软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2019,30(7):2109−2123 [doi: 10.13328/j.cnki.jos.005489]

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

软件开发活动数据集的层次化、多版本化方法*

朱家鑫[1,2,3], 周明辉[1,2]

[1](北京大学 信息科学技术学院 软件研究所,北京 100871)

[2](高可信软件技术教育部重点实验室(北京大学),北京 100871)

[3](中国科学院 软件研究所 软件工程技术研究开发中心,北京 100190)

通讯作者: 周明辉, E-mail: zhmh@pku.edu.cn

摘 要: 随着开源软件的兴起及软件开发支撑工具的普及,Internet 上积累了大量开放的软件开发活动数据,越来越多的实践者与研究者尝试从中获取提高软件开发效率和产品质量的洞察.为了提高数据分析的效率、方便分析结果的重现与对比,许多工作提出了构建与使用共享数据集.然而,现有软件开发活动数据集的构建过程可追溯性差、适用范围窄,对数据随时间、环境发生的变化欠考虑.这些不足直接威胁数据的质量及分析结果的有效性.针对该问题,提出一种层次化、多版本化的方法来构建与使用软件开发活动数据集.层次化是指在数据集中包括收集和后续处理所得的原始、中间和最终数据,建立数据集的可追溯性并扩展其适用范围.多版本化是指通过多种方式进行多次数据收集,使数据使用者能够观察到数据的变化,为数据质量及分析结果有效性的验证和提高创造条件.通过基于该方法构建的 Mozilla 问题追踪数据集进行示范,并验证了该方法能够帮助数据使用者高效地使用数据.

关键词: 数据驱动的软件工程;软件开发活动数据;数据分析;数据质量;数据集

中图法分类号: TP311

中文引用格式: 朱家鑫,周明辉.软件开发活动数据集的层次化、多版本化方法.软件学报,2019,30(7):2109−2123. http://www.jos.org.cn/1000-9825/5489.htm

英文引用格式: Zhu JX, Zhou MH. Multi-level and multi-version approach for software development dataset. Ruan Jian Xue Bao/ Journal of Software, 2019,30(7):2109−2123 (in Chinese). http://www.jos.org.cn/1000-9825/5489.htm

Multi-level and Multi-version Approach for Software Development Dataset

ZHU Jia-Xin[1,2,3], ZHOU Ming-Hui[1,2]

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

[2](Key Laboratory of High Confidence Software Technologies of Ministry of Education (Peking University), Beijing 100871, China)

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

Abstract: With the fast development of open source software and wide application of development supporting tools, there have been a great many of open software development data on the Internet. To improve the software development efficiency and product quality, more and more practitioners and researchers attempt to obtain insights of software development from the data. To facilitate the data analyses and their reproduction and comparison, building and using shared datasets are proposed and practiced. However, the existing datasets are lack of traceability of dataset construction process, application scope, and consideration of data variation over time and with environment changes, which threat the data quality and analysis validity. To address these problems, an advanced approach is proposed for sharing and using the software development datasets. It constructs datasets with multiple levels and multiple versions. Through multiple levels, the datasets remain the raw data, intermediate data, and final data to possess data traceability. Meanwhile, by multiple versions, users can compare and observe the data variety to verify and improve data quality and analysis validity. Based on the previously constructed

* 基金项目: 国家重点研发计划(2018YFB1004201); 国家自然科学基金(61432001, 61825201)

Foundation item: National Key R&D Program of China (2018YFB1004201); National Natural Science Foundation of China (61432001, 61825201)

收稿时间: 2017-01-13; 修改时间: 2017-06-27, 2017-09-17; 采用时间: 2017-11-07

Mozilla issue tracking dataset, it is demonstrated that how to build and use multi-level and multi-version software development dataset and verified that the proposed approach can help users efficiently use the dataset.

Key words: data-driven software engineering; software development data; data analysis; data quality; dataset

随着软件开发支撑技术的发展,相关工具在软件开发生命期中的覆盖度以及在软件项目中的普及程度越来越高.版本控制系统、问题(issue)追踪系统、邮件列表是在开发中最为常见的支撑工具.它们的数据库中(软件仓库)记录了软件的新功能开发、缺陷修复、代码审查等多种活动.随着开源运动的不断发展壮大,开放的软件开发活动数据在不断地累积并达到了非常可观的规模[1].通过分析这些数据,人们可以获取对开发活动更为准确和深刻的认识,进而有效地改进软件开发实践[2−4].

对软件开发活动数据的分析通常需要人们投入大量的时间与精力收集、清洗与组织数据[5].在过去,大多数研究工作都是独立地进行数据准备工作,而这种方式非常低效:一方面,很多研究问题所需要的数据是相同的,因而整个研究社区中会出现大量重复性的工作;另一方面,基于不同数据集的分析结果在重现和对比时容易出现数据集缺失(例如分析者没有公开所研究的数据集、数据集的链接失效等)、数据格式不统一、数据集差异引入混杂因素等问题.因此,很多工作尝试着去构建和共享公共的数据集来解决这些问题[5−9].

虽然有越来越多的研究者与实践者关注了软件开发活动数据的共享,但这些工作的关注点主要在于数据的公开和推广,关于数据可用性[10−13]的考虑还存在较大不足.首先,现有的共享数据集有原始数据和定制数据这两种形式.共享原始数据的做法减少了使用者在数据收集方面的成本,共享定制数据可以进一步减少数据预处理的成本.然而,大多数定制数据集的构建过程可追溯性差,即数据收集与处理的过程不清晰,缺少中间结果,不利于数据质量的验证以及分析异常的排查,适用范围也相对有限.其次,现有数据集往往是通过一次收集或者若干次增量收集而得到的,并没有考虑到数据收集环境的变化以及软件开发活动的变化,这些变化对数据可用性的影响也就无法得到揭示和处理.针对上述问题,本文提出一种层次化、多版本化的数据集构建及使用方法.首先,该方法提出根据数据处理的不同程度划分不同的数据层次,使数据集包含原始、中间和最终多个层次的数据,从而建立数据集构建过程的可追溯性,并兼顾应用时的普适性;其次,该方法提出使用不同的方式在不同的时间段进行多次数据收集,形成数据集的多个版本,尽可能地将数据变化纳入其中,为数据质量及分析结果有效性的验证和提高创造条件.基于我们构建的 Mozilla 问题追踪数据集[14],本文展示了这种改进的数据集的构建和使用方法,并验证了其效果.

本文第 1 节介绍软件开发活动数据集的构建与共享现状,揭示其中的不足,并有针对性地提出层次化、多版本化的数据构建与使用方法.第 2 节详细阐述该方法.第 3 节给出一个具体的数据集构建实例.第 4 节展示基于该实例的 5 个应用示例.第 5 节对全文进行总结.

1 软件开发活动数据集的构建与共享现状

利用软件开发支撑工具所记录的数据来支持软件开发实践的改进是当前软件工程研究的一大趋势[3].这种数据驱动的软件开发活动分析主要包括了图 1 所示的一系列步骤.首先,从目标项目所使用的开发支撑工具的数据库,即软件仓库中收集原始数据;其次,对这些原始数据进行清洗,去除无关数据、噪音数据;接下来,对这些预处理后的数据进行适当的组织及存储,方便后续的使用;在具体的研究中,分析者会根据研究问题建立相关的量度,从数据中得到度量结果并进行分析;最后一步是分析结果的验证与应用.可以看到,在进行分析之前,研究者要进行数据的收集、清洗、组织、存储,而这部分工作的成本可能非常高昂,特别是面向大规模数据的时候[15].需要特别注意的是,这种成本包括了以下两个方面:首先,数据的可得性由项目的数据开放政策决定,有些项目的数据需要与业务繁忙的项目管理者进行漫长的交涉,最终签署协议才能获得,即便是公开的数据,由于要保障开发活动的正常进行,对开发支撑工具高频率的数据获取请求也是被禁止的,因此数据收集需要大量额外的时间开销;其次,分析者个人的网络条件也可能造成一些困难,例如网络带宽、网络访问的限制等,这些因素也会增大数据收集的复杂性.除了数据的收集,大量异构原始数据的清洗、组织也需要较大的人力和数据存储成本.因此,很多研究者提出通过数据共享来提高数据分析的效率:减少每一次分析的数据准备成本,减少整个研

究社区中的重复数据准备工作,方便数据分析结果的重现与对比.

