软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(1):1−25 [doi: 10.13328/j.cnki.jos.004923]

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

静态软件缺陷预测方法研究*

陈 翔[1,2], 顾 庆[2], 刘望舒[2], 刘树龙[2], 倪 超[2]

[1](南通大学 计算机科学与技术学院,江苏 南通 226019)

[2](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

通讯作者: 陈翔, E-mail: xchencs@ntu.edu.cn

摘 要: 静态软件缺陷预测是软件工程数据挖掘领域中的一个研究热点.通过分析软件代码或开发过程,设计出与软件缺陷相关的度量元;随后,通过挖掘软件历史仓库来创建缺陷预测数据集,旨在构建出缺陷预测模型,以预测出被测项目内的潜在缺陷程序模块,最终达到优化测试资源分配和提高软件产品质量的目的.对近些年来国内外学者在该研究领域取得的成果进行了系统的总结.首先,给出了研究框架并识别出了影响缺陷预测性能的 3 个重要影响因素:度量元的设定、缺陷预测模型的构建方法和缺陷预测数据集的相关问题;接着,依次总结了这 3 个影响因素的已有研究成果;随后,总结了一类特殊的软件缺陷预测问题(即,基于代码修改的缺陷预测)的已有研究工作;最后,对未来研究可能面临的挑战进行了展望.

关键词: 软件质量保障;软件缺陷预测;软件度量元;机器学习;数据集预处理

中图法分类号: TP311

中文引用格式: 陈翔,顾庆,刘望舒,刘树龙,倪超.静态软件缺陷预测方法研究.软件学报,2016,27(1):1−25. http://www.jos.org.cn/1000-9825/4923.htm

英文引用格式: Chen X, Gu Q, Liu WS, Liu SL, Ni C. Survey of static software defect prediction. Ruan Jian Xue Bao/Journal of Software, 2016,27(1):1−25 (in Chinese). http://www.jos.org.cn/1000-9825/4923.htm

Survey of Static Software Defect Prediction

CHEN Xiang[1,2], GU Qing[2], LIU Wang-Shu[2], LIU Shu-Long[2], NI Chao[2]

[1](School of Computer Science and Technology, Nantong University, Nantong 226019, China)

[2](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

Abstract: Static software defect prediction is an active research topic in the domain of software engineering data mining. The phases of the study include designing novel code or process metrics to characterize the faults in the program modules, constructing software defect prediction model based on the training data gathered after mining software historical repositories, using the trained model to predict potential defect-proneness of program modules. The research on software defect prediction can optimize the allocation of testing resources and improve the quality of software. This paper offers a systematic survey of existing research achievements of the domestic and foreign researchers in recent years. First, a research framework is proposed and three key factors (i.e., metrics, model construction approaches, and issues in datasets) influencing the performance of defect prediction are identified. Next, existing research achievements in these three key factors are discussed in sequence. Then, the existing achievements on a special defect prediction issues (i.e., code change based defect prediction) are summarized. Finally a perspective of the future work in this research area is discussed.

Key words: software quality assurance; software defect prediction; software metrics; machine learning; data preprocessing

* 基金项目: 国家自然科学基金(61202006, 61373012, 61202030); 南京大学计算机软件新技术国家重点实验室开放课题(KFKT2012B29)

Foundation item: National Natural Science Foundation of China (61202006, 61373012, 61202030); Open Project Program of the State Key Laboratory for Novel Software Technology (Nanjing University) (KFKT2012B29)

收稿时间: 2015-05-12; 修改时间: 2015-06-24, 2015-07-27; 采用时间: 2015-10-09; jos 在线出版时间: 2015-11-03

CNKI 网络优先出版: 2015-11-04 17:10:06, http://www.cnki.net/kcms/detail/11.2560.TP.20151104.1710.006.html

软件缺陷(software defect)产生于开发人员的编码过程,需求理解不正确、软件开发过程不合理或开发人员的经验不足,均有可能产生软件缺陷.而含有缺陷的软件在运行时可能会产生意料之外的结果或行为,严重的时候会给企业造成巨大的经济损失,甚至会威胁到人们的生命安全.在软件项目的开发生命周期中,检测出内在缺陷的时间越晚,修复该缺陷的代价也越高.尤其在软件发布后,检测和修复缺陷的代价将大幅度增加.因此,项目主管借助软件测试或代码审查等软件质量保障手段,希望能够在软件部署前尽可能多地检测出内在缺陷.但是,若关注所有的程序模块会消耗大量的人力物力,因此,项目主管希望能够预先识别出可能含有缺陷的程序模块,并对其分配足够的测试资源.

软件缺陷预测[1,2]是其中一种可行的方法,根据软件历史开发数据以及已发现的缺陷,借助机器学习等方法来预测软件项目中的缺陷数目和类型等.目前,已有的软件缺陷方法可以简单地分为静态缺陷预测方法和动态缺陷预测方法[1],其中,静态预测方法基于缺陷相关的度量数据,对程序模块的缺陷倾向性、缺陷密度或缺陷数进行预测;而动态缺陷预测方法则是基于缺陷或失效产生的时间对系统缺陷随时间的分布进行预测,以发现软件缺陷随其生命周期或其中某些阶段的时间关系的分布规律.

本文重点对静态软件缺陷预测的已有研究工作进行综述.具体来说,该类方法通过分析软件代码或开发过程设计出与软件缺陷相关的度量元,随后,通过挖掘软件历史仓库(software historical repositories)来创建缺陷预测数据集.目前,可以挖掘与分析的软件历史仓库包括项目所处的版本控制系统(例如 CVS,SVN 或 Git 等)、缺陷跟踪系统(例如 Bugzilla,Mantis,Jira 或 Trac 等)或相关开发人员的电子邮件等.最后,基于上述搜集的缺陷预测数据集,构建缺陷预测模型,并用于预测出项目内的潜在缺陷程序模块.

静态软件缺陷预测属于当前软件工程数据挖掘领域[3]中的一个重要研究问题.随着新的数据挖掘技术的不断涌现以及研究人员对软件历史仓库挖掘的日益深入,静态软件缺陷预测研究在近些年来取得了大量的研究成果,其缺陷预测结果也逐渐成为判断一个系统是否可以交付使用的重要依据.因此,针对该问题的深入研究对提高和保障软件产品的质量具有重要的研究意义.

为了对该研究问题进行系统的分析、总结和比较,我们首先在 IEEE,ACM,Springer,Elsevier 和 CNKI 等论文数据库中进行检索,检索时采用的主要英文关键词包括 defect prediction,software defect prediction,fault prediction 和 software fault prediction 等;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;最终,我们选择出与该研究问题直接相关的高质量论文共 113 篇(截止到 2015 年 7 月).从选择出的论文所发表的会议和期刊来看,绝大部分论文发表在软件工程领域的权威会议或期刊上,例如 ICSE 会议(21 篇)、ESEC/FSE(或 FSE)会议(13 篇)、ISSRE 会议(5 篇)、TSE 期刊(16 篇)、IST 期刊(7 篇)和 JSS 期刊(4 篇)等.

本文第 1 节对静态软件缺陷预测方法的研究框架进行总结,并识别出其中 3 个重要的影响因素(即,度量元的设定、缺陷预测模型的构建方法和缺陷预测数据集的相关问题).第 2 节对已有的度量元设计进行总结.第 3 节对已有的缺陷预测模型构建方法进行总结.第 4 节对缺陷预测数据集相关问题的产生根源和解决方法进行总结.第 5 节对一类特殊的软件缺陷预测问题(即,基于代码修改的缺陷预测)的已有研究工作进行总结.传统的缺陷预测问题重点预测的是程序模块内部是否含有缺陷,而该问题的特殊之处在于需要预测出提交的代码修改是否会产生缺陷.最后总结全文,并对未来值得关注的研究方向进行初步探讨.

1 研究框架

静态软件缺陷预测可以将程序模块的缺陷倾向性、缺陷密度或缺陷数设置为预测目标.以预测模块的缺陷倾向性为例,其典型研究框架如图 1 所示.

图 1 上半部分总结的是软件缺陷预测过程,该过程主要包括两个阶段:模型构建阶段和模型应用阶段.具体来说,模型构建阶段包括 3 个步骤.

(1) 挖掘软件历史仓库,从中抽取出程序模块.程序模块的粒度根据实际应用的场景,可设置为文件、包、类或函数等.随后,将该程序模块标记为有缺陷模块或无缺陷模块,在图 1 中,我们将有缺陷模块用红

色进行标记,无缺陷模块用绿色进行标记;

(2) 通过分析软件代码或开发过程设计出与软件缺陷存在相关性的度量元,借助这些度量元对程序模块进行软件度量,并构建出缺陷预测数据集;

(3) 对缺陷预测数据集进行必要的数据预处理(例如噪音移除、特征子集选择、数据归一化等)后,借助特定的建模方法构建出缺陷预测模型.大部分建模方法都基于机器学习方法,其常用的模型性能评测指标包括准确率(accuracy)、查准率(precision)、查全率(recall)、*F*-measure 或 AUC(area under the ROC curve)取值等.

而在模型应用阶段,当面对新程序模块时,在完成对该模块的软件度量后,基于前一阶段构造出的缺陷预测模型和具体度量元取值,可以完成对该模块的分类,即将该模块预测为有缺陷倾向性(defect-proneness,简称 FP)模块或无缺陷倾向性(non defect-proneness,简称 NFP)模块.

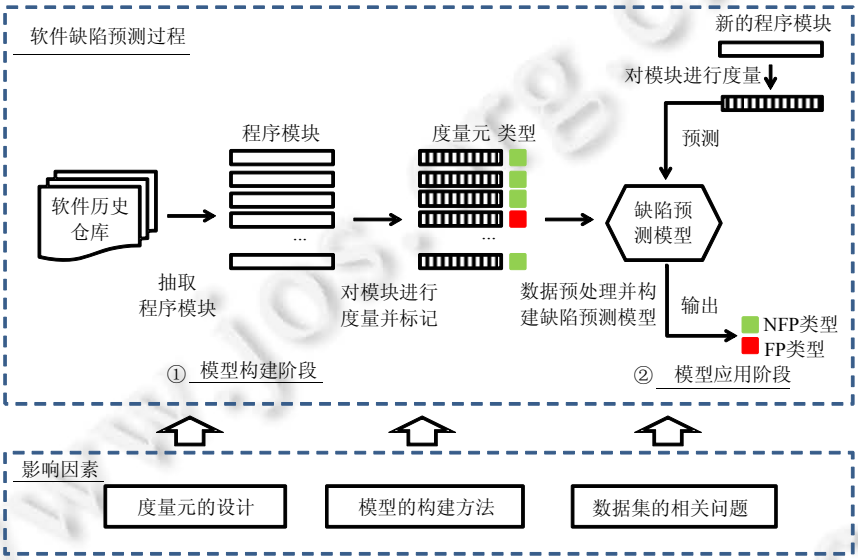

Fig.1 Static software defect prediction research framework using defect-proneness as prediction target

图 1 以缺陷倾向性为预测目标的静态软件缺陷预测研究框架

若将预测目标设置为缺陷密度或缺陷数,则其预测流程与图 1 基本相同,主要的不同点是模型构建阶段中的模块标记(即,需要标记出已有模块内的缺陷密度或缺陷数)和模型预测阶段中的新模块的类型输出(即,预测输出的是新模块内的缺陷密度或缺陷数).

通过分析上述软件缺陷预测过程,我们识别出影响缺陷预测性能的 3 个重要影响因素(如图 1 的下半部分所示).

(1) 度量元的设计(见第 2 节).

挖掘软件历史仓库、设置新颖的与软件缺陷存在强相关性的度量元,是构建高质量缺陷预测模型的关键.本文将已有的度量元分为两类,其中,第 1 类重点关注的是程序模块的代码规模和内在复杂度;而第 2 类则重点分析软件开发过程,从分析代码修改特征、开发人员经验、模块间的依赖性以及项目团队组织构架等角度出发来设计度量元.

(2) 缺陷预测模型的构建方法(见第 3 节).

本文将已有的构建方法分为两类,其中,基于机器学习的方法是目前主流的建模方法,根据预测目标的不同,可以进一步细分为分类方法和回归分析方法;而基于缓存的方法则借助缺陷的局部性原理来尝试识别出缺陷模块.

(3) 缺陷预测数据集的相关问题(见第 4 节).

本文从两个角度对缺陷预测数据集相关问题进行分析:首先分析了数据集质量对软件缺陷预测的影响,重点对其中的噪音问题、维数灾难问题和类不平衡问题的产生原因及其相应解决方案进行了分析和总结;随后,针对需要预测的目标项目可能是一个全新项目,或这个项目已有的训练数据较少的问题,分析了利用其他项目的数据集来为目标项目构建缺陷预测模型的可行性,并将该问题称为跨项目缺陷预测问题.然后,从实例选择、实例权重设置、特征映射和度量元选择等角度对基于迁移学习的跨项目缺陷预测方法进行了总结.

2 度量元的设计

挖掘软件历史仓库、设置新颖的与软件缺陷存在强相关性的度量元,是构建高质量缺陷预测模型的关键.因此,度量元的设计一直是软件缺陷预测研究中的一个核心问题[4].早期的研究工作主要集中于分析源代码,重点关注基于软件代码(software code)的软件度量.近些年来,更多的研究工作集中于挖掘不同的软件历史存档,重点关注基于软件开发过程(software process)的软件度量.本节将重点从这两个角度出发,对已有研究工作进行系统的总结.

2.1 基于软件代码的度量

在研究工作的早期阶段,大部分研究工作通过分析软件代码来设计度量元.这类度量元重点关注程序模块的代码规模和内在复杂度等属性,其潜在的假设是:代码规模或复杂度越高的程序模块,其内部含有缺陷的可能性越高.

研究人员最早借助代码行数(lines of code,简称 LOC)进行度量[5],例如,Akiyama 给出了缺陷数(D)与 LOC(L)的关系式:D=4.86+0.018L.但该度量元过于简单,难以合理地去度量软件系统的复杂性.随后,研究人员逐渐考虑了 Halstead 科学度量[6]和 McCabe 环路复杂度(cyclomatic complexity)[7].其中,

- Halstead 科学度量[6]通过统计程序内操作符和操作数的数量来度量代码的阅读难度,其假设是代码的阅读难度越高,则其含有缺陷的可能性也越高,涉及到的主要度量元包括程序的长度、容量、难度和工作量等;
- 而 McCabe 环路复杂度[7]关注的是程序的控制流复杂度,其假设是程序的控制流复杂度越高,则其含有缺陷的可能性也越高.在度量时,首先将程序建模为控制流图(control flow graph),其中,节点对应的是语句,边表示从一个语句到另一个语句的控制流.随后,通过公式 $v(G)=e-n+2$ 计算出控制流图 G 的环路复杂度,其中,e 表示边的数量,n 表示节点的数量.最后,可以进一步计算出程序的基本复杂度(essential complexity)和设计复杂度(design complexity).

