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.

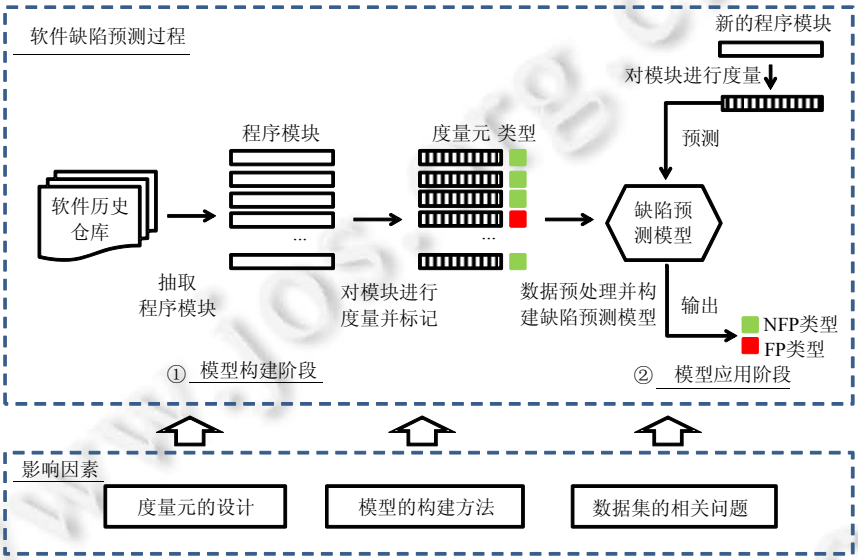

Fig.1 Static software defect prediction research framework using defect-proneness as prediction target

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