目前,研究社区中已经有了相当多构建与共享数据集的工作,它们主要包括两类:一类是构建与发布特定的数据集,另一类是建设数据集共享的平台.挖掘软件仓库的代表性会议 Int'l Conf. on Mining Software Repositories(MSR)[4]设有 Data Showcase 专区,其中发表的大多数数据集是其作者在各自的研究中所构建的.这些数据集中有的将软件项目的开发历史数据进行整合,例如 Spinellis 将 UNIX 系统 44 年的代码演化历史整合到一个 Git 仓库中,通过 Git 命令人们可以查询到过去 UNIX 代码发生的改变[16];有的则对某些项目托管网站的数据进行大规模收集,为研究者提供数量众多的软件项目的多种开发活动数据,包括版本控制数据、问题追踪数据等,例如 Gousios 领导的收集、共享 GitHub 平台上众多开源项目开发数据的 GHtorrent 项目[17],还有一些是从软件仓库中提取与加工面向特定问题的数据集,例如 Amann 等人基于 Sourceforge 和 GitHub 上众多版本控制库构建的用于测试 API 误用检测器性能的标杆数据集 MUBench[18].数据集共享平台建设的典型项目有 FLOSSmole[5]和 PROMISE[6],旨在为研究者们提供一个协作化的数据共享平台,人们可以贡献与获取符合一定规范的各类软件开发活动数据,并分享自己基于这些数据所得到的研究成果.而 MSR 会议的 Mining Challenge 专区为研究者提供了一个数据分析竞赛平台,研究者基于一个公共的数据集来开展各种分析并一较高下.为了促进跨软件仓库挖掘的研究,Keivanloo 等人又在平台的建设上提出了将各类开发活动数据进行连接[19],他们设计了一种软件开发活动数据模型,该模型中包含了一系列软件开发所涉及到的对象,例如项目、代码、缺陷、开发者、代码提交等,从版本控制、问题追踪等软件仓库中提取出这些对象并建立它们之间的关系,如开发者 *A* 执行了代码提交 *B*,代码提交 *B* 修复了缺陷 *C*.基于该模型,开发者们搭建了 SeCold 平台来共享有连接的软件开发活动数据集.

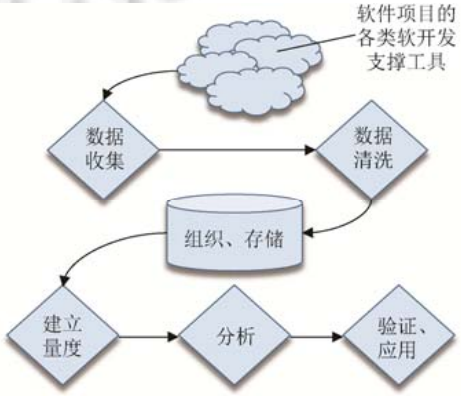

Fig.1 The process of software development data analysis

图 1 软件开发活动数据分析流程

上述工作的关注点主要在于促进数据的公开和传播,而数据共享中的数据可用性问题还有待进一步研究.数据可用性被关注最多的方面是数据质量[20],研究者们提出了数据的时效性、一致性、精确性、完整性、代表性等多个方面[21−26]的属性来刻画数据质量.从应用的角度来讲,数据质量决定了上层分析结果的有效性,例如在预测缺陷修复时间时,数据中标定的缺陷修复完成时间的精确性非常关键,错误的时间记录会使得预测模型无法得到有效的训练而给出错误的结果[27].再如,在训练缺陷预测模型时,训练数据中若包含非缺陷相关的噪音数据,则模型的性能会出现一定的下降[28].数据可用性还有另一个关注较少的方面——数据的适用范围,即一个数据集能够帮助解决多少问题,例如基于某个项目的版本控制数据定制的一个关于缺陷修复活动的数据集可以用来度量该项目的缺陷密度、项目中开发者修复缺陷的经验等,但因为它只包含了缺陷修复的活动而无法用来度量版本控制数据中记录的其他开发活动,如实现软件的新功能.

之前的数据集构建工作在数据可用性的保障上都或多或少地存在一些不足.这些不足主要表现为两点:一是数据集缺乏构建过程的可追溯性,即从原始数据到最终数据的整个数据处理过程没有详尽而准确的描述,中

间结果缺失;二是对数据收集与软件开发活动的改变所引发的数据变化欠考虑.

可追溯性决定数据使用者能否处理分析中遇到的异常、验证数据质量、解决更多的软件开发问题.有些数据共享工作提供所收集的原始数据,例如 MSR07 的 Mining Challenge 专区提供的问题追踪数据集[29],而更多的工作则提供经过清洗、转换等处理后定制化的数据,例如 Habayeb 等人的 Firefox Temporal Defect Dataset (FTDD)[30],Gousios 等人的 GHtorrent MySQL database dump 数据集[17].相对于原始数据,分享定制化的数据可以降低数据预处理的成本,然而,这些数据集大都以黑盒式的方式提供,原始数据以及中间的处理过程、中间数据都较为模糊或者完全缺失,可追溯性较差,由于缺少中间的过程及数据,数据质量验证以及分析异常的处理,这些需要追溯数据处理过程的工作难以开展,定制过程中被移除的开发活动信息也无法得到恢复,能够用以解决的软件开发问题只能是特定的.

对数据变化的考虑程度决定了数据分析的结果是否有效以及在什么样的条件下有效.当前大多数数据共享工作并没有对可能的数据变化予以考虑,共享的数据集大都是一次收集或者若干次增量式收集所得到的,由于缺少参照,数据收集环境的变化以及软件开发活动的变化无法被揭示,其中,数据收集环境的变化可能引起数据质量的改变,例如访问受限、网络连接中断、脚本缺陷等会导致数据的缺失和出错,软件开发活动的变化可能导致原有数据分析结果不再有效,如 Ioannidis 指出很多已有数据分析结果在新的数据集上无法得到重现[31].

为帮助解决上述问题,本文提出一种改进的数据构建及使用方法,通过层次化使数据集的构建过程可追溯,使数据集的适用范围可伸缩,通过多版本化来捕获数据的动态变化.在下一节中我们分别对这两种设计进行详细的介绍.

2 数据集的层次化与版本化

2.1 数据集的层次化

本文所提出的“层次化”概念中的“层次”指的是依据对数据进行预处理的程度而划分的不同数据层次.考虑多个数据层次既可以降低数据收集与预处理的成本,又可以通过保留全部业务信息来兼顾数据的适用范围,与此同时,数据集构建过程的追溯性也得到了建立,保障了数据质量与数据分析结果的有效性.

如图 2 所示,层次化的数据集包含两个基本的层次:原始数据层(层次 0)和标准化数据层(层次 1).层次 0 中的数据是从开发支撑工具中所收集到的未经其他处理的数据,这些数据保留了数据共享者在数据收集时能够获取的全部开发活动信息,并且不包含数据预处理所可能引入的偏倚.但是,层次 0 中数据的数据模式由支撑工具所定义,在使用前需要进行特定的解析来获取相关的信息,例如从版本控制系统的库中获取提交信息需要使用系统提供的接口(命令行或图形化用户接口、编程接口)来进行查询,从问题追踪系统中获取问题追踪信息需要对 Web 页面进行解析.此外,原始数据中可能包含大量的与开发活动无关的数据及冗余数据(如 Web 页面中用来布局和美化页面的标签等),增加不必要的数据传播和存储开销.因此,对原始数据进行数据提取和标准化十分必要.本文设计了层次 1 来加入标准化数据.标准化数据包含了从原始数据中提取出的全部开发活动相关数据,并且以方便后续使用的数据格式来组织,例如解析简便、以纯文本方式存储的 CSV(comma-separated values)格式[32].在层次 1 之上,数据共享者还可以增加面向特定研究问题的层次 2+的数据.这些较高层次中的数据经由对层次 1 中数据的进一步处理而得到.这些处理与研究问题相关,可以是特定开发活动信息的提取、数据的拼接、数据的转换等,例如从邮件列表的数据中只提取出那些关于代码审查的会话,构建代码审查数据集[33];将一个缺陷的报告与解决该缺陷的代码提交进行连接[34];将问题报告的解决活动进行抽象、编码[30].