随着面向对象开发方法的普及,其特有的封装、继承和多态等特性给传统的软件度量提出了挑战.研究人员提出了适用于面向对象程序的度量元,其中最为典型的是 Chidamber 和 Kemerer 提出的 CK 度量元[8].CK 度量元综合考虑了面向对象程序中的继承、耦合性和内聚性等特征,给定一个类,其包含的度量元名称及相关描述见表 1.

Table 1 CK metrics

表 1 CK 度量元

名称	描述
WMC	类的加权方法数
DIT	类在继承树中的深度
NOC	类在继承树中的孩子节点数
CBO	与该类存在耦合关系的其他类的数目
RFC	该类可以调用的外部方法数
LCOM	类内访问一个或多个属性的方法数

Basili 等人[9]基于一些中等规模的信息管理系统,首次验证了 CK 度量元与程序模块内的缺陷所存在的相关性.随后,Subramanyam 和 Krishnan[10]基于 8 个工业界项目,进一步验证了 Basili 等人的发现.周毓明等人[11]也

对基于面向对象程序的度量元与程序模块缺陷间的相关性进行了深入的分析,随后他们[12,13]发现:类规模度量元在分析时存在潜在的混合效应,并会对缺陷预测模型的性能产生影响.因此,他们提出了一种基于线性回归的方法来尝试移除这种混合效应.最后,他们[14,15]分别对 Sarkar 等人提出的 package-modularization 度量元[16]和基于程序切片的内聚性度量元[17]与程序模块缺陷间的相关性进行了深入分析.

Zimmermann 和 Nagappan[18]通过分析二进制文件间的依赖关系来预测模块部署后缺陷数(post-release defects),他们重点分析了二进制文件之间的数据依赖关系和调用依赖关系,并将子系统建模为依赖图.其中,数据依赖关系从分析变量的定义和变量的使用入手,而调用依赖关系从分析函数的声明和函数的调用入手.基于微软的 Windows Server 2003,他们发现,依赖图中存在的循环(cycle)和团(clique)与部署后缺陷数存在相关性.具体来说:若两个二进制文件之间循环依赖,则称这两个二进制文件构成一个循环.循环内二进制文件内含有的缺陷数大约是其他二进制文件的两倍.若多个二进制文件,两两之间均存在依赖关系,则称这些二进制文件构成一个团.若团的规模越大,则团内二进制文件内含有的缺陷数也越多.基于上述观察,他们借助图论,基于依赖图定义了一系列与拓扑性质相关的复杂性度量元.随后,他们借助社交网络分析中的网络中心度指标来识别出系统内的中心位置程序模块[19].实证研究表明,模块的中心度指标与部署后缺陷数存在相关性.

Shin 等人[20]通过分析方法之间的调用关系来设计度量元.假设方法 x 调用方法 y,则方法 x 称为方法 y 的调用者(caller),而方法 y 称为方法 x 的被调用者(callee).这些度量元的设计动机包括:若一个文件含有的调用者和被调用者数越多,则该文件含有缺陷的可能性越高;若一种方法与更多新的方法之间存在调用或被调用的关系,则该方法含有缺陷的可能性越高;若一个文件的内聚性越低,或耦合性越高,则该文件含有缺陷的可能性越高;若一种方法的调用者或被调用者的修改频率越高,则该方法含有缺陷的可能性越高.

Binkley 等人[21]借助自然语言处理技术,设计了 3 种度量元来对代码的编写良好度进行评估.其中,

- 第 1 个度量元最为简单,其关注的是程序标识符使用的自然语言比例,其设计动机是由自然语言构成的标识符更容易被开发人员理解,因此含有缺陷的可能性更小;
- 第 2 个度量元关注的是标识符的简洁性和一致性,其设计动机是:若标识符中含有的概念含糊不清,则开发人员在进行代码修改的时候更容易引入缺陷;
- 第 3 个度量元则基于 Lawrie 等人[22]提出的 QALP(quality assessment using language processing)评分,该评分通过衡量代码中使用的自然语言和相关注释中使用的自然语言的相关性,来对代码注释的质量进行评估.

2.2 基于软件开发过程的度量

软件缺陷不仅与程序模块的内部复杂度有关,更与代码修改特征、开发人员经验、模块间的依赖性以及项目团队组织构架等因素密切相关.近些年来,随着对软件历史仓库挖掘的日益深入,使得分析软件开发过程并设计出与上述因素相关的度量元成为可能.本小节从上述影响因素出发,对已有的研究工作进行分类和总结.

2.2.1 基于代码修改特征的度量

一些研究人员借助代码搅动(code churn)来预测缺陷密度,通过分析代码的修改历史,代码搅动可以度量指定程序模块在一段时间内的代码修改量.例如,借助文件对比工具 diff,可以自动统计出代码不同版本间新增、删除或修改的代码量.Nagappan 和 Ball[23]设计了 8 种基于相对代码搅动(relative code churn)的度量元,这些度量元使用代码规模或修改花费时间等将程序模块的代码修改量进行归一化处理.基于微软 Windows Server 2003 项目的实证研究表明:基于相对代码搅动的度量元可以很好地用于预测模块内的缺陷密度.同时,其预测效果要显著优于基于绝对代码搅动的度量元.

随后,Moser 等人[24]从文件的修改次数、重构次数、缺陷修复次数、修改涉及的开发人员数、修改的代码行数等角度出发,共设计了 18 种度量元.在实证研究中,他们发现:(1) 若一个文件的修改次数越多,则该文件含有缺陷的可能性也越高;(2) 若一次代码修改涉及的文件越多,则相关文件含有缺陷的可能性也越小,因为代码修改越复杂,开发人员在分析和修改相关代码时也会越谨慎;(3) 为修复缺陷的代码修改容易引入新的缺陷;(4) 代码重构有助于提高代码质量,因此,若一个文件的重构次数越多,则该文件含有的缺陷数也越少.

上述度量元在设计时重点从分析代码的修改量入手,但已有研究人员从分析代码修改过程的复杂度出发来设计度量元.例如,Hassan[25]认为,复杂的代码修改过程会显著降低软件产品的质量.例如,在短时间内,为软件添加大量新的特征或多个开发人员同时对代码进行修改.Hassan 基于信息熵来度量代码修改过程的杂乱(chaotic)程度,并假设代码修改过程的复杂度越高,则代码含有缺陷的可能性也越高.基于上述假设,Hassan 等人设计了 4 种基于代码修改过程复杂度的度量元.随后,D'Ambros 等人[26]设计出了 CHU 度量元和 HH 度量元.其中,CHU 度量元基于 CK 度量元来比较代码每两周的修改量.而 HH 度量元则是对 Hassan 提出的度量元[25]的扩展,即:当特定的代码度量元取值发生变化时,对涉及到的相关文件数进行统计.

2.2.2 基于开发人员的度量

早期的研究仅简单统计修改模块的开发人员数.例如,Graves 等人[27]基于一个大规模电信系统,发现开发人员数对预测缺陷数量的帮助并不显著.随后,Weyuker 等人[28,29]基于 3 个大规模工业系统,也发现开发人员数不是影响缺陷预测模型性能的主要因素.

随后,研究人员进一步从开发人员的经验、程序模块的所有权、开发行为等角度展开了更为深入的研究.Pinzger 等人[30]假设程序模块的工作碎裂度(work fragmentation)与缺陷存在相关性,即:一个程序模块,若参与的开发人员越多,提交的代码修改越多,则该模块的工作碎裂度也越高.他们通过分析开发人员的代码提交信息,构建出开发人员-模块网络(developer-module network),随后,借助社交网络分析中的网络中心度(network centrality)指标计算出模块的中心度取值,并以此度量对应模块的工作碎裂度.实证研究验证了他们的假设,即:相对于周边地区模块,中心位置模块含有缺陷的可能性更高.

Meneely 等人[31]假设开发人员经验与缺陷存在相关性,他们通过分析开发人员间的协作关系构建出开发人员网络(developer network).其中,节点表示开发人员.若两个开发人员之间存在边,则表示他们在同一版本中至少协作修改过一个程序模块.他们同样借助社交网络分析中的网络中心度指标来衡量开发人员经验,并用于预测程序模块的缺陷.

Jiang 等人[32]认为,每个开发人员都具有自己的编码风格、代码提交频率和开发经验等,因此,不同开发人员具有不同的缺陷模式.为验证该猜想,他们通过分析 2005 年~2010 年间的 Linux 内核开发存档,发现其中有一位开发人员,其 48%的代码修改与修复 for 循环相关缺陷有关,而另一位开发人员修复这种缺陷类型的代码修改比例则降到仅为 13.3%.他们认为:虽然已有研究工作考虑了开发人员因素,但并未充分考虑不同开发人员间存在的特征差异,因此,他们针对每个开发人员分别构建出不同的个性化缺陷预测模型.

Bird 等人[33]从分析程序模块的所有权(ownership)入手,模块的所有权可以被定义为开发人员与模块之间存在的代码修改关系.他们将贡献者(contributor)定义为与程序模块存在所有权关系的开发人员,将一个贡献者的所有权比例(proportion of ownership)定义为该贡献者提交的代码修改数与该模块所有提交的代码修改数的比值.基于贡献者的所有权比例,他们将贡献者细分为两类:微量贡献者(minor contributor)和主要贡献者(major contributor).其中,微量贡献者的所有权比例低于 5%,而主要贡献者的所有权比例不低于 5%.基于上述定义,他们设计了 4 种基于所有权的度量元.其中,MINOR 度量元计算出微量贡献者的数量,MAJOR 度量元计算出主要贡献者的数量,TOTAL 度量元计算出所有贡献者的数量,OWNERSHIP 度量元返回提交代码修改数最多的贡献者的所有权比例.基于微软的 Windows Vista 和 Windows 7 两个大规模商业软件,他们发现:(1) 若一个模块的 MINOR 取值越大,则该模块含有的缺陷数越多;(2) 若一个模块的 OWNERSHIP 取值越大,则该模块含有的缺陷数越少;(3) 一个模块的微量贡献者可能是与该模块存在依赖关系的其他模块的主要贡献者;(4) 若移除微量贡献者的信息,会大幅度降低缺陷预测模型的性能.

Rahman 等人[34]则在更细粒度的程序模块(即,仅针对修复缺陷的代码段)上分析了开发人员因素(例如所有权和开发人员经验)与缺陷间的相关性,他们认为:在缺陷预测时,考虑开发人员在特定目标文件上的经验,要比考虑开发人员在整个系统上的经验更为重要.Posnett 等人[35]在分析模块所有权的基础上还进一步考虑了开发人员注意力的集中度,他们发现:注意力越集中(即,一段时间内需要同时执行的任务数越少)的开发人员,其产生的缺陷数也越少.

Lee 等人[36]认为:开发人员的行为交互模式(behavioral interaction patterns)与软件产品质量存在相关性,一些不正常或者意料之外的行为更容易引入缺陷.例如:若一个开发人员在编辑一个源代码文件上花费过多的时间,则在编辑后的源文件内部发现缺陷的可能性会很高.他们通过分析存储在 Eclipse 插件 Mylyn 中的开发人员交互信息,设计了 56 种微交互度量元(micro interaction metric).这 56 种度量元可分为两类:基于文件级别和基于任务级别.其中,

- 基于文件级别的度量元可以在指定任务中,对特定文件上的行为交互进行度量.例如:NumEditEvent 度量元表示在指定任务中,开发人员针对特定文件的编辑次数;
- 而基于任务级别的度量元则对给定任务的属性进行度量,例如:TimeSpent 度量元表示开发人员完成给定任务时,需要花费的总时间.

2.2.3 基于程序模块间依赖性分析的度量

一些研究人员基于软件开发过程,从程序模块间的依赖性角度进行了分析.Bird 等人[37]认为:在构建缺陷预测模型时,除了需要考虑 Zimmermann 和 Nagappan[18]分析的程序模块间的依赖关系以外,还需要进一步考虑开发人员间的协作关系(即:若同一开发人员对两个模块均进行过代码修改,则认为这两个模块间存在链接关系).因此,他们通过融合依赖图[18,19]和开发人员网络[31],提出了 socio-technical 网络这一混合模型.基于微软的 Windows Vista 项目和开源项目 Eclipse 的实证研究表明,混合模型构建出的缺陷预测模型具有更高的查准率和查全率.

Hu 和 Wong[38]对 Bird 等人[37]的研究工作进行了拓展,主要针对 socio-technical 网络中的关系强度与部署后缺陷数的相关性进行了分析.他们认为:关系强度可以有效地反映出模块之间以及模块与开发人员之间的耦合程度,一般来说,耦合性越高意味着软件设计越差.他们通过主题模型中的引用影响模型(citation influence model)来度量这种关系强度.实证研究表明:弱关系强度和强关系强度与部署后缺陷数具有不同的相关性,因此,对其区分对待可以有效地提高模型的预测性能.

除此之外,一些研究人员从代码修改角度对模块间的依赖性进行分析.D'Ambros 等人[39]发现,修改耦合性(change coupling)与程序模块缺陷存在相关性.修改耦合性可以有效地反映出多个软件制品(例如类)在软件演化过程中经常发生同时修改的隐性关系,例如:当一个开发人员修改完一个类后,因为与该类存在修改耦合性的其他类可能会放置在不同的包(package)内,会造成开发人员忘记对这些类进行相应修改并引入缺陷.他们通过分析代码修改历史,从不同角度定义了 4 种基于修改耦合性的度量元.

Herzig 等人[40]同样认为:修改完一个模块后,可能会引发对其他模块的修改行为.他们用修改系谱图(change genealogy)来描述修改间的依赖关系.修改系谱图是一个有向无回路图,可以捕获早期的代码修改集如何引发后续的代码修改集.两个修改集的依赖关系可以通过分析修改集中新增和删除的方法定义和方法调用来获取.如果修改集 CS_N 和之前的修改集 CS_M 存在依赖关系,则它们之间至少存在如下 4 种情况之一.

(1) 在 CS_N 中,删除了在 CS_M 中新增的方法定义;
(2) 在 CS_N 中,新增了在 CS_M 中删除的方法定义;
(3) 在 CS_N 中,调用了在 CS_M 中新增的方法;
(4) 在 CS_N 中,删除了在 CS_M 中新增的方法调用.

