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.

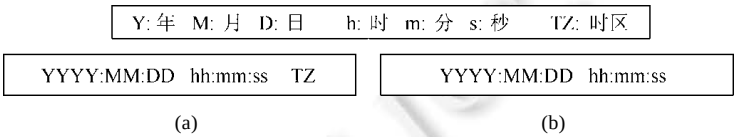

Fig.7 Time recorded in Web interface (a) and database dump (b) of Bugzilla

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.

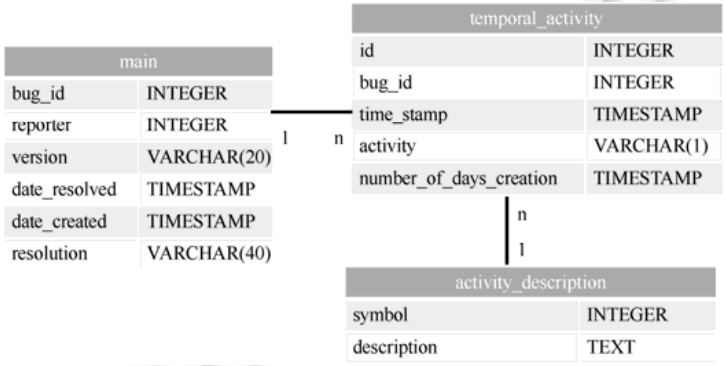

Fig.8 FTDD schema

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.

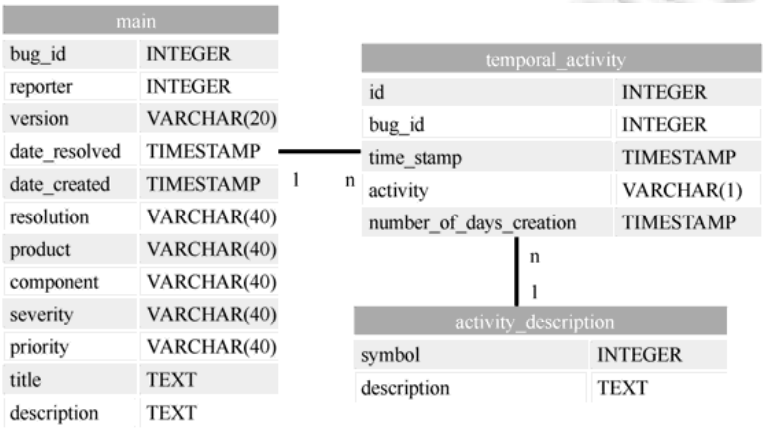

Fig.9 Extended FTDD schema

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.