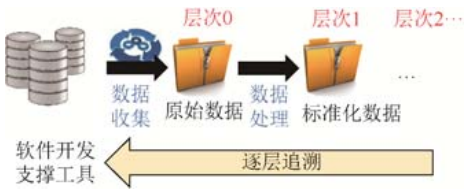

Fig.2 The multi-level dataset

图 2 层次化的数据集

使用层次化数据集的第 1 步是选取适当层次的数据.一般而言,当确定要使用某个软件开发支撑工具所记录的数据时,任何分析都可以使用相应数据集中层次 1 的数据.这是因为层次 1 的数据包含了目标工具中记录

的全部开发活动信息,能够满足不同问题的数据需求.虽然使用层次 0 的数据也可以达到同样的目的,但使用层次 1 的数据既可以减小数据下载及存储的代价,又不需要进行数据的提取及标准化.如果要重现某项研究的结果,或者是进行相似问题的研究,那么可以考虑直接使用更高层次的数据,省去相关的数据提取、拼接甚至度量的工作.例如,如果要研究问题报告(issue report)解决过程中的活动模式,则可以直接使用 Habayeb 等人所构建的 FTDD 数据集[30]而不是 MSR07 Mining Challenge 的数据集[29],该数据集已对原始的问题追踪数据进行了数据标准化之外的深层处理,其中的数据相当于本文介绍的层次 2 或层次 2+中的数据.利用这样的数据集可以免去对问题处理活动与问题报告属性之间的拼接(在原始数据中这两类数据是分别保存的)、对问题处理活动的编码等.但是,该数据集并不是层次化的,其所包含的问题追踪信息可能无法满足其他研究的需求.例如,该数据集在对活动编码时过滤了问题报告状态的变迁信息,对于需要考虑该信息的研究,FTDD 显然是不适用的.这个时候就需要回到层次 1,基于层次 1 中的数据重新构建更高层次的数据.

层次化数据集的可追溯性对于提高数据分析结果的有效性非常有帮助.在使用高层次的数据进行分析时,数据使用者如果遇到异常,可以回退到较低层次来排查异常产生的原因.这些异常可能来自于软件开发活动本身,也可能来自数据收集以及后续处理的过程.随着数据层次的提高,原始数据被处理的程度在不断加深,其间出现差错的概率也会随之增大.利用层次化的数据集,使用者可以以逐层回退的方式查找异常的源头.如果异常仅出现在某个层次以上的数据层中,那么它是由相关数据收集或处理脚本的缺陷所引起的.此时,要对错误进行修正并重新构建有问题的数据层.如果支撑工具的数据库中也存在该异常,那么它很可能由软件开发活动中的某些特殊情况所引发.此时,要对该异常作进一步的探查及解释,并在数据处理中根据具体情况对异常数据进行分离、过滤或者保留.

2.2 数据集的多版本化

本文提出的“多版本”概念中的“版本”指的是通过不同的方式或是在不同的时间所收集到的各个数据快照.包含多个版本的数据可以将数据收集及软件开发中发生的各种变化纳入到数据集中,为提高数据的质量及分析结果有效性建立基础.

如图 3 所示,多版本的数据要以多种方式在多个时间段以不同的方式进行数据收集.数据的收集一般有两类方式:一类是通过开发支撑工具的交互接口间接地获取数据,另一类是直接导出开发支撑工具后台数据库中的数据.对于前者,工具的交互接口又可以分为图形化的或命令行的人机交互界面,以及应用程序编程接口(API),例如 GitHub API[7];对于后者,数据库的类型可能是普通的关系型数据库,如 Bugzilla 的 MySQL 关系型数据库,也可能是工具自定义的数据库,如 Git 基于对象的有向无环图数据库.在这些收集方式中进行切换、对比可以揭示不同方式对原始数据所产生的影响,进而建立评估每种方式所得数据的质量的基础.间隔一定的时间,例如 1 年,进行多次数据收集则可以纳入我们在第 1 节中提到的数据变化,这些变化可以分为两类:一类是数据收集环境的变化,另一类是软件开发活动的变化.数据收集环境的变化可能导致数据收集过程中异常情况的发生,例如数据源访问受限导致部分信息无法获取.这些异常情况将使收集到的数据发生残缺或错误.开发活动的变化会改变相应的数据,可能使其具有不同的特点和性质.

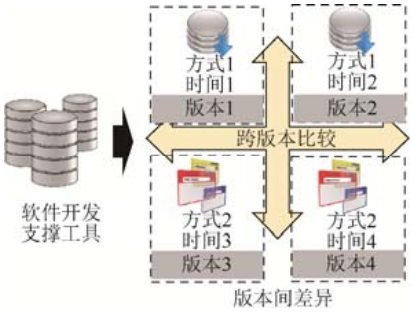

Fig.3 The multi-version dataset

图 3 多版本化的数据集

在进行数据分析时,数据使用者需要充分考虑上述数据收集方式的不同及数据本身的变化,分析它们是否反映了数据质量问题,对数据分析结果的有效性存在怎样的影响.使用多版本数据的核心思想是对不同版本数据进行交叉对比,实现数据质量的验证、开发活动变化的分析、数据分析有效性的评估和提高.

数据质量的验证在单一版本的数据集上相对困难,数据使用者需要对软件开发活动本身有深入的理解、拥有分析相关数据的丰富经验[2]才能发现其中存在的质量问题.而利用多版本进行对比则容易得多.具体来说,数据使用者首先找出版本间的不同,然后探究这些不同是否反映了数据质量问题.例如在第 4.2.2 节中提到的不完

整的用户邮件地址显然属于数据质量问题.该问题可以利用多版本中的冗余数据(其他版本中完整的地址)进行修复.

软件的开发活动在不断推进演化,多版本间的不同也包含了开发活动变化.我们可以借助该特点拓展研究问题的范围,探究新的有价值的问题.例如,探索识别敏感问题报告的技术、解决敏感问题报告的最佳实践.

当在单一的数据集上进行数据分析时,分析者对分析结果的有效性缺乏足够的认识,无法对分析过程及分析模型进行改进.而多版本的数据集中纳入了多种实际发生的开发活动的变化.该特点可以帮助评估和提高数据分析有效性.例如,在开源项目长期贡献者形成的研究[35]中,我们使用了 3 个版本(这里简称版本 A、B 和 C)的 Mozilla 问题追踪数据.版本 A 与 B 是通过爬取 Bugzilla 的 Web 页面所得到的,版本 C 是后来取得的官方的 Bugzilla 数据库 dump.借助于这 3 个版本的数据集,我们对研究结果的有效性进行了两项验证,在第 1 项验证中,我们使用版本 A 拟合(训练)模型,使用版本 B 测试模型;在第 2 项验证中,我们使用版本 C 重现前面得到的模型.通过这种方式,我们在更大程度上保障了研究结果的有效性.

3 数据集构建示例

根据上一节提出的方法,我们构建并发布了一个 Mozilla 问题追踪数据集(该数据集已发表于 MSR 2016 的 Data Showcase Track[9].本文系统化地阐述了构建此类数据集的方法论,并基于该数据集做了多项具体应用),其访问地址为 https://github.com/jxshin/mzdata.Mozilla 是一个具有 20 多年历史的大型开源软件项目,它旗下有我们耳熟能详的 Firefox 浏览器、Thunderbird 邮件客户端等产品.该项目使用了 Bugzilla 问题追踪系统[36]来管理各类开发问题,例如软件缺陷、新功能请求(它们被总称为“问题”)的报告及解决.在 Bugzilla 中,问题报告的报告时间、报告者、处理进度、处理结果等信息被跟踪记录,关于这些信息的细节,第 3.1 节有详细的介绍.目前,已有非常多软件工程的研究使用了问题追踪数据,所研究的问题包括缺陷的预测[37]、定位[38]、分类[39]和修复[40],开源代码的贡献的机制[41]等,其所具有的价值可见一斑.因此,我们利用上一节提出的方法构建具有更高可用性的 Mozilla 问题追踪数据集来支撑相关研究.下面两个小节分别从层次化和多版本化两个方面详细介绍了该数据集.