随后,他们基于修改系谱图,从不同角度分别设计出基于 EGO 网络的度量元、基于 GLOBAL 网络的度量元和基于结构洞(structural hole)的度量元.基于 4 个开源项目,他们发现:与基于代码依赖性的度量元[19]相比,基于修改系谱图的度量元的查准率更高;若同时使用这两种度量元,虽然可以进一步提高查全率,但同时也会降低查准率.

2.2.4 基于项目团队组织构架的度量

一些研究人员从分析企业的组织构架入手,一个软件项目经常由企业内不同团队协作完成,因此,企业组织构架对软件产品质量的影响非常显著.Conway's Law[41]中有一个推论:软件系统的架构可以有效地反映出开发公司的沟通架构.为了验证上述推论,Nagappan 等人[42]首次从组织构架角度出发对缺陷预测展开了研究.他们

通过设计 8 种度量元来衡量组织构架的复杂性,针对某一组件,这些度量元综合考虑了开发该组件的开发人员数量、辞职人员数量、修改该组件的总次数和相关开发人员间的组织距离等.基于微软的 Windows Vista 项目的实证研究表明:相对于传统的度量元,上述度量元在缺陷预测上具有更高的查准率和查全率.

Mockus[43]则分析了组织构架变动过程中与开发人员相关的度量元,他们发现:离开的开发人员数量对软件质量存在显著影响,因为这些人员的相关业务知识和开发经验很难在短期内顺利交接给其他开发人员.而新加入的开发人员数量对软件质量的影响较少,可能因为企业很少会为新来的开发人员立即分配重要的开发任务所致.除此之外,组织规模对软件质量也存在显著影响,因为随着组织规模的扩大,会大幅度增加沟通和协调的开销并显著降低组织制定决策的速度.

Bird 等人[44]则猜测:由于文化障碍、知识转移困难以及沟通和协调的开销较大,会使得基于分布式的开发方式(即,编写同一模块的开发人员分布在世界各地)比基于集中式的开发方式更容易引入缺陷.但令人意外的是,基于微软 Windows Vista 项目的实证研究表明,该猜测并不成立.

2.2.5 基于其他角度的度量

除此之外,也有研究人员从其他角度来设计度量元.

Bacchelli 等人[45]从分析开发人员间发送的电子邮件入手.在实际开发过程中,开发人员经常借助电子邮件对程序模块的缺陷修复、代码重构,甚至项目的设计决定等进行讨论.他们首先借助程序模块在电子邮件中的讨论次数来度量该模块的流行度;随后,通过挖掘电子邮件存档来对程序模块的流行度进行度量,并假设开发人员会在电子邮件中更多地讨论有缺陷的程序模块;最后,他们基于程序模块流行度设计出了 5 种不同的度量元.

Taba 等人[46]从分析反模式(antipattern)入手.反模式提供了一些特定的设计和实现风格,可以告诉开发人员,哪些设计选择是好的,哪些设计选择是不好的;对于不好的设计,如何通过代码重构进行改进.因此,反模式通过识别出不好的设计,可以降低未来产生缺陷的风险.他们基于反模式分析设计出了 4 种相关度量元.基于开源项目 Eclipse 和 ArgoUML 的实证研究表明:若文件存在反模式,则文件缺陷密度会更高.同时,基于度量元 ANA, ACM 和 ARL 构建的缺陷预测模型要优于基于传统度量元构建的模型.

Herzig[47]从分析测试用例的执行结果入手,基于测试用例的执行信息设计了多种度量元,这些度量元可以简单地分为基于测试用例数的度量元、基于质量门数的度量元、基于测试用例属性的度量元、基于测试用例失效迸发(test failure burst)的度量元和基于代码审查的度量元.基于微软 Windows 8 项目的实证研究表明,这些度量元可以很好地对部署后缺陷进行预测.

王青等人[48]从需求分析角度入手,他们首先将被测项目建模为需求依赖网络(requirement dependency network),其中,节点表示需求,边表示需求间的依赖关系.他们重点考虑了 3 类依赖关系:precondition 依赖关系表示当一个功能完成或一个条件满足后,另一个功能才能被执行;constraint 依赖关系表示需求间存在相关性或约束性;similar_to 依赖关系表示一个需求间的描述和另一个需求间的描述相似或部分重叠.随后,他们基于网络分析提出一系列度量元,并在两个商业软件项目中验证了这些度量元可以很好地预测出模块内的缺陷数.

2.3 不同类型的度量元间的比较

通过第 1.1 节和第 1.2 节的分析不难看出:基于软件代码(code)或开发过程(process)的度量元均可有效地构建出缺陷预测模型,但这两类不同类型的度量元的关注角度并不一样,因此,哪一类更能有效地构建出缺陷预测模型吸引了一些研究人员的关注.受选择的数据集、缺陷模型的构建方法或模型的评测指标等因素的影响,研究人员在实证研究中得到了一些不同的结论.

Graves 等人[27]基于一个电话交换系统发现,基于 process 的度量元比基于 code 的度量元更为有效.而 Menzies 等人[49]则基于 NASA 数据集发现,仅仅基于 code 的度量元也能够构建出高质量的缺陷预测模型.张洪宇[50]甚至认为:仅简单考虑 LOC 度量元,也能很好地对模块内的缺陷数进行预测.

随后,Moser 等人[24]将基于 code 的度量元构建出的缺陷预测模型称为代码模型(code model),基于 process 的度量元构建出的模型称为修改模型(change model),而基于所有度量元构建出的模型称为组合模型(combined model).基于 Eclipse 项目的 3 个版本,他们发现:无论采用哪种机器学习方法,修改模型的性能要优于代码模型;

而混合模型的性能与修改模型的性能接近,且不具有显著优势.他们通过一些简单实例对上述研究发现进行了解释,例如:虽然一个程序模块较为复杂,但是如果相关开发人员的编程经验较为丰富,并且在实现模块时认真、严谨,则该模块含有缺陷的可能性将会较低;但如果采用基于code度量元进行建模时,则该模块将可能会被误分类为 FP 模块.同样,如果一个程序模块在开发过程中经常被修改,则该模块含有的缺陷可能性将很高,而与该模块的代码复杂度无关.当然,上述解释并不是为了完全否定基于 code 的度量元与模块缺陷间的相关性.

Arisholm 等人[51]同样分析了不同类型的度量元对模型预测性能的影响,他们基于一个 Java 中间件遗留系统 COS 发现:若选择 AUC 值作为性能评测指标,则基于 code 的度量元更为有效;但若考虑缺陷预测模型的成本效益,则基于 code 的度量元并不一定更为有效.

Rahman 和 Devanbu[52]则从不同角度出发,他们认为:很多项目会随着软件版本的迭代,持续、动态地调整自己的软件开发方法和软件质量保障方法.因此他们认为,基于软件版本来评估缺陷预测模型的性能更为合理.基于 12 个大规模开源项目,他们对基于 code 的度量元和基于 process 的度量元进行了比较.最终得到的结论与 Moser 等人[24]的结论保持一致,即:无论使用哪种机器学习方法,基于 process 的度量元更为有效.除此之外,他们还发现,基于 process 的度量元具有一定的停滞性(stagnation).其中,度量元的停滞性是指随着版本的迭代,度量元取值变化不大,因此会造成同一程序模块会被缺陷预测模型持续预测为 FP 模块.

基于上述分析不难看出,基于 code 的度量元和基于 process 的度量元之间存在一定的互补性.Madeyski 和 Jureczko[53]在基于 code 的度量元的基础上同时考虑了基于 process 的度量元,实证研究结果表明,这种方式可以进一步提高缺陷预测模型的性能.

3 缺陷预测模型的构建方法

3.1 基于机器学习的方法

目前,大部分研究工作都基于机器学习的方法来构建缺陷预测模型.其中,常见的模型预测目标可以大致分为两类:一类是预测程序模块内含有的缺陷数或缺陷密度,另一类是预测程序模块的缺陷倾向性.缺陷预测目标的选择一般与程序模块的粒度设置有关,若程序模块设置为细粒度(例如类级别或文件级别),则以预测模块的缺陷倾向性为目标,常采用的是分类方法,包括 Logistic 回归、朴素贝叶斯和决策树等,针对这类方法的性能评估指标包括查准率、查全率、*F*-measure 或 AUC 取值等;若设置为粗粒度(例如包级别或子系统级别),则以预测模块内的缺陷数或缺陷密度为目标,常采用的是回归分析方法.具体来说,将度量元设置为自变量,将模块内的缺陷数或缺陷密度设置为因变量.回归分析尝试构建出有效模型,可以根据自变量取值来预测出因变量取值.其中,相关系数是度量变量之间相关强度的统计量:若变量之间呈线性关系,则使用 Person 相关系数;若变量之间呈非线性关系,则使用 Spearman 秩相关系数[18,23,30].相关系数的取值范围介于–1~1 之间,取值越接近于 1,表示正相关性越高;而取值越接近于–1,表示负相关性越高.多元线性回归是常用的建模方法,在构建模型时,根据变量的选择方式可以分为 3 类建模方法:前向选择法、后向移除法和逐步回归法.其中,前向选择法会从模型外的自变量中每次选出与因变量最为显著的自变量添加到模型中,直至模型外没有统计显著性的自变量为止;后向移除法会每次从模型内移除对模型贡献最不显著的自变量,直至模型内所有自变量都具有统计显著性;而逐步回归法是上述两种方法的综合,前两步与前向选择法一致,但当模型新增一个自变量时,会对模型内的所有变量进行重新考察,若发现有自变量对模型的贡献不显著时,则移除该变量.当完成模型构建后,借助判断系数 R^2(或校正后的 R^2)来评价模型对数据的拟合优度[18,23,30],其取值范围介于 0~1 之间,其取值越接近于 1,表示模型对数据的拟合程度越好.

虽然已有研究结果表明,基于机器学习的方法都可以获取较好的预测性能,但由于项目内在特征、机器学习方法内在参数取值以及评测指标的不同,因此不存在一种缺陷预测模型的构建方法可以在所有项目中均获得最优性能[2,54−60].

一些研究人员尝试着对不同机器学习方法的性能进行相互比较.例如,Elish 等人[55]基于 NASA 数据集,将支持向量机与其他 8 种机器学习方法进行了系统比较.结果表明:支持向量机在总体上要优于其他 8 种方法,尤

其是在选择查全率作为评测指标的时候.Catal 和 Diri[60]基于 NASA 数据集,以 AUC 值作为评测指标,深入分析了数据集规模、度量元和特征子集选择方法对缺陷预测模型性能的影响.结果表明:随机森林法在大规模数据集上性能最优,而朴素贝叶斯则在小规模数据集上的性能最优.除此之外,在基于人工免疫系统(artificial immune system)的一系列分类方法中,AIRS2Parallel 算法在基于方法级的度量元上性能最优,而 Immunos 2 算法在基于类级别的度量元上性能最优.Lessmann 等人[56]基于 NASA 数据集,以 AUC 值作为评测指标,系统地比较了 22 种不同的机器学习方法,这些方法主要分为 6 类:统计方法、最近邻方法、神经网络法、支持向量机法、决策树法和集成方法.他们发现,最优的 17 种机器学习方法间的性能差异并不显著.随后,Ghotra 等人[57]同样基于 NASA 数据集,并考虑了更多新颖的机器学习方法.他们对 Lessmann 等人的实验过程[56]进行了重现,得到了相同的结论.但在基于去除噪音的 NASA 数据集[59]和来自 Promise 库的 10 个开源项目数据集上他们发现,不同的机器学习方法之间存在显著的性能差异.Shepperd 等人[58]则借助随机效应 ANOVA 模型,通过分析已有的研究成果,对影响缺陷预测模型性能的影响因素进行了分析.他们考虑的影响因素包括机器学习方法、数据集、度量元以及研究小组.他们意外地发现:机器学习方法的选择对性能的影响并不显著,但不同研究小组之间却存在显著的差异.

Menzies 等人[49]和宋擒豹等人[61]对通用缺陷预测框架展开了深入的研究.Menzies 等人[49]首先提出了一种软件缺陷预测框架,即:先对数值型属性进行取对数操作,随后进行特征子集选择,最后选择一种机器学习方法.其中,取对数操作主要是针对一些取值呈指数分布的数值型度量元,预处理后其取值将接近均匀分布.而特征子集选择可以有效地移除数据集中的冗余特征和无关特征.基于 NASA 数据集的实证研究表明:他们的方法可以获得较好的预测性能,但同时他们也发现,不同数据集上选出的最优特征子集并不一致.因此 Menzies 等人认为,寻找适用于所有项目的最优特征子集不具有研究意义.随后,宋擒豹等人[61]对 Menzies 等人[49]提出的软件缺陷预测框架进行了扩展,他们提出的框架主要包括两个阶段:首先,基于训练数据选出最优方案;其次,基于最优方案构建出缺陷预测模型并用于预测未知数据.其中,每个方案同样由数据预处理方法(即,是否执行取对数操作)、特征子集选择方法和机器学习方法确定.他们在实证研究中共考虑了 12 种不同的方案,结果表明:不存在一种方案,可以在任何数据集上均获得最优性能.

除此之外,一些研究人员尝试借助目前机器学习领域的最新研究进展,例如主动学习(active learning)和半监督学习(semi-supervised learning)等.黎铭等人[62]借助基于半监督学习和主动学习的采样方法来从大型软件系统中选出少量代表性程序模块进行标记,并随后构建缺陷预测模型.其中,CoForest 方法是一种基于半监督学习的采样方法,该方法基于随机森林分类法,借助随机采样来找出最优的采样实例;而 ACoForest 方法则通过进一步使用主动学习来对 CoForest 方法进行扩展.实证研究表明:这两种方法均要优于基于传统机器学习的采样方法.Lu 等人[63]基于主动学习来构建缺陷预测模型,即:在模型的训练过程中,不仅需要考虑前一软件版本的数据集,而且还需要从当前软件版本中选择出少量实例进行标记.随后,他们借助降维(dimensionality reduction)和特征子集选择方法来进一步提高模型性能.

3.2 基于缓存的方法

虽然大部分研究工作都基于机器学习方法来构建缺陷预测模型,但也有研究人员从缺陷的局部性原理入手来识别出其中的 FP 程序模块.我们将这类方法统一称为基于缓存的方法.

Hassan 和 Holt[64]最早对这一类方法进行了研究,他们设计出了“the top ten list”方法.该方法基于动态更新策略,以确保缓存中始终包含前 10 个最有可能含有缺陷的子系统.缓存中,子系统的动态更新策略基于一些启发式设定,例如优先选择最近代码修改次数最多的、最近缺陷修复次数最多的或最近刚修复缺陷的.随后,Kim 等人[65]和 Rahman 等人[66]对这类方法进行了更为深入的研究.

