总结与展望

高质量的数据是分析、预测、推荐的基础.随着越来越多的工作围绕软件活动开发数据展开,也逐渐有一些工作认识到数据质量的重要性并发现了数据中潜在的数据质量问题.然而人们对于软件开发活动数据质量问题的关注度还是远远不够.因此,本文通过文献调研和访谈的形式,总结出了 9 点软件开发活动数据中可能的数据质量问题.并从发现问题和解决问题方面分别提出了建议.

我们注意到有越来越多的研究者关注到了数据质量的问题,相信随着智能化软件开发的蓬勃发展,数据质量问题会得到巨大的关注.同时希望在解决了数据质量问题后,智能软件领域可以像其他智能领域(例如计算机视觉)一样,构建出可靠的受认可的数据集.

本文的工作未来将在以下几个方面继续探索:

1. 从数据使用者们的习惯出发,总结出数据使用者们能够普遍注意到的数据质量问题,以及经常被忽视的问题.虽然文章按照数据质量问题产生的不同阶段进行了归纳,但是并没有总结出这些数据质量问会影响到哪些类型的相关工作.因此,将加入这部分内容使得文章的结果更具有参考价值.

2. 挖掘更多修正数据的方法和例子.文中提出了用冗余数据和挖掘用户行为模式进行问题数据的修正.但是这两种方法的适用范围有限.因此,接下来将深入调研,挖掘其他可能的数据修正方法.

3. 探究其他智能化软件开发过程中常用数据的质量问题.本文总结的数据质量问题都是与版本控制数据和问题追踪数据相关.为了更好的服务于智能化软件开发,将探索其他常用数据的数据质量问题,例如 Stack Overflow 问答数据.

References:

[1] Zhu J, Zhou M, Mockus A. Effectiveness of code contribution: From patch-based to pull-request-based tools[C]//Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2016: 871-882.

[2] Herzig K, Just S, Zeller A. It's not a bug, it's a feature: how misclassification impacts bug prediction[C]//Proceedings of the 2013 international conference on software engineering. IEEE Press, 2013: 392-401.

[3] Fritz T, Shepherd D C, Kevic K, et al. Developers' code context models for change tasks[C]//Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. ACM, 2014: 7-18.

[4] Zhou M, Mockus A. Developer fluency: Achieving true mastery in software projects[C]//Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering. ACM, 209: 137-146.

[5] Stamelos I, Angelis L, Oikonomou A, et al. Code quality analysis in open source software development [J]. Information Systems Journal, 2002, 12(1): 43-60.

[6] Kim S, Whitehead Jr E J. How long did it take to fix bugs?[C]//Proceedings of the 2006 international workshop on Mining software repositories. ACM, 2006: 173-174.

[7] Zheng Q, Mockus A, Zhou M. A method to identify and correct problematic software activity data: Exploiting capacity constraints and data redundancies[C]//Proceedings of the 2015 9th Joint Meeting on Foundations of Software Engineering. ACM, 2015: 637-648.

[8] Weiss C, Premraj R, Zimmermann T, et al. How long will it take to fix this bug?[C]//Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on. IEEE, 2007: 1-1.

[9] Zhu J, Zhou M, Mei H. Multi-extract and multi-level dataset of mozilla issue tracking history[C]//Proceedings of the 13th International Conference on Mining Software Repositories. ACM, 2016: 472-475.

[10] Zhou M, Mockus A. What make long term contributors: Willingness and opportunity in OSS community[C]//Proceedings of the 34th International Conference on Software Engineering. IEEE Press, 2012: 518-528.

[11] Bird C, Rigby P C, Barr E T, et al. The promises and perils of mining git[C]//Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on. IEEE, 2009: 1-9.

[12] Izquierdo-Cortazar D, Robles G, Ortega F, et al. Using software archaeology to measure knowledge loss in software projects due to developer turnover[C]//System Sciences, 2009. HICSS'09. 42nd Hawaii International Conference on. IEEE, 2009: 1-9.

[13] Sun C, Lo D, Khoo S C, et al. Towards more accurate retrieval of duplicate bug reports[C]//Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society, 2011: 253-262.

[14] Mockus A. Engineering big data solutions[C]//Proceedings of the on Future of Software Engineering. ACM, 2014: 85-99.

[15] Giger E, Pinzger M, Gall H. Predicting the fix time of bugs[C]//Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering. ACM, 209: 52-56.

[16] Bhattacharya P, Neamtiu I. Bug-fix time prediction models: can we do better?[C]//Proceedings of the 8th Working Conference on Mining Software Repositories. ACM, 2011: 207-29.

[17] Negara S, Vakilian M, Chen N, et al. Is it dangerous to use version control histories to study source code evolution?[C]//European Conference on Object-Oriented Programming. Springer, Berlin, Heidelberg, 2012: 79-93.

[18] Bird C, Bachmann A, Aune E, et al. Fair and balanced?: bias in bug-fix datasets[C]//Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. ACM, 2009: 121-130.

[19] Tamrawi A, Nguyen T T, Al-Kofahi J, et al. Fuzzy set-based automatic bug triaging (NIER track)[C]//Proceedings of the 33rd International Conference on Software Engineering. ACM, 2011: 884-887.

[20] Beyer D, Hassan A E. Animated visualization of software history using evolution storyboards[C]//Reverse Engineering, 2006. WCRE'06. 13th Working Conference on. IEEE, 2006: 199-29.

[21] 任磊, 杜一, 马帅, 等. 大数据可视分析综述[J]. 软件学报, 2014, 25(9).

[22] Tu F, Zhu J, Zheng Q, et al. Be careful of when: an empirical study of time-related misuse of issue tracking data[C]// /Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE). ACM, 2018