3.1 层次化的Mozilla问题追踪数据集

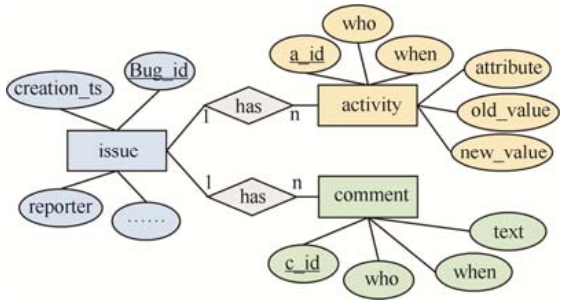

Fig.4 The data model of Mozilla issue tracking dataset

图 4 Mozilla 问题追踪数据模型

该数据集包含了层次 0 和层次 1 两个基本数据层.其中,层次 0 是从 Bugzilla 所获取的原始数据,对于不同版本的数据(详见第 3.2 节),原始数据的形式会有所区别,在某些版本中,层次 0 的数据是通过 Bugzilla 的用户 Web 界面所获取的 Web 页面,而在其他的版本中,这些原始数据是从 Bugzilla 后台数据库中导出的脱敏 dump.如图 4 所示,这两种形式的原始数据包含了相同类型的信息,包括每个问题报告的报告者、报告时间、严重程度、状态等属性,某些属性(如问题状态)的变更历史以及关于该问题报告的评论、评论者.在 Bugzilla 的 Web 界面中,这些信息由两类页面所记录,一类是关于问题报告基本信息的页面(HTML 或 XML 格式),包括问题报告的当前属性和相关评论,另一类是部分问题报告属性变更记录的页面(HTML 格式);而在 Bugzilla 的后台数据库中,每一类问题追踪信息记录于相应的数据表中.关于这些原始数据的数据模式,XML 及 HTML 文件中的标签的 name 说明了每个域的含义,而数据库中有一张叫作 fielddef 的表,表中包含了各个域的含义.

从 Web 界面获得的层次 0 数据除了包含开发中人们报告和解决问题的活动记录,还包含了大量的 HTML 及 XML 标签,这些标记使得数据产生较大的膨胀,不利于数据的传播及存储.此外,为了从 Web 页面中分离出问题追踪的业务数据,还需要对这些页面进行解析.因此,我们进一步构建了层次 1 的标准化数据,将层次 0 中的

XML、HTML 页面、数据库 dump 标准化.这种标准化处理仅仅是数据格式的转换,所有层次 0 中的问题追踪信息都得到了保留,但层次 1 数据的体量有了大幅度的减小.层次 1 的数据采用了 CSV 格式,数据的解析与提取相比于原始的 Web 页面要容易得多.该层数据分为问题报告基本信息与问题报告处理活动历史两部分,图 5 描述了基本信息(a)与活动历史(b)的数据模式,每一项基本信息由 record_id 唯一标识,每一条评论域的命名方式为“long:⟨评论 id(此条记录内)⟩:⟨评论属性名⟩”,每一个问题属性域由对应的属性命名,“=”后接各个域的值;每一项活动历史同样由 record_id 唯一标识,每个域的命名方式为“⟨活动 id⟩:*x*”,其中,*x* 编码了活动的属性,0 为活动执行者,1 为执行时间,2 为被修改的问题属性,3 为属性原值,4 为属性新值,Bug 表示对应问题报告的 id,“=”后接各个域的值.

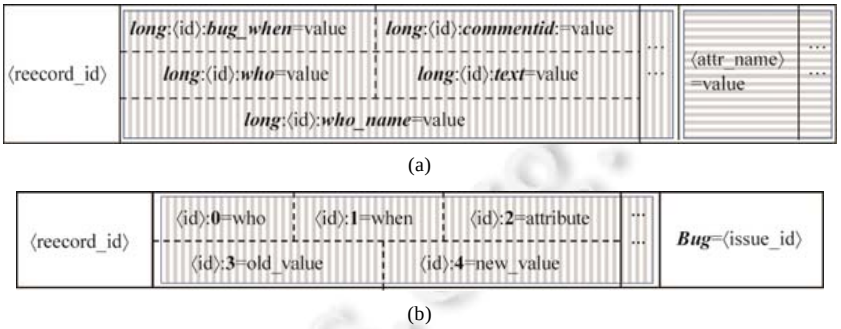

Fig.5 The data schema of Mozilla issue tracking dataset Level 1

图 5 Mozilla 问题追踪数据集层次 1 的数据模式

在层次 1 之上,数据使用者可以对数据进行提取、组合、计算等处理,构建面向某些特定研究问题的定制化问题追踪数据,例如第 4.2.1 节中所介绍的示例.

3.2 多版本化的Mozilla问题追踪数据集

目前,我们的 Mozilla 问题追踪数据集共有 4 个版本,每个版本最直观的区别体现在原始数据获取的时间及方式的不同上.首先,这 4 个版本的原始数据分别收集于 2011 年、2012 年、2013 年和 2016 年,为了方便阐述,我们以这 4 个年份来命名这 4 个版本的数据;其次,这 4 次数据收集使用了两种不同的方法:一种是目前研究社区中最常使用的基于爬虫的方法,我们批量地下载 Bugzilla 的用户 Web 页面;而另一种方法则与众不同,我们通过积极地与 Mozilla 社区沟通,获得了的经过脱敏处理的 Bugzilla 后台数据库 dump.我们使用前者收集了 2011 年与 2012 年的数据,而通过后者收集了后两个版本的数据.相对于 Web 页面,数据库 dump 的下载要容易很多,它既不会干扰社区的正常工作,也不会受到访问频率及下载速度的限制以及爬虫自身 bug 的影响.表 1 是这 4 个版本数据的概况,从中我们可以看到,它们在问题报告数量、参与者人数等各个方面都存在明显的不同.

Table 1 Summary of the Mozilla issue tracking dataset

表 1 Mozilla 问题追踪数据集的概况

版本	2011	2012	2013	2016
时间跨度	1998.4~2011.3	1998.4~2012.4	1998.4~2013.1	1998.4~2016.3
问题报告数量	620K	709K	775K	1.161M
报告者数量	143K	153K	159K	186K
讨论者数量	5.2M	6.0M	6.7M	10.5M
讨论者人数	179K	192K	200K	238K
属性变更次数	6.8M	8.0M	9.5M	15.3M
属性变更操作者人数	84K	94K	102K	140K
总参与人数	186K	200K	209K	254K

我们进一步对比和总结了各个版本间的差异.首先,本文在第 3.1 节已经阐述过,不同的数据收集方式获得

的原始数据模式不同.其次,爬取的数据由于经历了较长的数据下载过程,一致性较弱.例如,一小部分问题报告的属性值与其最后一次变更的值不匹配,这是因为第 3.1 节提到了两类 Web 页面是分开收集的,存在一定的时间差,这期间可能发生相关属性的修改活动.最后,新版本的数据还包含了一些新的变化,主要包括以下 5 类.

(1) 新增加与减少的问题报告.新增问题报告主要是随时间推移而新提交的报告,此外,还包括了过去未被公开的敏感报告.减少的问题报告主要包括因具有敏感性而被隐藏的报告.

(2) 新增的问题报告属性值.随着开发的演进,某些问题报告属性的值域会产生一定变化,例如某个问题所属的软件版本,在较新版本的数据集中,该属性的取值范围会增加后续编排的新版本号.

(3) 新增的开发活动参与者.开发社区中不断地有新的人员参与进来,如新的问题报告者、开发者、评论者等,他们的活动记录会出现在较新版本的数据集中.

(4) 新增的问题处理活动.在一次数据收集中,会有相当一部分问题报告还处于处理过程中,因此,在新收集到的数据中会增加这些问题报告的后续处理记录.

(5) 问题报告属性值的变化.根据变化的原因我们将其归为 4 种类型:① 由问题的后续处理活动所引起的,例如,问题被分配到新的开发者,问题的处理状态更新;② 由开发者对自己账户修改所引发的,如用户邮件地址的变更;③ 由于收集过程异常而导致的,如因爬虫丢失登录状态而导致的用户邮件地址后缀的缺失;④ 不同数据收集方式下原始数据格式不同造成的,例如 Web 页面中的时间戳有时区信息而数据库中的时间戳缺少该信息.

4 数据集应用示例

我们通过层次化使数据集具备了可追溯、可扩展性,通过多版本化在数据集中纳入了数据的变化,我们期望数据使用者可以利用这些特性来提高数据质量、提高数据分析结果的有效性、拓展研究范围等.本节以上一节中介绍的 Mozilla 问题追踪数据集为例,从对数据变化性的挖掘及对数据集构建过程的追溯两个方面来展示 5 个应用,示范层次化、多版本化数据集的使用,验证本文方法的有效性.