Kim 等人[65]设计出了 BugCache 方法,该方法主要考虑了如下的缺陷局部性原理:

(1) 修改模块局部性,即:一个程序模块,若最近被修改过,则将来可能还会含有缺陷;

(2) 新增模块局部性,即:一个程序模块,若是最近新增加的,则将来可能还会含有缺陷;

(3) 时间局部性,即:一个程序模块,若最近检测出一个缺陷,则将来也会含有其他缺陷;

(4) 空间局部性,即:一个程序模块,若最近检测出一个缺陷,则将来它附近的程序模块可能也会含有缺陷.

BugCache 方法的执行过程可简单总结如下:首先,在缓存内会预先加载一些代码规模最大的程序模块;随后,当提交对模块内缺陷进行修复的代码(bug fixing change)后,BugCache 方法会在缓存内查看是否包含该程序模块,若不存在,则根据时间局部性和空间局部性,将该模块和附近的模块加载到缓存中;随后,若存在一些新增或修改过的程序模块,则该方法也会将这些程序模块加载到缓存中;最后,该方法会采用特定的缓存置换策略(例如最近最少使用策略)来从缓存中移除掉一部分程序模块,以确保缓存内部包含的程序模块数始终保持不变.他们基于7个开源项目发现:假设缓存仅能容纳10%的程序模块,则若将程序模块粒度设置为文件,BugCache 方法可以达到 73%~95%的预测精度;而若将程序模块粒度设置为函数,则可以达到 46%~72%的预测精度.

Kim 等人采用命中率(hit rate)指标对 BugCache 方法的有效性进行评估,该指标可以统计出在项目开发过程中,缓存成功命中缺陷模块的累计比例.Rahman 等人[66]认为:若 BugCache 方法经常加载一些大规模的程序模块,虽然可以提高命中率,但也会提高代码的审查成本;相反,若 BugCache 方法经常加载一些小规模的程序模块,虽然命中率会下降,但可能会有机会审查到更多缺陷密度高的程序模块,从而提高代码审查的成本效益.因此,他们从代码审查成本和缺陷密度角度出发,对 BugCache 方法的有效性进行了重新评估.最终研究发现:(1) 虽然 BugCache 方法倾向于加载规模大的程序模块,但缓存内模块的缺陷密度要高于缓存外模块的缺陷密度;(2) 不同的缓存置换策略,对 BugCache 方法的性能影响并不显著;(3) 他们提出了一种简单方法,即:将程序模块按照关闭的缺陷数从大到小进行排序,并依次将模块加载到缓存,直至缓存内代码总规模比例达到 20%为止.他们发现:该方法与 BugCache 方法相比,并不存在显著的性能差异.

Lewis 等人[67]基于谷歌公司内部的两个成熟项目对上述两种方法[65,66]的有效性进行了验证,他们发现,开发人员更倾向于使用 Rahman 等人[66]提出的简单方法.随后,为了将软件缺陷预测更好地与企业的实际开发流程相结合,他们通过与开发人员的沟通与交流,总结出,一种好的缺陷预测方法需要具备如下的特征:

(1) 当预测出一个程序模块可能含有缺陷时,需要提供一些修复缺陷的线索;
(2) 当预测出一个程序模块可能含有缺陷时,需要给出有说服力的预测依据;
(3) 预测出的 FP 模块需要偏向于新增模块或最近修改过的模块;
(4) 通过支持并行计算,可以更为迅速地返回程序模块的缺陷预测结果;
(5) 方法中的缓存规模可以根据项目的实际情况进行灵活设置.

基于上述特征分析,他们对 Rahman 等人[66]提出的方法进行了相应扩充,并集成到谷歌的现有开发过程中.但出乎意料的是,扩充后的方法并不能显著改变开发人员的行为.

4 缺陷预测数据集的相关问题

这一节从两个角度对缺陷预测数据集的相关问题进行分析.

- 首先分析数据集质量对缺陷预测性能的影响.以研究人员经常使用的 NASA 数据集为例,Shepperd 等人[59]分析了该数据集的两个不同版本(分别来自 Promise 库和 NASA 网站),发现这些数据集中存在诸多质量问题,例如部分数据取值遗失或不一致以及一些实例存在重复等.除此之外,他们还发现,不同版本的数据集对实证研究结论存在显著的影响.因此他们认为:研究人员在选用 NASA 数据集时,需要明确使用的数据集版本并详细介绍数据集预处理步骤,以确保实证研究可以重现.我们依次对缺陷预测数据集中的噪音问题、维数灾难问题和类不平衡问题的产生原因及其相应解决方法进行分析和总结;
- 其次,针对需要预测的目标项目可能是一个全新项目或这个项目已有的训练数据较少的问题,我们分析了利用其他项目的数据集为目标项目构建缺陷预测模型的可行性,并随后从实例选择、实例权重设置、特征映射和度量元选择等角度对基于迁移学习的缺陷预测方法进行了总结.

4.1 数据集中的噪音问题

在挖掘软件历史存档时,在对程序模块进行类型标记和软件度量时均可能产生噪音,这些噪音的存在会影响到缺陷预测模型的构建,并会对已有实证研究结论的有效性产生严重影响.以图 2 所示的程序模块类型标记

过程为例,我们分析其中噪音产生的原因及相应的解决方法.

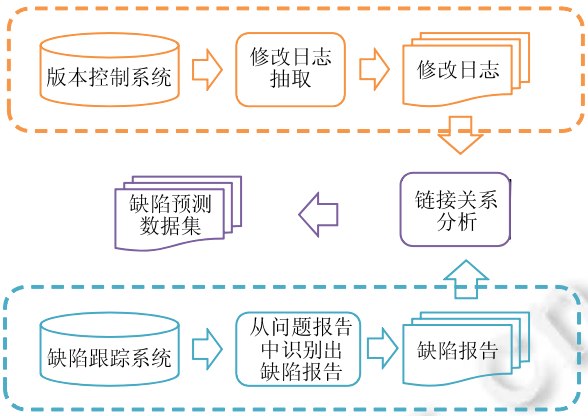

Fig.2 Class labeling process for program modules

图 2 程序模块类型的标记过程

首先,将版本控制系统中的修改日志与缺陷跟踪系统中的缺陷报告建立链接关系时容易产生噪音.例如:通过在修改日志中搜索特定关键词(例如 fixed 或 bug)和缺陷 ID 号(例如#53322)可以去建立这种链接关系,随后,基于该链接关系将代码修改涉及的程序模块标记为有缺陷模块.但在实际的软件开发过程中,开发人员在修复完缺陷并提交代码修改时,经常会忘记在修改日志中输入这次修复的缺陷 ID 号,从而造成一些链接关系的遗漏,这种遗漏会造成一些有缺陷模块被误标记为无缺陷模块,我们称这种噪音为假阴性(false negative)噪音.Bird 和 Bachmann 等人[68,69]通过分析一些开源项目,发现大约 54%的已修复缺陷并未与对应的修改日志建立链接关系,因此造成数据集内含有大量的假阴性噪声.除此之外,他们基于 Kim 等人提出的 BugCache 方法[65]进一步分析了含有噪音的数据集对缺陷预测方法的影响,最终结果表明,含有假阴性噪音的数据集会显著降低缺陷预测方法的性能.Nguyen 等人[70]同样在一个商业项目中也发现了上述问题.

Wu 等人[71]尝试去重新发现这些遗漏的链接关系,他们通过分析已确定的链接关系发现了一些特征.例如:缺陷报告中,缺陷修复的时间与代码修改的提交时间较为接近;提交缺陷修复代码的开发人员一般也是缺陷报告中要求修复该缺陷的开发人员;代码修改日志与对应的缺陷报告会存在描述相似的文本.基于上述特征,他们设计了一种自动化的链接关系发现方法 ReLink,在该方法中,若一些未知链接关系满足上述特征,则认为该链接关系是有效的.

但在实际开发过程中,缺陷报告提交者一般从用户角度出发,更多描述的是失效现象和失效重现步骤等,而修改日志提交者一般从开发人员角度出发,更多描述的是修复缺陷的方法,因此造成了修改日志和相应的缺陷报告存在文本相似度不高的问题,并影响到 ReLink 方法[71]的预测准确率.针对该问题,Nguyen 等人[72]提出了一种多层链接关系发现方法 MLink,该方法不仅考虑了与修改日志相关的文本特征,而且还考虑了与修改日志相关源程序的程序代码特征.通过分析已确定的链接关系,他们寻找出修改日志中词项与修改代码中实体或组件名称的关联关系,并基于上述关联关系推导出修改日志和缺陷报告间是否存在链接关系.

Rahman 等人[73]则从另一个角度出发,深入分析了缺陷预测数据集的假阴性噪音和规模对缺陷预测模型性能的影响.实证研究发现:数据集的假阴性噪音和规模对性能均存在重要影响,当选择一些基于信息检索的评价指标(例如 AUC 和 *F*-measure)时,数据集规模的影响程度要超过假阴性噪音.因此他们认为:即使数据集内部存在假阴性噪音,但通过尝试提高数据集的规模,仍不失为一种提高缺陷预测模型性能的有效方法.

其次,在缺陷跟踪系统中,如果问题报告(issue report)存在错误分类也会产生噪音.Herzig 等人[74]手工检查了来自 5 个开源项目的 7 000 多份问题报告,他们发现,大部分的问题报告均被分类为缺陷报告(即,为了修复缺陷).但深入分析之后也发现,其中 33.8%的问题报告存在误分类问题.即:其提交的问题报告不是为了修复缺陷,

而是为程序模块添加新的特征、修改相应文档或重构内部代码等.这些被误分类的问题报告会造成一些无缺陷模块被误标记为有缺陷模块,我们称这种噪音为假阳性(false positive)噪音.他们提出了一套分类指南,可以有效提高对这些问题报告的分类准确率.

通过上述分析不难看出,缺陷数据集中的噪声问题难以避免.Kim 等人[75]采用随机方式,手工注入假阳性噪声和假阴性噪声,并对已有缺陷预测方法的噪声抗干扰能力进行了分析.他们在实证研究中发现:(1) 当数据集中的缺陷模块足够多时,增加噪音并不会显著降低已有方法的预测性能;(2) 已有方法对假阴性噪声的抗干扰能力更强一些;(3) 当数据集中包含的假阳性和假阴性噪声的总比例不超过 20%~35%时,已有方法的性能不会显著下降.除此之外,他们提出一种基于聚类分析的方法,可有效地检测并移除噪声.Tantithamthavorn 等人[76]认为,Kim 等人采用随机方式来为数据集注入噪音并不合理.他们基于 Herzig 等人[74]的研究工作发现:因问题报告误分类产生的噪音并不是随机出现的.同时,不同的噪音注入方式对最终的实证研究结果存在显著的影响.

4.2 数据集中的维数灾难问题

在程序模块度量时,若考虑了大量与代码或开发过程相关的度量元,会使得一些数据集存在维数灾难问题(the curse of dimensionality).因此,需要提出有效方法来重点识别出数据集中的冗余特征(在软件缺陷预测问题中,特征指度量元)和无关特征.其中,冗余特征大量或完全重复了其他单个或多个特征中含有的信息,而无关特征则对采用的数据挖掘算法不能提供任何帮助.研究结果表明:无关特征和冗余特征的存在会提高缺陷预测模型的构建时间,并会降低模型的预测性能.

特征子集选择(feature subset selection)是解决维数灾难问题的一种有效方法,通过识别并移除数据集中的冗余特征和无关特征,来确保选出最为有用的特征子集以构建缺陷预测模型.目前常见的特征选择方法可以简单地分为两类:包装(wrapper)法和过滤(filter)法.其中,包装法借助预先指定的学习算法的预测精度来决定选择出的特征子集,因此,虽然精度较高,但计算开销较大;过滤法则通过分析数据集来完成特征的选择,因此与选择的学习算法无关,具有更好的通用性且计算开销较小,但精确度较低.

Menzies 等人[49]和宋擒豹等人[61]提出的通用缺陷预测框架中均包含了基于包装法的特征子集选择方法,在特征子集搜索时,分别考虑了两种不同的搜索方式,即,前向选择策略和后向选择策略.Gao 等人[77]针对一个大规模遗留电信软件系统,考虑了一种混合特征选择方法,在该方法中考虑了不同的特征排序评估方法和特征子集评估方法.最终结果表明:移除 85%的特征并不会大幅度降低缺陷预测效果,有时甚至会提高预测效果.

我们也基于过滤法提出一种新颖的基于聚类分析的特征子集选择框架 FECAR[78].FECAR 首先根据特征间的相关性将特征进行聚类分析,这样可以将具有冗余关系的特征集中于同一聚类中;随后,对于每一个聚类,根据特征与类间的相关性从高到低进行排序,并从中选出指定数量的特征,该阶段可以有效地避免选出无关特征.Wang 等人[79]借助遗传算法来同时进行特征子集选择和数据集中异常值的检测.他们在设定适应值函数的时候,同时考虑了聚类准则和特征子集质量.与上述研究工作不同,Turhan 和 Bener[80]认为,可以借助特征抽取(feature extraction)方法来缓解维数灾难问题.特征抽取会根据原有的特征集来重构出新的特征集,新的特征具有更强的代表性,并且相互间冗余度更低.他们考虑了两种特征抽取方法——PCA(principal component analysis)法和 Isomap 法.

4.3 数据集中的类不平衡问题

类不平衡问题是软件缺陷预测训练数据集中普遍存在的问题,其产生的根源在于缺陷在软件模块中的分布大致符合二八原则,即,80%的缺陷集中分布于 20%的程序模块内.因此在软件缺陷预测中,无缺陷程序模块(即多数类)的数量要远超过有缺陷程序模块(即少数类)的数量,因此会造成模型对少数类的预测精度(例如查准率和查全率)较低.同时,在缺陷预测模型的构建过程中需要考虑不同预测错误类型的开销,其中,错误类型 I 是将 FP 模块错误地预测为 NFP 模块,该错误类型将造成测试人员不会对 FP 模块进行必要的代码审查和测试,并最终将含有缺陷的软件产品推向市场;错误类型 II 是将 NFP 模块错误地预测为 FP 模块,该错误类型会造成测试人员对 NFP 模块进行大量的代码审查和测试,因此造成测试资源的浪费.显然,第 1 类错误会给企业带来更大的

损失.

类不平衡学习[81]是当前机器学习中的一个热点研究领域,可以有效缓解数据集中的类不平衡问题.已有方法可以大致分为两类.