4.1 数据变化性的发掘与应用

软件开发活动会使相应的数据产生多种变化,某些变化可能会给数据使用者的分析造成一定的困难,例如同一个开发者在不同阶段使用不同的身份标识,造成开发者识别的困难;而某些变化则会给人们带来探索新问题的机遇,例如新数据集中所包含的在过去敏感的数据为敏感问题的研究提供了机会.将数据多版本化之后,我们可以通过对比某些对象及其属性在不同的版本间的变化来检查和修复数据的不一致,验证分析模型的有效性或者研究新的软件开发问题.

第 3.2 节总结了示例数据集新旧版本间的变化情况,下面就其中的 3 项具体变化:Bugzilla 用户账户的改变、活动时间戳的改变以及新增问题报告与活动数据,我们做了示范性的 3 个应用.

4.1.1 Bugzilla 用户账户的改变

Fig.6 The relationship between user account and user, e-mail address

图 6 用户账户与用户、邮件地址间的关系

Bugzilla 用户,即软件开发活动的参与者在使用该系统时需要创建自己的账户,该账户以用户所填写的邮件地址作为用户标识.因此,数据分析者通常利用邮件地址来对不同的用户进行识别,然而这种方法存在一定的局限性.图 6 展示了用户与账户、账户与邮件地址之间的对应关系.其中,一个用户可以注册多个 Bugzilla 的账户,而一个账户在使用过程中也可能更换邮件地址.因此,直接以邮件地址来识别用户可能会把一个实际的用户当作多个.目前,已有研究者尝试提出识别这些“别名”的方法[42,43].然而,“别名”的识别仍然存在很多的困难尚待解决,尤其是问题追踪数据中的“别名”识别,其中最主要的是缺乏检验这些方法是否有效的测试数据(特别地,基于有监督学习的方法缺乏必要的训练数据).因此,我们尝试在多个版本的

Mozilla 数据间进行比对,探究是否可以发现“别名”实例,是否可以进一步利用多版本的手段更完全地抽取出开发者们所使用的“别名”,建立“别名”的数据集.

Table 2 Usage of multiple e-mail address and accounts

表 2 用户使用多邮件地址、多账户的情况

邮件地址数	1	2	3	4
用户数	183 495	1 949	88	9
账户数	1	2	3	4
用户数	185 506	33	1	1

Bugzilla 中每个问题的报告者是某一个特定的开发者,不会发生改变,这是客观事实.对于某个问题,如果系统中记录的报告者的邮件地址在不同版本的数据集中不同,那么这些不同的邮件地址一定都属于该报告者,他们代表同一个人.根据该原理,我们将 4 个版本的数据集中的同一个问题的报告者的邮件地址提取出来进行对比、聚集.具体地,我们首先将关联到同一个问题的邮件地址收集到同一个集合中,然后通过非大小写敏感的字符串完全匹配将这些集合中包含相同地址的集合合并,最终得到使用过多个邮件地址用户以及他们使用过的邮件地址.此外,在一个版本的数据集中,作为一个账户的标识的邮件地址一定是唯一的,如果一个用户的多个邮件地址出现在了一个版本的数据集中,说明该用户使用了多个账户.我们利用得到的每一个用户的邮件地址集合,在每一个版本的数据集中进行邮件地址的完全匹配查找,结果发现,确实存在一个用户使用多个账户的实例.表 2 是对用户使用多个账户、多个邮件地址的情况的总结.从中可以看到,大约有 2 046 个(1949+88+9,约 1%)用户使用过两个及以上邮件地址,所涉及的邮件地址有 4 198 个.使用多个账户的用户则要少很多,我们只发现了 35 个,占比不到万分之二.

我们利用 4 个版本的数据集挖掘出了 4 000 多个“别名”邮件地址.排除外部的客观因素,例如 Bugzilla 系统的 bug 可能引入的错误数据,该方法所识别的多邮件地址、多账户一定都是真正例(true-positive),达到了本示例的目的.至于识别的完整性,该方法无法保证.如果拥有更多的版本,我们可以得到更为完全的结果.当拥有这样的“别名”数据集后,研究者便可判断“别名”对所要分析的问题所造成的威胁大小,对“别名”进行去重,甚至可以对用户使用“别名”的习惯进行挖掘,寻找相关的模式.

4.1.2 活动时间戳的改变

在全球开发背景下,数据分析者要特别注意开发活动时间戳的时区,如果时区信息被忽略,那么计算中的某个活动的时间误差可能会高达 24 小时(地理位置最多可差 24 个时区),这种误差会对某些较为精细的时间度量造成较大的威胁,例如开发者在一天当中的工作时段及在不同时段中工作的效率.在从 Bugzilla 的 Web 界面获取的原始数据中我们可以看到,所有的时间戳都标注了时区(如图 7(a)所示),而在 Bugzilla 的数据库 dump 中,时区信息是缺失的(如图 7(b)所示),那么我们能希望知道在 Bugzilla 数据库 dump 中的时间是否是以统一的时区来进行记录的.

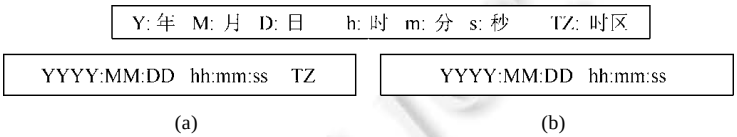

Fig.7 Time recorded in Web interface (a) and database dump (b) of Bugzilla

图 7 Bugzilla Web 界面(a)与数据库 dump (b)中记录的时间戳

如果只有 Bugzilla 数据库的 dump,我们很难去回答这个问题.但当我们拥有了从 Web 界面获取的数据,就可以对比同一个事件,例如一个问题报告的提交在两个不同版本数据集中的时间.我们对比了通过所有报告在 Web 界面获得的数据中记录的提交时间和数据库 dump 中记录的提交时间,结果发现,数据库 dump 中的时间戳比 Web 界面中的时间戳少了时区,其他数字一致.这说明,数据库 dump 中所记录的活动时间是不准确的.为了探查这种偏差的程度以及是否有可能修复,我们进一步分析了 Web 界面中时间戳的时区,结果发现,只涉及到了两

个时区,分别是太平洋标准时间(PST-0800)和太平洋夏季时间(PDT-0700).其中,太平洋夏季时间在美国的夏季使用:在 2006 年以及之前,PDT 始于每年 4 月第 1 个星期日深夜二时正,并终于 10 月的最后一个星期日深夜二时正.在 2007 年及以后,PDT 始于每年 3 月的第 2 个星期六深夜二时正,并终于每年 11 月的第 1 个星期日深夜二时正.我们对 Web 界面获取的数据集进行了分析,查看了 PST 与 PDT 在月份上的分布,结果符合上述规则.因此,我们可以利用该规则对数据库 dump 中的时间进行修正,填补相应的时区信息,例如在数据库 dump 中,第 665317 号问题的报告时间是 2011-06-18 13:35:00,它发生于 2007 年之后,具体时间在 6 月,处于 3 月的第 2 个星期六深夜二时和 11 月的第 1 个星期日深夜二时之间,填补时区 PDT.再如第 619558 号问题的报告时间为 2010-12-15 16:22:00,在上述范围之外,填补时区 PST.

4.1.3 新增问题报告与活动数据

在挖掘问题追踪数据工作中,有很多都是构造某种分类模型,帮助提高问题处理的效率.例如,对问题报告是否为重复报告(与某个正在处理的报告反映的是相同的问题)进行识别的模型.这些模型需要训练数据来构造,也需要测试数据来评估模型的性能并对其进行调整.获取这两类数据的通常的做法是将一个完整的数据集分成两个部分:一部分作为训练数据,另一部分作为测试数据.这样的测试数据是否能有效地代表实际应用中的输入数据决定了模型的可用性,是一个值得关注的问题.当我们拥有了两个在不同时间段所收集的数据集时,便可以构造出 3 类数据:实验中的训练数据、测试数据(较早版本中的数据)、实际应用中的数据(较新版本中的数据)来回答该问题.下面,我们就以 2013 年与 2016 两个版本的 Mozilla 数据以及识别重复问题报告的分类器为例进行研究.

首先,我们构造一个简单的识别重复问题报告的分类器:该模型基于逻辑斯蒂回归,使用问题报告提交者的问题报告熟练程度作为预测因子(很多研究显示,开发者的经验与其工作产出的质量相关[44]),其中,问题报告是否是重复报告以问题的在 Bugzilla 中所记录的 Solution 来标定,Solution 为 DUPLICATE 的为重复报告;问题报告的熟练程度以问题报告提交者在此之前所提交的问题数量来量化,由于熟练程度还与距离上一次提交问题报告的时间间隔存在一定的关联性,我们将时间限制为半年内,即问题报告提交者在此之前半年内所提交的问题数量(NR).只要该模型的判定能力强于随机猜测即可达到实验目的.模型如下所示:

$$is_{duplicate} \sim NR \tag{1}$$

其次,基于 2013 年的数据集进行模型的训练、测试及调整.我们选取了数据集中最近 5 年,即 2008 年 1 月 1 日(北京时间,下同)后得到解决的(有 Solution 的)问题报告.此外,问题的最终解决需要经过一段时间的验证,即发现 Solution 的错误并纠正,因此,数据集中最后一段时间的问题报告的 Solution 有相对较高的不稳定性,无法准确地判断其是否是重复问题报告.我们对关于 DUPLICATE 的 Solution 错误出现的频率及其纠正时间进行了统计,发现有约 2%的报告存在这样的错误,其中超过 75%的报告在 1 年内得到了纠正.因此,我们过滤掉其中最后一年的问题报告,即保留 2008 年 1 月 1 日~2012 年 1 月 1 日之间提交的问题报告,保证其中不准确报告的数量小于 0.5%(2%×(1–75%))(由于部分错误的纠正需要 10 年以上的时间,为了保证有足够的数据,本文以损失 1 年的数据为代价使不准确的报告数量控制在 0.5%以内).对剩余报告,我们按照其被提交的先后顺序选取了前 80%作为训练数据,而后 20%作为测试数据.训练结果见表 3,可以看到,NR 的系数显著不为 0,其解释度为 2.9%,这说明,该模型具备判定能力,满足本示例的需求.

Table 3 Model training result

表 3 模型训练结果

	Estimate	Std. Error	*P*-value	Deviance
(Intercept)	–1.239 802 5	0.007 469 9	<0.01	–
NR	–0.010 043 4	0.000 176 4	<0.01	2.9% (4820.5/167671)

模型 1 的输出值表征了一个问题报告是否是重复报告的概率,在应用该模型做判定前还要设置一个阈值,当模型的输出大于这个阈值时,即判定该问题报告是重复的.为了选取合适的阈值,我们对输入训练集得到的输出的分布进行统计,每 10%分位点取一个值(即 10%,20%,30%,…,90%分位数)作为 9 个候选的阈值,利用测试集分别评估选取不同阈值时模型的性能,模型的性能以 *F*1-score,即准确率与召回率的调和平均数的两倍来度量.

表 4 展示了在不同阈值下模型的性能,从中我们可以看到,在阈值为 0.214 2(60%分位点)时,模型性能达到最优.

Table 4 Threshold selection and model performance in test

表 4 阈值选择及模型在测试中的性能

分位点	阈值	准确率	召回率	*F*1-score
10%	0.085 122 16	0.155 397 7	0.931 374	0.266 354 7
20%	0.132 093 03	0.167 335 4	0.846 538 4	0.279 434 9
30%	0.158 202 01	0.181 557	0.765 944 6	0.293 535 4
40%	0.180 794 57	0.201 835 7	0.682 926 8	0.311 584 2
50%	0.199 335 44	0.230 764 8	0.601 121	0.333 501 4
60%	0.214 154 46	0.268 476 9	0.506 286 9	0.350 884 6
70%	0.220 992 96	0.289 546 3	0.397 364	0.334 993 6
80%	0.222 726 82	0.292 944 8	0.318 285 1	0.305 089 7
90%	0.222 726 82	0.292 944 8	0.318 285 1	0.305 089 7

接下来,我们探究在实际应用中,选取这些阈值的模型是否也能够有相同的表现.我们使用 2016 年数据集来模拟实际应用的场景.同样,考虑到问题处理结果的稳定性,除去数据集中最后一年的问题报告,为了测试已有模型在未来应用中的性能,选取 2013 年数据集收集 1 年之后所提交的新报告,即从 2016 年数据集中节选 2014 年 1 月~2015 年 1 月的 1 年间提交的已解决的问题报告.实验结果见表 5,从中可以看到,60%分位点的阈值已不具有最佳的分类表现,并且在新的数据集上无论选取哪个阈值,模型的 *F*1-score 都不如实验阶段的结果.为了探索其中的原因,我们首先比较了自变量 NR 在训练数据和实际应用数据中的均值和平均数,它们分别从 40 和 15 上升到 831 和 43,发生了较大的改变;其次.我们使用新增数据对模型重新训练,结果表明,报告者近期提交报告的数量的解释度远高于使用 2013 年数据训练的模型(见表 6),也就是说,模型预测能力的下降是因为实验中的模型参数已不再适用于新的应用场景.而模型解释度的提升反映出,随着时间的推移而新增的数据出现了新的特点,我们推测在 2013 年之后,Mozilla 社区的一些实践方法的优化措施,例如 Bugzilla 问题报告引导程序的改进有效地训练了新手完成问题报告的能力.

Table 5 Threshold selection and model performance in application

表 5 阈值选择及模型在实际应用中的性能

分位点	阈值	准确率	召回率	*F*1-score
10%	0.085 122 16	0.143 431	0.885 432 9	0.246 871 4
20%	0.132 093 03	0.149 879 6	0.776 179 7	0.251 244 2
30%	0.158 202 01	0.158 044 7	0.672 098 6	0.255 911 5
40%	0.180 794 57	0.171 305 2	0.556 468 8	0.261 965 9
50%	0.199 335 44	0.193 057 8	0.442 539 3	0.268 835 9
60%	0.214 154 46	0.228 511 8	0.326 059 2	0.268 706 4
70%	0.220 992 96	0.255 434 8	0.236 431 9	0.245 566 3
80%	0.222 726 82	0.261 061	0.180 600 8	0.213 502
90%	0.222 726 82	0.261 061	0.180 600 8	0.213 502

Table 6 Result of model training with new data

表 6 使用新增数据进行模型训练的结果

	Estimate	Std. Error	*P*-value	Deviance
(Intercept)	–1.759	9.845e-03	<0.01	–
NR	–1.232e-03	5.633e-05	<0.01	5.2% (4 560.4/86890)

4.2 数据可追溯性的应用

本文第 2 节阐释了数据集构建过程的可追溯性可以解决两个方面的问题:一是数据集的适用范围,当数据使用者发现其他人定制的高层数据(例如 FTDD 数据集[30])中缺少了所需要的信息时,可以去回溯到层次 1,寻找缺失的数据,重新构建层次 2 的数据;二是数据质量的保障,由于整个数据集构建的处理过程及中间数据都是公开的,任何人都可以对数据的处理脚本进行测试,对数据进行比对,发现数据存在的缺陷或者局限,并评估他们对上层分析结果的影响.在下面两个小节中,我们继续以 Mozilla 问题追踪数据集为例,分别展示面向这两方面问题的应用.

4.2.1 数据的定制与适用范围

定制化的数据为解决某些特定的问题提供了一定的便利.在第 2.2 节中我们提到了 Habayeb 等人的 FTDD 数据集[30],该数据集记录了缺陷处理过程中的各类事件及其发生时间,可以帮助人们研究它们发生的模式及产生的影响.该数据集以 Firefox 所使用的 Bugzilla 问题追踪数据为原始数据,作者编码了 3 大类 10 种事件,据此提取出每个缺陷报告中的事件.按照本文所提出的数据层次的概念,该数据集属于定制层,即层次 2 或层次 2+的数据.由于具有同源性,该数据集可以由我们 Mozilla 问题追踪数据集中层次 1 的数据加工而成.图 8 所示为该数据集的数据模型,可以看到,相对于原始数据,每个问题丢失了一部分信息,比如缺陷报告的优先级、严重程度等属性,缺陷报告的标题、具体描述等文本信息.如果某个研究者想要研究不同类型的缺陷的处理模式,由于缺少这些问题属性和描述,FTDD 数据集并不能提供相应的数据支撑.