一类是从调整数据集中的类分布角度出发,经典方法包括随机过采样(random oversampling)法、随机欠采样(random undersampling)法和 SMOTE(synthetic minority oversampling technique)法.Menzies 等人[82]在实证研究中采用一些随机采样方法进行数据预处理,他们发现,移除数据集中的一部分实例并不会显著降低缺陷预测模型的性能.Shatnawi[83]基于模块内的缺陷数来对少数类中的实例进行随机过采样.实证研究结果表明,该方法要优于 SMOTE 法.Pelayo 和 Dick[84]则在实证研究中借助 SMOTE 方法进行数据预处理,他们发现:应用 SMOTE 法后,可以将模型性能至少提高 23%.

另一类从修改学习算法角度出发,即:通过修改算法的训练过程,使得学习算法可以针对少数类取得更好的预测精度,典型方法包括代价敏感学习(cost-sensitive learning)方法.代价敏感学习方法的目标并不是简单地使得预测错误数最少,而是使得错误分类开销总和最小,因此更符合实际需求.Moser 等人[24]将代价敏感学习方法引入到缺陷预测研究,并对基于 code 的度量元和基于 process 的度量元的效果进行了对比.Wang 和 Yao[85]进一步深入分析了 3 类不同的类不平衡学习方法在缺陷预测中的应用,主要包括随机欠采样方法、成本敏感分类方法(阈值移动法)和集成学习方法(包括 AdaBoost.NC 和 SMOTEBoost).他们发现:总体来说,AdaBoost.NC 方法性能最优.为进一步提升 AdaBoost.NC 方法的性能,他们对该方法进行了扩充,即:在模型训练时,可以自动确定方法内在参数(例如,实例采样率和类预测错误开销)的取值.Zheng[86]基于神经网络分析了 3 种代价敏感的 Boosting 算法在缺陷预测中的应用,其中,第 1 种算法通过设置分类阈值,使得预测模型倾向于提高缺陷模块的分类准确性;而后两种算法将预测错误类型的开销融合到 Boosting 过程中的权重更新规则中,使得被误分类的缺陷模块在下一轮中可以被赋予更高的权重取值.

姜远等人[87]提出了一种半监督学习方法 ROCUS,该方法通过同时利用未标记实例和随机欠采样方法来解决类不平衡问题.荆晓远等人[88]发现:不同的程序模块间具有相似性,即,一个程序模块可以被其他少量模块近似表示.因此,他们将有监督字典学习(dictionary learning)方法应用到缺陷预测中以缓解类不平衡问题.具体来说,该方法基于数据集学习出多个字典(包括针对 FP 模块和 NFP 模块的 sub-dictionaries 以及针对所有模块的 total dictionary)和稀疏表示系数(sparse representation coefficient).除此之外,他们进一步将不同错误类型的开销考虑在内,提出了一种成本敏感的有区别字典学习方法.

Rodriguez 等人[89]对不同类型的类不平衡学习方法进行了系统比较,考虑的方法包括随机采样方法、代价敏感的学习方法、集成学习方法以及混合方法.他们发现:虽然上述类不平衡学习方法可以有效地提高缺陷预测模型对少数类的预测精度,但这种提高受到了诸多因素的影响,包括数据集的特征、类不平衡程度以及数据集中重复或不一致数据的移除情况等.Seiffert 等人[90]深入分析了缺陷预测数据集中的类不平衡问题和噪音问题对分类方法和随机采样方法的影响,在实证研究中,他们考虑了 11 种分类方法和 7 种随机采样方法.结果表明,朴素贝叶斯和支持向量机这两种分类方法以及随机欠采样方法对上述两个问题具有更好的鲁棒性.

一些研究人员进一步尝试同时解决类不平衡问题和维数灾难问题.Khoshgoftaar 和 Gao[91]首先使用基于排序的特征选择法,然后,借助随机欠采样法来缓解类不平衡问题.结果表明:当使用随机欠采样法后,可以有效提高特征选择的效果.张道强等人[92]提出了一种两阶段成本敏感的学习方法,即,在特征子集选择阶段和缺陷预测模型构建阶段均考虑了不同错误类型的开销.他们认为:在特征选择阶段,若考虑不同错误类型的开销,会更容易选出与少数类存在强相关性的特征.同样,我们也提出了一种两阶段数据集预处理方法[93],即:在特征子集选择阶段使用基于聚类分析的特征子集选择方法,随后,在类不平衡学习阶段借助随机欠采样方法来保持有缺陷实例和无缺陷实例的平衡.Laradji 等人[94]则从集成学习方法入手,他们认为,集成学习方法(例如随机森林方法)受数据集中类不平衡问题的影响较小.同样,集成学习方法内部的投票机制也可以有效缓解数据集中无关特征或冗余特征的影响.因此,他们提出了一种 APE(average probability ensemble)学习方法,其内部包含了 7 种不同类型的分类方法.随后,通过融合特征子集选择方法来进一步提高缺陷预测性能.

4.4 目标项目无训练数据的问题

已有研究工作大部分集中于同项目缺陷预测(within-project defect prediction),即:基于同一项目的部分数据来构造缺陷预测模型,并用剩余数据来评估模型的性能.但在实际软件开发过程中,需要进行缺陷预测的项目(即目标项目)可能是一个全新的项目,或这个项目已有的训练数据较少.一种解决方法是使用其他项目(即源项目)的训练数据,但不同项目所处的应用领域、采用的编程语言或开发过程并不相同,所以其研究的挑战性在于源项目与目标项目的数据集不满足独立同分布的假设.因此,如何有效迁移源项目的知识来为目标项目构造缺陷预测模型,吸引了研究人员的关注,并将该问题称为跨项目缺陷预测(cross-project defect prediction).两者的区别如图 3 所示.

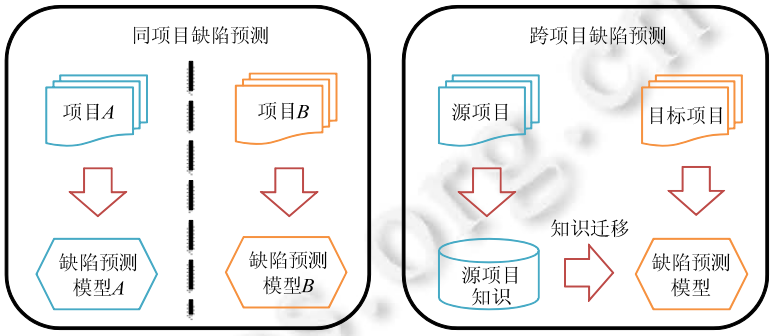

Fig.3 Difference of within-project defect prediction and cross-project defect prediction

图 3 同项目缺陷预测与跨项目缺陷预测的区别

研究人员首先对跨项目缺陷预测的可行性展开了调研.Briand 等人[95]尝试将基于开源项目 Xpose 构建的模型应用于另一个开源项目 Jwriter,他们发现,构建出的模型性能要优于随机模型.Zimmermann 等人[96]对跨项目缺陷预测的可行性进行了更为深入的分析,他们对同属浏览器软件领域的 Firefox 项目和 Internet Explorer 项目进行了分析,发现这两个项目因厂商的开发流程不同造成度量元取值的差异性较大.在进行跨项目缺陷预测时,基于 Firefox 项目构建的缺陷预测模型可以在 Internet Explorer 项目中取得好的预测效果,反之则不然.基于上述观察,他们又额外选择了来自微软公司和开源社区的 10 个大规模项目(共可构成有效组合 622 对)对跨项目缺陷预测性能进行了分析,他们发现,仅有 21 对可以取得满意的缺陷预测性能(即,查准率、查全率和精确度取值均超过 75%).He 等人[97]基于 5 种不同的分类方法发现,仅有 0.32%~4.67%的跨项目缺陷预测可以取得满意的预测性能(即:查准率超过 50%,且查全率超过 70%).Raham 等人[98]认为,不能借助传统的基于信息检索的度量指标(例如查准率、查全率和 *F*-measure)来评估跨项目缺陷预测的性能.在实际软件测试中,由于测试资源有限,研究人员无法对所有预测为 FP 的程序模块进行软件测试或代码审查,因此需要从成本收益角度(即,使用需要测试或代码审查的程序模块比例)进行评估.从这个角度出发,他们得到了不同的结论,即:跨项目缺陷预测的性能并不一定比同项目缺陷预测性能差,并且要显著优于随机预测.

迁移学习(transfer learning)[99]是运用已存在的知识对不同但相关领域的问题进行求解的一种机器学习方法,其目的是迁移已有的知识来解决目标领域仅有少量已标记数据甚至没有的问题.研究人员基于迁移学习,分别从实例选择、实例权重设置、特征映射和度量元选择等角度,对提高跨项目缺陷预测性能展开了研究.

一些研究人员根据目标项目的特征,通过从源项目中抽取典型实例来提高跨项目缺陷预测性能.例如:

- Turhan 等人[100]借助最近邻过滤(nearest neighbor filter)法,即,通过计算出目标项目中实例与源项目中实例的欧氏距离后,从源项目中选择合适的实例来缓解跨项目缺陷预测问题,但其预测性能仍显著低于同项目缺陷预测;
- Peters 等人[101]借助 Peters 过滤法,通过分析所有源项目的结构来为目标项目选出相关实例;
- He 等人[97]借助 16 个不同的分布特征指标(例如中位数、均值、方差、变异系数和偏度等)来搜集每

个度量元的分布特征并得到数据集的特征分布,最后,基于上述信息从源项目中选出合适的实例.

- Chen 等人[102]从移除源项目中的反实例(negative instances)角度出发,提出了 DTB(double transfer boosting)方法,该方法包括两个阶段:首先借助 DGM(data gravitation method)法,根据目标项目数据集特征来重塑源项目数据集的分布;其次,借助 TBM(transfer boosting method)法,通过利用目标项目中的小部分已标记实例来移除源项目数据集中的反实例;
- 杨叶等人[103]借助数据的相似性度量和特征子集的选择来移除源项目数据集中的无关数据和不稳定数据.

Ma 等人[104]从为源项目的实例权重设置入手,提出一种新的跨项目缺陷预测 TNB(transfer naïve Bayes)方法.该方法根据目标项目调整源项目中每个实例的条件概率,但该方法仅适用于朴素贝叶斯分类方法.Nam 等人基于特征映射方法[105],即,采用 TCA(transfer component analysis)方法[106]来进行跨项目缺陷预测.TCA 方法在保持原有数据属性时,通过最小化数据分布间的距离来为源项目和目标项目寻找潜在的特征空间(latent feature space).一旦确定潜在特征空间后,就可以将源项目和目标项目映射到该空间上,从而缓解不同项目间存在的数据分布差异性问题.在实证研究中他们发现,数据归一化方法(例如 min-max 法、*z*-score 方法等)对 TCA 方法的性能影响较大.因此,他们进一步提出 TCA+方法.该方法通过设定一系列规则,可以在分析源项目和目标项目的特征后,自动选择出合适的归一化方法.

一些研究人员尝试构建出适用于所有项目的通用缺陷预测模型来缓解跨项目缺陷预测问题.同时,对通用缺陷预测模型的研究,也有助于更好地解释度量元与程序模块缺陷间的相关性.Zhang 等人[107]选择了 32 种度量元,其中 21 种与软件代码相关,5 种与软件开发过程相关,剩余 6 种与项目的上下文相关(包括编程语言、是否使用缺陷跟踪系统、源代码总行数、项目总文件数、项目代码总提交数和参与项目的总人数).但不同项目的度量元取值分布变化幅度较大,因此他们提出了一种有效的数据预处理方法.具体来说:首先,根据项目上下文相关度量元取值,将所有项目进行分组;其次,将这些分组按照度量元取值分布的相似性进行聚类;最后,对于每个聚类,他们推导出一个转化函数,该函数可以使得转化后的度量元取值具有相同的尺度.在实证研究中,他们搜集了来自 SourceForge 和 Google Code 的 1 398 个开源项目,借助上述方法对各个项目的度量元取值进行预处理后,最终构建出通用缺陷预测模型,并在 5 个新项目中验证了该模型的有效性.马于涛等人[108]同样尝试选出少量具有代表性的度量元,希望可以构建出能达到一定性能标准的缺陷预测模型.

除了上述基于迁移学习的解决方案外,也有研究人员从其他角度对跨项目缺陷预测模型构造方法进行研究. Turhan 等人[109]从综合使用源项目数据和目标项目数据角度出发,通过实证研究发现:如果目标项目数据充足,则并不需要进一步考虑源项目数据;但是,如果目标项目仅有少数数据,则综合使用一些源项目的数据来构建缺陷预测模型是合理的.Canfora 等人[110]针对跨项目缺陷预测问题,为了在预测出更多的 FP 模块数和审查更少的的代码这两个不同的目标中取得更好的折中方案,他们借助多目标遗传算法来构建 Logistic 回归模型. Panichella 等人[111]发现,不同的机器学习方法预测出的 FP 程序模块并不相同.基于上述观察,他们提出一种混合方法 CODEP,该方法通过充分利用不同的且具有一定互补性的机器学习方法来提高跨项目缺陷预测性能.

5 基于代码修改的缺陷预测研究

这一节对一类特殊的软件缺陷预测问题(即,基于代码修改的缺陷预测)的已有研究工作进行总结.与上述传统的缺陷预测问题相比,虽然预测流程相似,但该问题仍存在如下特殊之处:首先,该问题的预测目标是,需要预测出提交的代码修改是否会产生缺陷;其次,该问题在软件历史仓库挖掘过程中,关注的模块粒度是代码修改,因此需要提出有效的方法对代码修改进行正确标记;最后,在度量元设计时,需要从分析代码修改内容、修改前后代码相关特征的变化以及开发过程特征来为代码修改设计相关度量元.接下来,我们将围绕上述不同点,对已有的研究工作进行总结.

软件在开发和维护过程中,为了移除软件内在缺陷、完善已有功能、重构已有代码或提高运行性能等,需要执行代码修改.但一些代码修改在完成修改任务后,可能会意外引入新的缺陷.因此,开发人员希望存在一个

缺陷预测模型,可以快速、准确地判断提交的代码修改是属于 buggy 代码修改(即,代码修改会产生缺陷)或 clean 代码修改(即,代码修改不会产生缺陷).若代码修改被预测为 buggy 代码修改,则开发人员在还熟悉代码修改内容的时候,可以通过设计更多的单元测试用例、进行代码审查或查看项目中的相似代码修改,来迅速定位并移除缺陷.本节将该问题称为基于代码修改的缺陷预测问题,Kamei 等人[112]又将这类软件质量保障活动称为 Just-in-time 质量保障.