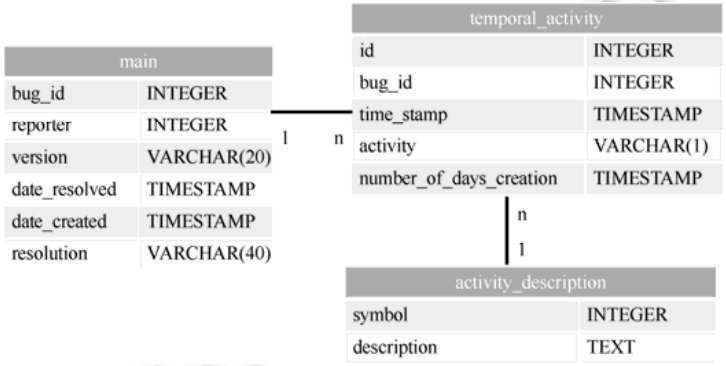

Fig.8 FTDD schema

图 8 FTDD 数据模型

层次化的数据集为解决这种定制数据应用范围有限的问题提供了有效的解决途径.对于缺陷的类型,我们可以从很多角度进行分析,可以按问题所属的产品、模块分类,也可以按缺陷报告的优先级、严重程度分类,还可以利用主题生成模型从缺陷的描述文本中挖掘主题并根据主题对缺陷报告进行分类.因此,我们尝试回溯到层次 1,对相关的信息进行再提取,包括缺陷报告的 4 种属性(所属产品、所属模块、优先级、严重程度)以及每个报告的标题和描述.之后,根据缺陷的 ID,我们将这些信息关联到 FTDD 数据中的每一个缺陷报告,构建了一个新的如图 9 所示的层次 2 数据集,它可以支撑不同类型缺陷的处理模式的研究.FTDD 的作者在相关文献中同样提到了数据集扩展的问题,但没有说明如何获取扩展所需的数据,而当该数据集以层次化的方法构建与使用时,这种扩展在实际应用中的可行性将大为增加.

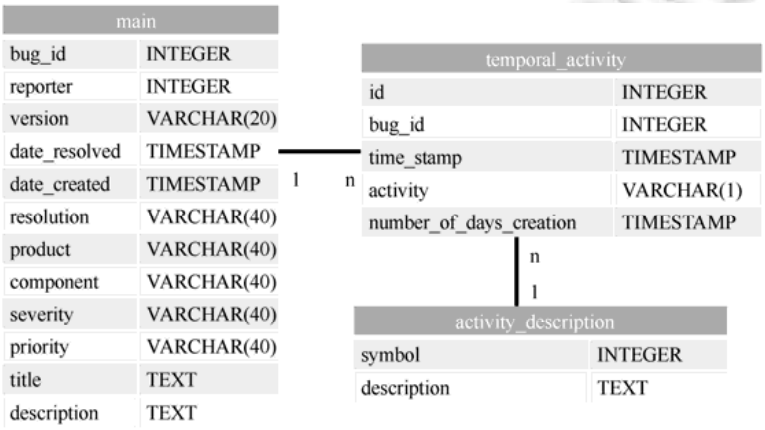

Fig.9 Extended FTDD schema

图 9 扩展的 FTDD 数据模型

4.2.2 数据质量的保障

产品质量是消费者们最为关心的一种属性.很多食品生产厂商通过使食品生产过程可追溯[45],让每一个生产环节公开、可查验来保证食品的质量.同样,通过使数据集的构建过程可追溯来保证其面向最终用户的质量同样值得数据发布者们尝试.层次化的 Mozilla 数据集实现了数据集构建过程的可追溯.除了直接面向数据使用者的定制数据外,我们还保留了定制数据构建的全过程:在层次 0,有从数据源获取的全部原始的数据和数据获取的描述及脚本.在层次 1,有将原始数据标准化后的中间数据及进行标准化处理的脚本.数据集构建的所有环节都是可查验的,当数据使用者在使用过程中发现了可疑的问题时可以追溯到上述任意一个处理环节,查找问题的根源.

下面是我们在使用该数据集时所遇到的一个典型的例子.当我们利用 Bugzilla 用户的邮件地址来识别不同的用户时,发现 2011 年数据集中有一些用户的邮件地址并不完整.这很可能会导致同一个用户被当作不同的用户来处理.为了查找问题的根源,我们取得含有不完整地址的问题报告的 ID,直接在 Mozilla 的 Bugzilla 页面中进行查询,此时看到的邮件地址是完整的,也就是说,问题出在了数据集构建的某个环节上.我们首先从层次 0 的原始数据开始排查,结果发现,原始数据中这些邮件地址已不完整.这说明,我们的页面下载脚本存在问题.Bugzilla 有这样一种访问规则:只有登录的用户才可以查看其他用户的完整邮件地址,因此,数据下载的脚本要进行登录操作并在下载过程中保持登录状态.虽然之前的脚本中有登录机制,但存在不能保证登录状态的缺陷,部分数据的下载是在未登录状态下进行的,因而没有获取到完整的用户邮件地址.据此,我们修复了该脚本,提高了下载到的数据的质量.

5 结束语

共享数据集的构建与使用是提高软件开发活动数据分析效率的一种途径,而现有工作存在对数据可用性欠考虑的问题,直接威胁数据质量与数据分析结果的有效性.为此,本文提出一种层次化、多版本化的数据集构建与使用方法,通过划分不同的数据层建立数据的可追溯性,通过多版本数据的收集纳入可能出现的数据变化.利用这两项设计,使用者可以验证、提高数据质量和数据分析结果的有效性.目前,我们已在该方法框架下完成了 Mozilla 问题追踪数据集的构建与使用,本文中也分享了我们的相关经验,验证了该方法的有效性.近期,我们注意到有些工作同样尝试了在定制数据之外提供原始数据,以此方便数据使用者开展更多的分析,例如 Beller 等人的 TravisTorrent 数据集[46].在未来工作中,我们将继续实践该方法,构建其他类型软件开发活动的数据集,如代码审查等.我们也将继续使用层次化、多版本化的数据集开展软件开发问题的研究,积累更多的经验,进一步完善该方法.特别地,当数据集的层次 2 和层次 2+中积累了纷杂的定制数据时,基于信息检索等方法,实现面向特定用户需求的自动化数据推荐.此外,数据规模的不断扩大也会给数据集的使用带来新的挑战,我们在未来也将尝试借鉴 Gousios 等人的方法[7,17],通过分布式、P2P 等方式使大规模数据的存储与访问更为高效.

References:

[1] Zhou MH, Guo CG. Bigdata-based thought of software engineering. Communications of the CCF, 2014,10(3):37–42 (in Chinese with English abstract).

[2] Mockus A. Engineering big data solutions. In: Proc. of the Future of Software Engineering. ACM, 2014. 85–99.

[3] Hassan AE. The road ahead for mining software repositories. In: Frontiers of Software Maintenance, FoSM 2008. IEEE, 2008. 48–57.

[4] Hassan AE, Xie T. Mining software engineering data. In: Proc. of the 32nd ACM/IEEE Int'l Conf. on Software Engineering, Vol. 2. ACM, 2010. 503–504.

[5] Howison J, Conklin M, Crowston K. FLOSSmole: A collaborative repository for FLOSS research data and analyses. Int'l Journal of Information Technology and Web Engineering (IJITWE), 2006,1(3):17–26.

[6] Boetticher G, Menzies T, Ostrand T. The promise repository of empirical software engineering data. 2016. http://openscience.us/repo

[7] Gousios G, Spinellis D. GHTorrent: Github's data from a firehose. In: Proc. of the 9th IEEE Working Conf. on Mining Software Repositories (MSR). IEEE, 2012. 12–21. [doi: 10.1109/MSR.2012.6224294]

[8] Zhu JX, Lin HW, Zhou MH, Mei H. Review code evolution history in OSS universe. In: Proc. of the 4th Asia-Pacific Symp. on Internetware. ACM, 2012. 13.

[9] Bacchelli A. Mining challenge 2013: Stack overflow. In: Proc. of the 10th Working Conf. on Mining Software Repositories. 2013.

[10] Liebchen GA, Shepperd M. Data sets and data quality in software engineering. In: Proc. of the 4th Int'l Workshop on Predictor Models in Software Engineering. ACM, 2008. 39–44.

[11] Siegmund J, Siegmund N, Apel S. Views on internal and external validity in empirical software engineering. In: Proc. of the 2015 IEEE/ACM, the 37th IEEE Int'l Conf. on Software Engineering. IEEE, 2015. 1:9–19.

[12] Adcock R, Collier D. Measurement validity: A shared standard for qualitative and quantitative research. American Political Science Review, 2001,95(3):529–546.

[13] Guo ZM, Zhou AY. Research on data quality and data cleaning: A survey. Ruan Jian Xue Bao/Journal of Software, 2002,13(11): 2076–2082 (in Chinese with English abstract). http://www.jos.org.cn/jos/ch/reader/create_pdf.aspx?file_no=20021103&journal_id=jos

[14] Zhu JX, Zhou MH, Mei H. Multi-extract and multi-level dataset of mozilla issue tracking history. In: Proc. of the 13th Int'l Workshop on Mining Software Repositories. ACM, 2016. 472–475.

[15] Mockus A. Amassing and indexing a large sample of version control systems: Towards the census of public source code history. MSR, 2009,9:11–20.

[16] Spinellis D. A repository with 44 years of Unix evolution. In: Proc. of the 12th Working Conf. on Mining Software Repositories. IEEE Press, 2015. 462–465.

[17] Gousios G, Vasilescu B, Serebrenik A, Zaidman A. Lean GHTorrent: GitHub data on demand. In: Proc. of the Working Conf. on Mining Software Repositories. ACM, 2014. 384–387.

[18] Amani S, Nadi S, Nguyen HA, Nguyen TN, Mezini M. MUBench: A benchmark for API-misuse detectors. In: Proc. of the 13th Int'l Workshop on Mining Software Repositories. ACM, 2016. 464–467.

[19] Keivanloo I, Forbes C, Hmood A, Erfani M, Neal C, Peristerakis G, Rilling J. A linked data platform for mining software repositories. In: Proc. of the 9th IEEE Working Conf. on Mining Software Repositories (MSR). IEEE, 2012. 32–35.

[20] Li JZ, Liu XM. An important aspect of big data: Data usability. Journal of Computer Research and Development, 2013,50(6): 1147–1162 (in Chinese with English abstract).

[21] Sidi F, ShariatPanahy PH, Affendey LS, Jabar MA, Ibrahim H, Mustapha A. Data quality: A survey of data quality dimensions. In: Proc. of the 2012 Int'l Conf. on Information Retrieval & Knowledge Management. IEEE, 2012. 300–304.

[22] Wang RY, Strong DM. Beyond accuracy: What data quality means to data consumers. Journal of Management Information Systems, 1996,12(4):5–33.

[23] McGilvray D. Executing Data Quality Projects: Ten Steps to Qualitydata and Trusted Information. Elsevier, 2008. 16–59.

[24] Ding XO, Wang HZ, Zhang XY, Li JZ, Gao H. Association relationships study of multi-dimensional data quality. Ruan Jian Xue Bao/Journal of Software, 2016,27(7):1626–1644 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5040.htm [doi: 10.13328/j.cnki.jos.005040]

[25] Nagappan M, Zimmermann T, Bird C. Diversity in software engineering research. In: Proc. of the 9th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2013). ACM, 2013. 466–476.

[26] Mockus A. Missing data in software engineering. In: Guide to Advanced Empirical Software Engineering. London: Springer-Verlag, 2008. 185–200.

[27] Zheng QM, Mockus A, Zhou MH. A method to identify and correct problematic software activity data: Exploiting capacity constraints and data redundancies. In: Proc. of the Joint Meeting on Foundations of Software Engineering. ACM, 2015. 637–648.

[28] Tantithamthavorn C, McIntosh S, Hassan AE, Ihara A, Matsumoto K. The impact of mislabelling on the performance and interpretation of defect prediction models. In: Proc. of the 2015 IEEE/ACM, the 37th IEEE Int'l Conf. on Software Engineering. IEEE, 2015,1:812–823.

[29] MSR07 Mining Challenge. http://msr.uwaterloo.ca/msr2007/challenge/

[30] Habayeb M, Miranskyy A, Murtaza SS, Buchanan L, Bener A. The Firefox temporal defect dataset. In: Proc. of the 12th Working Conf. on Mining Software Repositories. IEEE Press, 2015. 498–501.

[31] Ioannidis JPA. Why most published research findings are false. PLoS Medicine, 2005,2(8):e124.

[32] Shafranovich Y. Common Format and MIME Type for Comma-Separated Values (CSV) Files. Heise Zeitschriften Verlag, 2005.

[33] Rigby PC, German DM, Storey MA. Open source software peer review practices: A case study of the apache server. In: Proc. of the 30th Int'l Conf. on Software Engineering. ACM, 2008. 541–550.

[34] Kagdi H, Maletic J, Sharif B. Mining software repositories for traceability links. In: Proc. of the 15th IEEE Int'l Conf. on Program Comprehension (ICPC 2007). Banff, 2007. 145–154.

[35] Zhou MH, Mockus A. Who will stay in the floss community? modeling participant's initial behavior. IEEE Trans. on Software Engineering, 2015,41(1):82–99.

[36] Serrano N, Ciordia I. Bugzilla, ITracker, and other bug trackers. IEEE Software, 2005,22(2):11–13.

[37] Kamei Y, Fukushima T, McIntosh S, Yamashita K, Ubayashi N, Hassan AE. Studying just-in-time defect prediction using cross-project models. Empirical Software Engineering, 2016,21(5):2072–2106.

[38] Zhou J, Zhang H, Lo D. Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports. In: Proc. of the Int'l Conf. on Software Engineering. IEEE, 2012. 14–24.

[39] Xie JL, Zhou MH, Mockus A. Impact of triage: A study of Mozilla and Gnome. In: Proc. of the 2013 ACM/IEEE Int'l Symp. on Empirical Software Engineering and Measurement. IEEE, 2013. 247–250.

[40] Liu C, Yang J, Tan L, *et al*. R2Fix: Automatically generating bug fixes from bug reports. In: Proc. of the 6th IEEE Int'l Conf. on Software Testing, Verification and Validation. IEEE, 2013. 282–291.

[41] Nurolahzade M, Nasehi SM, Khandkar SH, Rawal S. The role of patch review in software evolution: An analysis of the Mozilla firefox. In: Proc. of the Joint Int'l and ERCIM Workshops on Principles of Software Evolution. ACM, 2009. 9–18.

[42] Anwar T, Abulaish M, Alghathbar K. Web content mining for alias identification: A first step towards suspect tracking. In: Proc. of the 2011 IEEE Int'l Conf. on Intelligence and Security Informatics (ISI). IEEE, 2011. 195–197.

[43] Subathra M, Nedunchezhian R. A novel fuzzy logic model to identify closeness for alias detection. Indian Journal of Science and Technology, 2015,8(28):1.

[44] Xia X, Lo D, Shihab E, Wang XY. Automatic, high accuracy prediction of reopened bugs. Automated Software Engineering, 2015, 22(1):75–109.

[45] Wei XL, Deng CJ, Meng QX. Domestic and international research progress of quality and safety traceability system of the whole beef production process. Feed Research, 2012,(9):16–17 (in Chinese with English abstract).

[46] Beller M, Gousios G, Zaidman A. Travistorrent: Synthesizing Travisci and Github for full-stack research on continuous integration. In: Proc. of the 14th Int'l Conf. on Mining Software Repositories. IEEE Press, 2017. 447–450.

附中文参考文献:

[1] 周明辉,郭长国.基于大数据的软件工程新思维.中国计算机学会通信,2014,10(3):37–42.

[13] 郭志懋,周傲英.数据质量和数据清洗研究综述.软件学报,2002,13(11):2076–2082. http://www.jos.org.cn/jos/ch/reader/create_pdf.aspx?file_no=20021103&journal_id=jos

[20] 李建中,刘显敏.大数据的一个重要方面:数据可用性.计算机研究与发展,2013,50(6):1147–1162.

[24] 丁小欧,王宏志,张笑影,李建中,高宏.数据质量多种性质的关联关系研究.软件学报,2016,27(7):1626–1644. http://www.jos.org.cn/1000-9825/5040.htm [doi: 10.13328/j.cnki.jos.005040]

[45] 魏秀莲,邓程君,孟庆翔.肉牛生产全程质量安全追溯体系国内外研究进展.饲料研究,2012,(9):16–17.

朱家鑫(1988－),男,河北行唐人,博士,助理研究员,CCF 专业会员,主要研究领域为软件工程,软件及软件开发活动度量和分析,开源软件.

周明辉(1974－),女,博士,副教授,CCF 专业会员,主要研究领域为软件工程,软件及软件开发活动度量和分析,开源软件.