Kim 等人[113]首次将该问题视为分类问题,并展开了深入研究.假设通过挖掘版本控制系统后,可以得到如图 4 所示的某个文件的代码修改历史.通过分析第 1 次修改到第(n–1)修改,构建缺陷预测数据集,随后,借助机器学习方法从数据集中学习出 buggy 代码修改和 clean 代码修改的修改模式,并完成对第 n 次代码修改的预测.

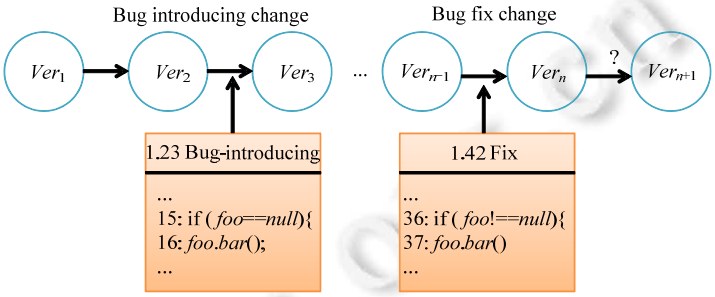

Fig.4 Modification history of a specific file

图 4 某个文件的修改历史

针对该问题的训练数据集的构造过程可总结如下:首先,通过分析版本控制系统,可以抽取出所有的代码修改;其次,通过从修改日志中搜索 Fixed 或 Bug 之类的关键词或者缺陷报告的引用(例如#42233),可以从所有代码修改中识别出修复缺陷的代码修改(bug fix changes),借助 Sliwerski 等人[114]提出的 SZZ 方法,从修复缺陷的代码修改处开始往回追溯,可以进一步识别出 buggy 代码修改(bug-introducing change)和 clean 代码修改(clean change),以图 4 为例,假设修改 1.42 修复了行 36 中的一个缺陷,通过往回追溯发现,该行曾经在修改 1.23 中被改动过(对应的是行 15),因此可以标记修改 1.23 为 buggy 代码修改;最后,他们通过分析源代码、代码修改和修改日志,抽取出大量与代码修改相关的特征(即度量元),这些特征主要分为 3 类.

(1) 分析与代码修改相关的元数据.

抽取的特征包括提交代码修改的开发人员、提交时间、累积修改次数、累积缺陷数、修改日志长度以及新增、删除和修改的代码行数等.

(2) 分析代码的复杂性.

Kim 等人共选择了 61 种传统复杂性度量元,包括 LOC、注释行数、环路复杂度和最大嵌套数等.因为每次代码修改均涉及到修改前程序和修改后程序,所以针对每个度量元,他们会分别对修改前后程序进行代码度量,并取两者间的差值作为度量元取值.

(3) 分析修改日志、源代码和文件的名称.

因为修改日志与电子邮件或新闻相似,日志中的每个单词都会传递一些信息,因此,Kim 等人基于文本挖掘领域中的 BOW(bag-of-word)方法来进行特征抽取.同样,他们对 BOW 方法进行扩展,使之可以将新增代码、删除代码和修改后代码中的操作符、数值、关键词和注释等特征抽取出来.除此之外,他们也将目录和文件的名称转化为特征,因为这些名称包含了一些模块信息和源代码的行为语义.

在实证研究中,Kim 等人通过分析 12 个开源项目,以支持向量机作为分类方法,发现可以平均获得 78%的准确率和 60%的查全率.

Kim 等人[113]提出的方法虽然可以取得较好的预测性能,但 Shivaji 等人[115]发现:上述方法因为考虑了文本挖掘方法,造成构造出的训练集的特征数太多(实证研究对象包含的特征数介于 6 127~41 942 个之间),所以存在

明显的维数灾难问题.他们考察了多种不同的特征子集选择方法,其中,过滤法主要基于 Gain Ratio,Chi-Squared,Significance 或 Relief-*F* 来计算特征与类之间的相关性;而包装法主要基于朴素贝叶斯和支持向量机.结果表明,大部分时候只需最多使用其中 10%的特征.除此之外,与已有工作相比,以朴素贝叶斯作为分类方法,特征选择方法可以使得 *F*-measure 值取得 21%的提高;而以支持向量机作为分类方法,特征选择方法可以使得 *F*-measure 值取得 9%的提高.

与 Kim 等人[113]的研究工作不同,Kamei 等人[112]从 5 个角度入手,抽取出与代码修改相关的度量元.这 5 个角度包括:

(1) 代码修改的分散度.该类度量元认为若代码修改涉及到多个程序模块,则代码修改将难以理解并容易引入缺陷;

(2) 代码修改的规模.该类度量元认为:若代码修改涉及的行数较多,则代码修改容易引入缺陷;

(3) 代码修改的目的.该类度量元认为,以修复缺陷为目的的代码修改更容易引入新的缺陷.其假设基于缺陷具有时间局部性,即,过去含有缺陷的程序模块在将来也可能含有缺陷;

(4) 代码修改的历史.该类度量元认为:通过分析代码修改的历史,可以为代码修改的分类提供有用的信息.例如,代码的修改次数和代码修改涉及到的开发人员数均与缺陷存在相关性;

(5) 开发人员的经验.该类度量元认为,经验丰富的开发人员引入缺陷的可能性较小.

在实证研究中,Kamei 等人通过分析 6 个开源项目和 5 个商业项目,以 Logistic 回归作为分类方法,发现可以平均获得 68%的准确率和 64%的查全率.除此之外,他们在方法有效性评估时,还进一步考虑了代码审查和软件测试的开销,结果发现:仅需要花费 20%的开销,就可以发现 35%的 buggy 代码修改.

Fukushima 等人[116]则进一步从跨项目缺陷预测角度对该问题展开了研究,通过分析 11 个开源软件项目,他们同样发现:跨项目缺陷预测性能要低于同项目缺陷预测性能;但若两个项目间度量元取值分布具有较高的相似度,则跨项目缺陷预测经常可以获得较好的性能.

刘超等人[117]则考虑了在长时间软件演化过程中,由于新增需求或人员重组等外在因素所带来的概念漂移问题.他们主要从修改的上下文、内容、时间和人员这 4 个角度来设计度量元,同时采用特征熵差值矩阵分析了软件演化过程中概念漂移问题的特点,并通过一种伴随概念回顾的动态窗口学习机制来确保长时间内缺陷预测性能保持稳定.

Tan 等人[118]在将基于代码修改的缺陷预测应用到企业的实际开发过程中,发现存在两个问题:首先,数据集存在明显的类不平衡问题;其次,代码修改之间存在时序关系.因此,已有的交叉验证方式并不完全适用于基于代码修改的缺陷预测,即,不能用后提交的代码修改来预测先提交的代码修改.他们应用了一种时间敏感的在线分类方法来遵循代码修改间存在的时序关系,并进一步借助重采样技术和可更新的分类方法来解决数据集中的类不平衡问题.

6 总结和展望

随着软件项目的内在规模和复杂度的不断提高,静态软件缺陷预测方法可以通过预测出潜在缺陷程序模块来优化测试资源的分配,并提高软件产品质量.因此,该问题具有丰富的理论价值和应用前景,并日益得到了学术界和工业界的广泛关注.虽然研究人员针对该问题取得了大量的研究成果,但我们认为,该研究领域还存在大量值得国内研究人员进一步关注的研究问题.

(1) 随着 Promise 库的建立和美国国家航空航天局项目(简称 NASA 数据集)的共享,极大地推动了软件缺陷预测的研究进展.据我们统计,有 51 篇论文选用了公开的评测对象,即,来源于 Promise 库提供的数据集或 NASA 数据集.但同时我们也发现,有 60 篇论文选择了非公开的评测对象,这些评测对象或者来自于商业企业(例如微软、谷歌或思科等)的内部项目,或者来自于研究人员自己对一些开源项目的挖掘,因此,这些研究工作存在实证研究难以重现或研究结论不具有一般性等问题.随着开源软件的迅猛发展,研究人员可以从开源软件所处的一些网站(例如 SourceForge,GitHub 或 Google Code)上获取更多新项目的历史开发数据或一些项目的最新

开发数据.通过搜集和挖掘这些数据并共享到 Promise 库,可以进一步提高缺陷预测模型的性能,同时也可以对一些已有实证研究的结论进行重新验证.

(2) 已有研究对缺陷预测模型的应用关注得较少.我们认为,缺陷预测模型可以在回归测试和软件缺陷定位等领域取得成功的应用.例如在回归测试中,测试用例优先级排序问题[119]通过设计排序准则,然后将测试用例按重要程度从高到低进行排序并依次执行,旨在最大化测试用例集的早期缺陷检测速率.若将缺陷预测模型引入到该问题中,则提供了一个新的研究角度,即:对于预测出的那些含有缺陷可能性高的程序模块,在考虑测试用例的执行次序时,应该让更多覆盖了这些程序模块的测试用例尽早去执行.同样,在基于程序频谱的缺陷定位问题中[120],已有研究一般通过搜集测试用例的程序频谱和执行结果,基于特定模型以定位缺陷语句在被测程序内的可能位置.若将缺陷预测模型引入到该问题,则在程序频谱构造时,可以预先排除一些含有缺陷可能性低的程序模块;而对于含有缺陷可能性高的程序模块,则对该模块内的语句可以赋予更高的怀疑率取值.

(3) 虽然大部分研究人员都愿意将他们研究过程中搜集的数据集进行共享,但他们担心数据集在共享后,其内部存在的隐私会被攻击者获取.因此,数据集的隐私问题是造成很多实证研究难以重现并阻碍其他研究人员开展后续研究工作的一个重要因素.Peters 等人[121,122]对该问题进行了初步探索,他们主要借助 MORPH 隐私方法(即,基于搅动的数据转换方法)来合成数据集.具体来说,通过对实例中的特征取值进行随机搅动,使得该实例可以在 n 维空间中移动到一个新的位置,但同时可以确保该移动不会穿越不同类间的边界.因此,该方法可以确保数据集中的某些敏感度量元的取值不会被攻击者获取,同时又能保证合成的数据集仍能用于跨项目缺陷预测研究.

(4) 与已有的静态缺陷检测方法相融合.静态缺陷检测方法从最简单的借助预定义的代码模式来检测出潜在的缺陷语句,到复杂的基于程序静态分析,借助预定义的代码语义抽象来检测出潜在的缺陷语句.一些代表性工具包括 PMD,FindBugs,ESC/Java 和 CodeSonar 等.Rahman 等人[123]在实证研究中将静态缺陷检测方法与本文综述关注的静态软件缺陷预测方法进行了系统比较,发现这两类不同类型的方法间存在一定的互补性,即,这两类方法经常会找到不同的软件缺陷.因此,如何有效地融合这两类不同方法以进一步提高软件缺陷预测精度,具有一定的研究意义.

(5) 支持更细粒度的缺陷预测,可以大幅度缩小开发人员需要人工审查的代码范围.已有研究大部分将程序模块粒度设置为粗粒度(即,包或者文件级别),很少有研究人员将程序模块粒度设置为细粒度,例如方法级别.Hata 等人[124]和 Giger 等人[125]在这一方面做了一些初步尝试,但这类方法主要存在分析开销大等问题.除此之外,基于代码修改的软件缺陷预测也是针对该问题的一种可行的解决方案,但同样,这种方案也存在计算开销大的问题,因此造成上述研究成果难以应用到实际的软件开发过程中.

(6) 跨项目缺陷预测问题仍是当前软件缺陷预测研究中的一个难点,其中有两个问题值得去关注:首先,给定一个目标项目,研究人员需要提出一种有效方法可以预先评估该项目进行跨项目缺陷预测的可行性;其次,虽然现在从 Promise 库中或者挖掘开源软件所在网站可以获取大量的候选源项目,但这些源项目采用的度量元与目标项目采用的度量元很难保持一致.因此,如何针对这种异构数据集设计新颖、高效的迁移学习方法,是一个重要的研究问题.

(7) 目前,产业界和学术界都比较关注的问题是如何将缺陷预测模型与企业的已有开发流程加以融合,以及融合之后如何有效提高开发人员的开发和调试效率.目前,已有一些研究人员在一些商业企业中进行了尝试,例如微软[126]、谷歌[67]和思科[118]等.但距离成功应用仍存在很多问题,例如:不存在成熟的工具支持、已有缺陷预测模型的预测性能不能令人满意、模型的预测结果的可读性不高等.因此,如何与企业已有的软件测试和调试流程相结合,仍是未来需要重点关注的研究问题.

若能对上述研究挑战提出有效的解决方案,将进一步提高该领域的应用价值,最终达到大幅度提高企业内的测试资源分配效率、缓解项目测试资源有限和软件产品高质量需求间的矛盾.

References:

[1] Wang Q, Wu SJ, Li MS. Software defect prediction. Ruan Jian Xue Bao/Journal of Software, 2008,19(7):1565−1580 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/1565.htm

[2] Hall T, Beecham S, Bowes D, Gray D, Counsell S. A systematic literature review on fault prediction performance in software engineering. IEEE Trans. on Software Engineering, 2012,38(6):1276−1304. [doi: 10.1109/TSE.2011.103]

[3] Yu SS, Zhou SG, Guan JH. Software engineering data mining: A survey. Journal of Frontiers of Computer Science and Technology, 2012,6(1):1−31 (in Chinese with English abstract). [doi: 10.1299/jcst.6.1]

[4] Radjenovic D, Hericko M, Torkar R, Zivkovic A. Software fault prediction metrics: A systematic literature review. Information and Software Technology, 2013,55(8):1397−1418. [doi: 10.1016/j.infsof.2013.02.009]

[5] Akiyama F. An example of software system debugging. In: Proc. of the Int'l Federation of Information Proc. Societies Congress. New York: Springer Science and Business Media, 1971. 353−359.

[6] Halstead MH. Elements of Software Science (Operating and Programming Systems Series). New York: Elsevier Science Inc., 1977.

[7] McCabe TJ. A complexity measure. IEEE Trans. on Software Engineering, 1976,2(4):308−320. [doi: 10.1109/TSE.1976.233837]

[8] Chidamber SR, Kemerer CF. A metrics suite for object oriented design. IEEE Trans. on Software Engineering, 1994,20(6): 476−493. [doi: 10.1109/32.295895]

[9] Basili VR, Briand LC, Melo WL. A validation of object-oriented design metrics as quality indicators. IEEE Trans. on Software Engineering, 1996,22(10):751−761. [doi: 10.1109/32.544352]

[10] Subramanyam R, Krishnan MS. Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects. IEEE Trans. on Software Engineering, 2003,29(4):297−310. [doi: 10.1109/TSE.2003.1191795]

[11] Zhou YM, Xu BW, Leung H. On the ability of complexity metrics to predict fault-prone classes in object-oriented systems. Journal of Systems and Software, 2010,83(4):660−674. [doi: 10.1016/j.jss.2009.11.704]

[12] Zhou YM, Leung H, Xu BW. Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness. IEEE Trans. on Software Engineering, 2009,35(5):607−623. [doi: 10.1109/TSE.2009.32]

[13] Zhou YM, Xu BW, Leung H, Chen L. An in-depth study of the potentially confounding effect of class size in fault prediction. ACM Trans. on Software Engineering and Methodology, 2014,23(1):10:1−10:51. [doi: 10.1145/2556777]

[14] Zhao YY, Yang YB, Lu HM, Zhou YM, Song QB, Xu BW. An empirical analysis of package-modularization metrics: Implications for software fault-proneness. Information and Software Technology, 2015,57:186−203. [doi: 10.1016/j.infsof.2014.09.006]

[15] Yang YB, Zhou YM, Lu HM, Chen L, Chen ZY, Xu BW, Leung H, Zhang ZY. Are slice-based cohesion metrics actually useful in effort-aware post-release fault-proneness prediction? an empirical study. IEEE Trans. on Software Engineering, 2015,41(4): 331−357. [doi: 10.1109/TSE.2014.2370048]

[16] Sarkar S, Kak AC, Rama GM. Metrics for measuring the quality of modularization of large-scale object-oriented software. IEEE Trans. on Software Engineering, 2008,34(5):700−720. [doi: 10.1109/TSE.2008.43]

[17] Meyers TM, Binkley D. An empirical study of slice-based cohesion and coupling metrics. ACM Trans. on Software Engineering and Methodology, 2007,17(1):2:1−27. [doi: 10.1145/1314493.1314495]

[18] Zimmermann T, Nagappan N. Predicting subsystem failures using dependency graph complexities. In: Proc. of the Int'l Symp. on Software Reliability. 2007. 227−236. [doi: 10.1109/ISSRE.2007.19]

[19] Zimmermann T, Nagappan N. Predicting defects using network analysis on dependency graphs. In: Proc. of the Int'l Conf. on Software Engineering. 2008. 531−540. [doi: 10.1145/1368088.1368161]

[20] Shin Y, Bell RM, Ostrand TJ, Weyuker EJ. On the use of calling structure information to improve fault prediction. Empirical Software Engineering, 2012,17(4−5):390−423. [doi: 10.1007/s10664-011-9165-9]

[21] Binkley D, Field H, Lawrie D, Pighin M. Increasing diversity: Natural language measures for software fault prediction. Journal of Systems and Software, 2009,82(11):1793−1803. [doi: 10.1016/j.jss.2009.06.036]

[22] Lawrie DJ, Field H, Binkley D. Leveraged quality assessment using information retrieval techniques. In: Proc. of the Int'l Conf. on Program Comprehension. 2006. 149−158. [doi: 10.1109/ICPC.2006.34]

[23] Nagappan N, Ball T. Use of relative code churn measures to predict system defect density. In: Proc. of the Int'l Conf. on Software Engineering. 2005. 284−292. [doi: 10.1145/1062455.1062514]

[24] Moser R, Pedrycz W, Succi G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In: Proc. of the Int'l Conf. on Software Engineering. 2008. 181−190. [doi: 10.1145/1368088.1368114]

[25] Hassan AE. Predicting faults using the complexity of code changes. In: Proc. of the Int'l Conf. on Software Engineering. 2009. 78−88. [doi: 10.1109/ICSE.2009.5070510]

[26] D'Ambros M, Lanza M, Robbes R. An extensive comparison of bug prediction approaches. In: Proc. of the Working Conf. on Mining Software Repositories. 2010. 31−41. [doi: 10.1109/MSR.2010.5463279]

[27] Graves TL, Karr AF, Marron JS, Siy H. Predicting fault incidence using software change history. IEEE Trans. on Software Engineering, 2000,26(7):653−661. [doi: 10.1109/32.859533]

[28] Weyuker EJ, Ostrand TJ, Bell RM. Using developer information as a factor for fault prediction. In: Proc. of the Int'l Workshop on Predictor Models in Software Engineering. 2007. 1−7. [doi: 10.1109/PROMISE.2007.14]

[29] Weyuker EJ, Ostrand TJ, Bell RM. Do too many cooks spoil the broth? Using the number of developers to enhance defect prediction models. Empirical Software Engineering, 2008,13(5):539−559. [doi: 10.1007/s10664-008-9082-8]

[30] Pinzger M, Nagappan N, Murphy B. Can developer-module networks predict failures? In: Proc. of the Int'l Symp. on Foundations of Software Engineering. 2008. 2−12. [doi: 10.1145/1453101.1453105]

[31] Meneely A, Williams L, Snipes W, Osborne J. Predicting failures with developer networks and social network analysis. In: Proc. of the Int'l Symp. on Foundations of Software Engineering. 2008. 13−23. [doi: 10.1145/1453101.1453106]

[32] Jiang T, Tan L, Kim S. Personalized defect prediction. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2013. 279−289. [doi: 10.1109/ASE.2013.6693087]

[33] Bird C, Nagappan N, Murphy B, Gall H, Devanbu P. Don't touch my code! Examining the effects of ownership on software quality. In: Proc. of Joint Meeting of the European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. 2011. 4−14. http://dl.acm.org/citation.cfm?id=2025119 [doi: 10.1145/2025113.2025119]

[34] Rahman F, Devanbu P. Ownership, experience and defects: A fine-grained study of authorship. In: Proc. of the Int'l Conf. on Software Engineering. 2011. 491−500. [doi: 10.1145/1985793.1985860]

[35] Posnett D, D'Souza R, Devanbu P, Filkov V. Dual ecological measures of focus in software development. In: Proc. of the Int'l Conf. on Software Engineering. 2013. 452−461. [doi: 10.1109/ICSE.2013.6606591]

[36] Lee T, Nam J, Han D, Kim S, In HP. Micro interaction metrics for defect prediction. In: Proc. of the Joint Meeting of the European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. 2011. 311−321. [doi: 10.1145/2025113.2025156]

[37] Bird C, Nagappan N, Gall H, Murphy B, Devanbu P. Putting it all together: Using socio-technical networks to predict failures. In: Proc. of the Int'l Symp. on Software Reliability Engineering. 2009. 109−119. [doi: 10.1109/ISSRE.2009.17]

[38] Hu W, Wong K. Using citation influence to predict software defects. In: Proc. of the Working Conf. on Mining Software Repositories. 2013. 419−428. [doi: 10.1109/MSR.2013.6624058]

[39] D'Ambros M, Lanza M, Robbes R. On the relationship between change coupling and software defects. In: Proc. of the Working Conf. on Reverse Engineering. 2009. 135−144. [doi: 10.1109/WCRE.2009.19]

[40] Herzig K, Just S, Rau A, Zeller A. Predicting defects using change genealogies. In: Proc. of the Int'l Symp. on Software Reliability Engineering. 2013. 118−127. [doi: 10.1109/ISSRE.2013.6698911]

[41] Conway ME. How do committees invent. Datamation, 1968,14(4):28−31.

[42] Nagappan N, Murphy B, Basili VR. The influence of organizational structure on software quality: An empirical case study. In: Proc. of the Int'l Conf. on Software Engineering. 2008. 521−530. [doi: 10.1145/1368088.1368160]

[43] Mockus A. Organizational volatility and its effects on software defects. In: Proc. of the Int'l Symp. on Foundations of Software Engineering. 2010. 117−126. [doi: 10.1145/1882291.1882311]

[44] Bird C, Nagappan N, Devanbu P, Gall H, Murphy B. Does distributed development affect software quality? an empirical case study of Windows Vista. In: Proc. of Int'l Conf. on Software Engineering. 2009. 518−528. [doi: 10.1109/ICSE.2009.5070550]

[45] Bacchelli A, D'Ambros M, Lanza M. Are popular classes more defect prone. In: Proc. of the Int'l Conf. on Fundamental Approaches to Software Engineering. 2010. 59−73. [doi: 10.1007/978-3-642-12029-9_5]

[46] Taba SES, Khomh F, Zou Y, Hassan AE, Nagappan M. Predicting bugs using antipatterns. In: Proc. of the Int'l Conf. on Software Maintenance. 2013. 270−279. [doi: 10.1109/ICSM.2013.38]

[47] Herzig K. Using pre-release test failures to build early post-release defect prediction models. In: Proc. of the Int'l Symp. on Software Reliability Engineering. 2014. 300−311. [doi: 10.1109/ISSRE.2014.21]

[48] Wang JJ, Li J, Wang Q, Yand D, Zhang H, Li MS. Can requirements dependency network be used as early indicator of software integration bugs. In: Proc. of the Int'l Conf. on Requirement Engineering. 2013. 185−194. [doi: 10.1109/RE.2013.6636718]

[49] Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors. IEEE Trans. on Software Engineering, 2007,3(1):2−13. [doi: 10.1109/TSE.2007.256941]

[50] Zhang HY. An investigation of the relationships between lines of code and defects. In: Proc. of the Int'l Conf. on Software Maintenance. 2009. 274−283. [doi: 10.1109/ICSM.2009.5306304]

[51] Arisholm E, Briand LC, Johannessen EB. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. Journal of Systems and Software, 2010,83(1):2−17. [doi: 10.1016/j.jss.2009.06.055]

[52] Rahman F, Devanbu P. How, and why, process metrics are better. In: Proc. of the Int'l Conf. on Software Engineering. 2013. 432−441. [doi: 10.1109/ICSE.2013.6606589]

[53] Madeyski L, Jureczko M. Which process metrics can significantly improve defect prediction models? An empirical study. Software Quality Journal, 2015,23(3):393−422. [doi: 10.1007/s11219-014-9241-7]

[54] D'Ambros M, Lanza M, Robbes R. Evaluating defect prediction approaches: A benchmark and an extensive comparison. Empirical Software Engineering, 2012,17(4-5):531−577. [doi: 10.1007/s10664-011-9173-9]

[55] Elish KO, Elish MO. Predicting defect-prone software modules using support vector machines. Journal of Systems and Software, 2008,81(5):649−660. [doi: 10.1016/j.jss.2007.07.040]

[56] Lessmann S, Baesens B, Mues C, Pietsch S. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. IEEE Trans. on Software Engineering, 2008,34(4):485−496. [doi: 10.1109/TSE.2008.35]

[57] Ghotra B, McIntosh S, Hassan AE. Revisiting the impact of classification techniques on the performance of defect prediction models. In: Proc. of the Int'l Conf. on Software Engineering. 2015. 789−800. [doi: 10.1109/ICSE.2015.91]

[58] Shepperd M, Bowes D, Hall T. Research Bias: The use of machine learning in software defect prediction. IEEE Trans. on Software Engineering, 2014,40(6):603−616. [doi: 10.1109/TSE.2014.2322358]

[59] Shepperd M, Song QB, Sun ZB, Mair C. Data quality: Some comments on the NASA software defect datasets. IEEE Trans. on Software Engineering, 2013,39(9):1208−1215. [doi: 10.1109/TSE.2013.11]

[60] Catal C, Diri B. Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Information Science, 2009,179(8):1040−1058. [doi: 10.1016/j.ins.2008.12.001]

[61] Song QB, Jia ZH, Shepperd M, Ying S, Liu J. A general software defect-proneness prediction framework. IEEE Trans. on Software Engineering, 2011,37(3):356−370. [doi: 10.1109/TSE.2010.90]

[62] Li M, Zhang HY, Wu RX, Zhou ZH. Sample-Based software defect prediction with active and semi-supervised learning. Automated Software Engineering, 2012,19(2):201−230. [doi: 10.1007/s10515-011-0092-1]

[63] Lu HH, Kocaguneli E, Cuki B. Defect prediction between software versions with active learning and dimensionality reduction. In: Proc. of the Int'l Symp. on Software Reliability Engineering. 2014. 312−322. [doi: 10.1109/ISSRE.2014.35]

[64] Hassan AE, Holt RC. The top ten list: Dynamic fault prediction. In: Proc. of the Int'l Conf. on Software Maintenance. 2005. 263−272. [doi: 10.1109/ICSM.2005.91]

[65] Kim S, Zimmermann T, Whitehead J, Zeller A. Predicting faults from cached history. In: Proc. of the Int'l Conf. on Software Engineering. 2007. 489−498. [doi: 10.1109/ICSE.2007.66]

[66] Rahman F, Posnett D, Hindle A, Barr E, Devanbu P. BugCache for inspections: Hit or miss. In: Proc. of the Joint Meeting of the European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. 2011. 322−331. [doi: 10.1145/2025113.2025157]

[67] Lewis C, Lin ZP, Sadowski C, Zhu XY, Ou R, Whitehead J. Does bug prediction support human developers? Findings from a Google case study. In: Proc. of the Int'l Conf. on Software Engineering. 2013. 372−381. [doi: 10.1109/ICSE.2013.6606583]

[68] Bird C, Bachmann A, Aune E, Duffy J, Bernstein A, Filkov V, Devanbu P. Fair and balanced? Bias in bug-fix datasets. In: Proc. of the the Joint Meeting of the European Software Engineering Conf. and the Symp. on The Foundations of Software Engineering. 2009. 121−130. [doi: 10.1145/1595696.1595716]

[69] Bachmann A, Bird C, Rahman F, Devanbu P, Bernstein A. The missing links: bugs and bug-fix commits. In: Proc. of the Int'l Symp. on Foundations of Software Engineering. 2010. 97−106. [doi: 10.1145/1882291.1882308]

[70] Nguyen TH, Adams B, Hassan AE. A case study of bias in bug-fix datasets. In: Proc. of the Working Conf. on Reverse Engineering. 2010. 259−268. [doi: 10.1109/WCRE.2010.37]

[71] Wu RX, Zhang HY, Kim S, Cheung SC. Relink: Recovering links between bugs and changes. In: Proc. of the European Conf. on Foundations of Software Engineering. 2011. 15−25. [doi: 10.1145/2025113.2025120]

[72] Nguyen AT, Nguyen TT, Nguyen HA, Nguyen TN. Multi-Layered approach for recovering links between bug reports and fixes. In: Proc. of the Int'l Symp. on the Foundations of Software Engineering. 2012. 63:1−63:11. [doi: 10.1145/2393596.2393671]

[73] Rahman F, Posnett D, Herraiz I, Devanbu P. Sample size vs. bias in defect prediction. In: Proc. of the Joint Meeting of the European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. 2013. 147−157. [doi: 10.1145/2491411.2491418]

[74] Herzig K, Just S, Zeller A. It's not a bug, it's a feature: How misclassification impacts bug prediction. In: Proc. of the Int'l Conf. on Software Engineering. 2013. 392−401. [doi: 10.1109/ICSE.2013.6606585]

[75] Kim S, Zhang HY, Wu RX, Gong L. Dealing with noise in defect prediction. In: Proc. of the Int'l Conf. on Software Engineering. 2011. 481−490. [doi: 10.1145/1985793.1985859]

[76] Tantithamthavorn C, McIntosh S, Hassan AE, Ihar A, Matsumoto K. The impact of mislabeling on the performance and interpretation of defect prediction models. In: Proc. of the Int'l Conf. on Software Engineering. 2015. 812−823.

[77] Gao KH, Khoshgoftaar TM, Wang HJ, Seliya N. Choosing software metrics for defect prediction: An investigation on feature selection techniques. Software-Practice and Experience, 2011,41(5):579−606. [doi: 10.1002/spe.1043]

[78] Liu SL, Chen X, Liu WS, Chen JQ, Gu Q, Chen DX. FECAR: A feature selection framework for software defect prediction. In: Proc. of the Annual Computer Software and Applications Conf. 2014. 426−435. [doi: 10.1109/COMPSAC.2014.66]

[79] Wang Q, Zhu J, Yu B. Feature selection and clustering in software quality prediction. In: Proc. of the Int'l Conf. on Evaluation and Assessment in Software Engineering. 2007. 21−32. http://www.bcs.org/upload/pdf/ewic_ea07_paper3.pdf

[80] Turhan B, Bener A. A multivariate analysis of static code attributes for defect prediction. In: Proc. of the Int'l Conf. on Quality Software. 2007. 231−237. [doi: 10.1109/QSIC.2007.4385500]

[81] He HB, Garcia EA. Learning from imbalanced data. IEEE Trans. on Knowledge and Data Engineering, 2009,21(9):1263−1284. [doi: 10.1109/TKDE.2008.239]

[82] Menzies T, Turhan B, Bener A, Gay G, Cukic B, Jiang Y. Implications of ceiling effects in defect predictors. In: Proc. of the Int'l Workshop on Predictor Models in Software Engineering. 2008. 47−54. [doi: 10.1145/1370788.1370801]

[83] Shatnawi R. Improving software fault-prediction for imbalanced data. In: Proc. of the Int'l Conf. on Innovations in Information Technology. 2012. 54−59. [doi: 10.1109/INNOVATIONS.2012.6207774]

[84] Pelayo L, Dick S. Applying novel resampling strategies to software defect prediction. In: Proc. of the Annual Meeting of the North American Fuzzy Information Processing Society. 2007. 69−72. [doi: 10.1109/NAFIPS.2007.383813]

[85] Wang S, Yao X. Using class imbalance learning for software defect prediction. IEEE Trans. on Reliability, 2013,62(2):434−443. [doi: 10.1109/TR.2013.2259203]

[86] Zheng J. Cost-Sensitive boosting neural networks for software defect prediction. Expert Systems with Application, 2010,37(6): 4537−4543. [doi: 10.1016/j.eswa.2009.12.056]

[87] Jiang Y, Li M, Zhou ZH. Software defect detection with Rocus. Journal of Computer Science and Technology, 2011,26(2): 328−342. [doi: 10.1007/s11390-011-9439-0]

[88] Jing XY, Ying S, Zhang ZW, Wu SS, Liu J. Dictionary learning based software defect prediction. In: Proc. of the Int'l Conf. on Software Engineering. 2014. 414−423. [doi: 10.1145/2568225.2568320]

[89] Rodriguez D, Herraiz I, Harrison R, Dolado J, Riquelme JC. Preliminary comparison of techniques for dealing with imbalance in software defect prediction. In: Proc. of the Int'l Conf. on Evaluation and Assessment in Software Engineering. 2014. 43:1−43:10. [doi: 10.1145/2601248.2601294]

[90] Seiffert C, Khoshgoftaar TM, Van Hulse J, Folleco A. An empirical study of the classification performance of learners on imbalanced and noisy software quality data. Information Sciences, 2014,259:571−595. [doi: 10.1016/j.ins.2010.12.016]

[91] Khoshgoftaar TM, Gao KH. Feature selection with imbalanced data for software defect prediction. In: Proc. of the Int'l Conf. on Machine Learning and Applications. 2009. 235−240. [doi: 10.1109/ICMLA.2009.18]

[92] Liu MX, Miao LS, Zhang DQ. Two-Stage cost-sensitive learning for software defect prediction. IEEE Trans. on Reliability, 2014, 63(2):676−686. http://rs.ieee.org/publications.html

[93] Chen JQ, Liu SL, Liu WS, Chen X, Gu Q, Chen DX. A two-stage data preprocessing approach for software fault prediction. In: Proc. of the Int'l Conf. on Software Security and Reliability. 2014. 20−29. [doi: 10.1109/SERE.2014.15]

[94] Laradji IH, Alshayeb M, Ghouti L. Software defect prediction using ensemble learning on selected features. Information and Software Technology, 2015,58:388−402. [doi: 10.1016/j.infsof.2014.07.005]

[95] Briand LC, Melo WL, Wust J. Assessing the applicability of fault-proness models across object-oriented software projects. IEEE Trans. on Software Engineering, 2002,28(7):706−720. [doi: 10.1109/TSE.2002.1019484]

[96] Zimmermann T, Nagappan N, Gall H, Giger E, Murphy B. Cross-Project defect prediction: A large scale experiment on data vs. domain vs. process. In: Proc. of the Joint Meeting of the European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. 2009. 91−100. [doi: 10.1145/1595696.1595713]

[97] He ZM, Shu FD, Yang Y, Li MS, Wang Q. An investigation on the feasibility of cross-project defect prediction. Automated Software Engineering, 2012,19(2):167−199. [doi: 10.1007/s10515-011-0090-3]

[98] Rahman F, Posnett D, Devanbu P. Recalling the "imprecision" of cross-project defect prediction. In: Proc. of the Int'l Symp. on the Foundations of Software Engineering. 2012. 61:1−61:11. [doi: 10.1145/2393596.2393669]

[99] Pan SJ, Yang Q. A survey on transfer learning. IEEE Trans. on Knowledge and Data Engineering, 2010,22(10):1345−1359. [doi: 10.1109/TKDE.2009.191]

[100] Turhan B, Menzies T, Bener AB, Stefano JD. On the relative value of cross-company and within-company data for defect prediction. Empirical Software Engineering, 2009,14(5):540−578. [doi: 10.1007/s10664-008-9103-7]

[101] Peters F, Menzies T, Marcus A. Better cross company defect prediction. In: Proc. of the Working Conf. on Mining Software Repositories. 2013. 409−418. [doi: 10.1109/MSR.2013.6624057]

[102] Chen L, Fang B, Shang ZW, Tang YY. Negative samples reduction in cross-company software defects prediction. Information and Software Technology, 2015,62:67−77. [doi: 10.1016/j.infsof.2015.01.014]

[103] He ZM, Peters F, Menzies T, Yang Y. Learning from open-source projects: An empirical study on defect prediction. In: Proc. of the Int'l Symp. on Empirical Software Engineering and Measurement. 2013. 45−54. [doi: 10.1109/ESEM.2013.20]

[104] Ma Y, Luo GC, Zeng X, Chen AG. Transfer learning for cross-company software defect prediction. Information and Software Technology, 2012,54(3):248−256. [doi: 10.1016/j.infsof.2011.09.007]

[105] Nam J, Pan SJ, Kim S. Transfer defect learning. In: Proc. of the Int'l Conf. on Software Engineering. 2013. 382−391. [doi: 10.1109/ICSE.2013.6606584]

[106] Pan SJ, Tsang IW, Kwok JT, Yang Q. Domain adaptation via transfer component analysis. IEEE Trans. on Neural Networks, 2011, 22(2):199−210. [doi: 10.1109/TNN.2010.2091281]

[107] Zhang F, Mockus A, Keivanloo I, Zou Y. Towards building a universal defect prediction model. In: Proc. of the Working Conf. on Mining Software Repositories. 2014. 182−191. [doi: 10.1145/2597073.2597078]

[108] He P, Li B, Liu X, Chen J, Ma YT. An empirical study on software defect prediction with a simplified metric set. Information and Software Technology, 2015,59:170−190. [doi: 10.1016/j.infsof.2014.11.006]

[109] Turhan B, Msrl AT, Bener A. Empirical evaluation of the effects of mixed project data on learning defect predictors. Information and Software Technology, 2013,55(6):1101−1118. [doi: 10.1016/j.infsof.2012.10.003]

[110] Canfora G, Lucia AD, Penta MD, Oliveto R, Panichella A, Panichella S. Multi-Objective cross-project defect prediction. In: Proc. of the Int'l Conf. on Software Testing, Verification and Validation. 2013. 252−261. [doi: 10.1109/ICST.2013.38]

[111] Panichella A, Oliveto R, Lucia AD. Cross-Project defect prediction models: L'Union fait la force. In: Proc. of the Software Maintenance, Reengineering and Reverse Engineering. 2014. 164−173. [doi: 10.1109/CSMR-WCRE.2014.6747166]

[112] Kamei Y, Shihab E, Adams B, Hassan AE, Mockus A, Sinha A, Ubayashi N. A large-scale empirical study of just-in-time quality assurance. IEEE Trans. on Software Engineering, 2013,39(6):757−773. [doi: 10.1109/TSE.2012.70]

[113] Kim S, Whitehead Jr EJ, Zhang Y. Classifying software changes: Clean or buggy. IEEE Trans. on Software Engineering, 2008, 34(2):181−196. [doi: 10.1109/TSE.2007.70773]

[114] Sliwerski J, Zimmermann T, Zeller A. When do changes induce fixes. In: Proc. of the Int'l Workshop on Mining Software Repositories. 2005. 1−5. [doi: 10.1145/1083142.1083147]

[115] Shivaji S, Whitehead Jr EJ, Akella R, Kim S. Reducing features to improve code change-based bug prediction. IEEE Trans. on Software Engineering, 2013,39(4):552−569. [doi: 10.1109/TSE.2012.43]

[116] Fukushima T, Kamei Y, McIntosh S, Yamashita K, Ubayashi. N. An empirical study of just-in-time defect prediction using cross-project models. In: Proc. of the Working Conf. on Mining Software Repositories. 2014. 172−181. [doi: 10.1145/2597073.2597075]

[117] Yuan Z, Yu LL, Liu C. Bug prediction method for fine-grained source code changes. Ruan Jian Xue Bao/Journal of Software, 2014, 25(11):2499−2517 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4559.htm [doi: 10.13328/j.cnki.jos.004559]

[118] Tan M, Tan L, Dara S, Mayeux C. Online defect prediction for imbalanced data. In: Proc. of the Int'l Conf. on Software Engineering. 2015. 99−108. [doi: 10.1109/ICSE.2015.139]

[119] Chen X, Chen JH, Ju XL, Gu Q. Survey of test case prioritization techniques for regression testing. Ruan Jian Xue Bao/Journal of Software, 2013,24(8):1695−1712 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4420.htm [doi: 10.3724/SP.J.1001.2013.04420]

[120] Chen X, Ju XL, Wen WZ, Gu Q. Review of dynamic fault localization approaches based on program spectrum. Ruan Jian Xue Bao/Journal of Software, 2015,26(2):390−412 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4708.htm [doi: 10.13328/j.cnki.jos.004708]

[121] Peters F, Menzies T. Privacy and utility for defect prediction: Experiments with morph. In: Proc. of the Int'l Conf. on Software Engineering. 2012. 189−199. [doi: 10.1109/ICSE.2012.6227194]

[122] Peters F, Menzies T, Gong L, Zhang HY. Balancing privacy and utility in cross-company defect prediction. IEEE Trans. on Software Engineering, 2013,39(8):1054−1068. [doi: 10.1109/TSE.2013.6]

[123] Rahman F, Khatri S, Barr ET, Devanbu P. Comparing static bug finders and statistical prediction. In: Proc. of the Int'l Conf. on Software Engineering. 2014. 424−434. [doi: 10.1145/2568225.2568269]

[124] Hata H, Mizuno O, Kikuno T. Bug prediction based on fine-grained module histories. In: Proc. of the Int'l Conf. on Software Engineering. 2012. 200−210. [doi: 10.1109/ICSE.2012.6227193]

[125] Giger E, D'Ambros M, Pinzger M, Harald CG. Method-Level bug prediction. In: Proc. of the Int'l Symp. on Empirical Software Engineering and Measurement. 2012. 171−180. [doi: 10.1145/2372251.2372285]

[126] Czerwonka J, Das R, Nagappan N, Tarvo A, Teterev A. CRANE: Failure prediction, change analysis and test prioritization in practice—Experiences from Windows. In: Proc. of the Int'l Conf. on Software Testing, Verification and Validation. 2011. 357−366. [doi: 10.1109/ICST.2011.24]

附中文参考文献:

[1] 王青,伍书剑,李明树.软件缺陷预测技术.软件学报,2008,19(7):1565−1580. http://www.jos.org.cn/1000-9825/19/1565.htm

[3] 郁抒思,周水庚,关佶红.软件工程数据挖掘研究进展.计算机科学与探索,2012,6(1):1−31. [doi: 10.1299/jcst.6.1]

[117] 原子,于莉莉,刘超.面向细粒度源代码变更的缺陷预测方法.软件学报,2014,25(11):2499−2517. http://www.jos.org.cn/1000-9825/4559.htm [doi: 10.13328/j.cnki.jos.004559]

[119] 陈翔,陈继红,鞠小林,顾庆.回归测试中的测试用例优先排序技术述评.软件学报,2013,24(8):1695−1712. http://www.jos.org.cn/1000-9825/4420.htm [doi: 10.3724/SP.J.1001.2013.04420]

[120] 陈翔,鞠小林,文万志,顾庆.基于程序频谱的动态缺陷定位方法研究.软件学报,2015,26(2):390−412. http://www.jos.org.cn/1000-9825/4708.htm [doi: 10.13328/j.cnki.jos.004708]

陈翔(1980－),男,江苏南通人,博士,副教授,CCF 会员,主要研究领域为软件缺陷预测,软件缺陷定位,回归测试和组合测试.

顾庆(1972－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件质量保障,分布式计算.

刘望舒(1987－),男,博士生,主要研究领域为软件缺陷预测.

刘树龙(1990－),男,工程师,主要研究领域为软件缺陷预测.

倪超(1990－),男,硕士生,主要研究领域为软件缺陷预测.