软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2016,27(12):3014−3029 [doi: 10.13328/j.cnki.jos.004869]

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

基于演化数据的软件缺陷预测性能改进*

王丹丹[1], 王 青[1,2]

[1](中国科学院 软件研究所 互联网软件技术实验室,北京 100190)

[2](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

通讯作者: 王丹丹, E-mail: wangdandan@itechs.iscas.ac.cn

摘 要: 软件持续演化已经是不争的事实,演化意味着需求的变化,也就必然导致了缺陷的不断产生.现有的缺陷预测技术多偏重于基于软件工作制品,如文档、代码、测试用例等的属性来预测缺陷,但如果把软件看作一种物种,其生命周期内的演化本质上是一个物种的逐步进化,其缺陷的表现也必然带着该物种的特征,而且还受到进化历史中的演化轨迹的影响.已有一些研究人员开始研究软件演化过程,并提出了一些演化度量元.研究和提出了可以刻画软件演化轨迹的两类演化度量元,并通过案例研究,建立缺陷预测模型.在 6 个著名开源软件数据集上训练和验证了由软件演化度量元建立的缺陷预测模型,获得了良好的预测性能,验证了演化度量元对缺陷预测性能的改进.

关键词: 缺陷预测;软件演化;演化度量元

中图法分类号: TP311

中文引用格式: 王丹丹,王青.基于演化数据的软件缺陷预测性能改进.软件学报,2016,27(12):3014−3029. http://www.jos.org.cn/1000-9825/4869.htm

英文引用格式: Wang DD, Wang Q. Improving the performance of defect prediction based on evolution data. Ruan Jian Xue Bao/Journal of Software, 2016,27(12):3014−3029 (in Chinese). http://www.jos.org.cn/1000-9825/4869.htm

Improving the Performance of Defect Prediction Based on Evolution Data

WANG Dan-Dan[1], WANG Qing[1,2]

[1](Laboratory for Internet Software Technologies, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](State Key Laboratory of Computer Science (Institute of Software, The Chinese Academy of Sciences), Beijing 100190, China)

Abstract: It is an undisputed fact that software continues to evolve. Software evolution is caused by requirement changes which often result in injection of defects. Existing defect prediction techniques mainly focus on utilizing the attributes of software work products, such as documents, source codes and test cases, to predict defects. Consider an evolving software as a species and its development process as a natural species' evolutionary process, the injection of defects may have the characters of a species and will be impacted by its evolution. A great many of researchers have studied the process of software evolution and proposed some evolution related metrics. In this study, a set of new metrics is first proposed based on evolutionary history to characterize software evolution process, and then a case study on building defect prediction models is presented. Experiments on six well-known open source projects achieved good performance, demonstrating the effectiveness of the proposed metrics.

Key words: defect prediction; software evolution; evolution metrics

为了更好地满足用户的需求、提高软件产品的市场竞争能力,一个软件产品需要不断地演化来适应需求的变化.每次软件演化都存在引入缺陷的风险,为了准确地检测缺陷,合理地安排测试工作量,许多缺陷预测技术应运而生.不过,目前大多数缺陷预测技术都是基于软件工作制品的属性进行预测,如需求、设计等的文档和代

* 基金项目: 国家自然科学基金(91318301, 91218302, 61432001)
Foundation item: National Natural Science Foundation of China (91318301, 91218302, 61432001)
收稿时间: 2015-01-26; 修改时间: 2015-03-18; 采用时间: 2015-06-12

码等,很多与工作制品相关的度量元被提出来.在这些度量元中,代码类和设计类的度量元被研究人员和实践者广泛的接受并应用,比如 Halstead[1],McCabe[2],CK[3]等.然而,软件都不是一蹴而就的,所有成功的软件都经历了一个漫长的演化历史,并不得不继续保持演化.演化过程中累积了大量的制品和数据,这些制品刻画了该软件品种的本质特征,而过程数据必然可以反映出其作为一种物种的演化轨迹.这些数据是否有助于我们建立更好的缺陷预测模型?这就是本文研究工作的主要动机.

自从软件演化这个概念被提出来,已有研究人员挖掘出了一些与演化相关的度量元,应用在缺陷预测领域.例如:Moser 等人[4]使用文件存在的周期作为一个缺陷预测的度量元;Kpodjedo 等人[5−7]提出了两个通过 Error-Correcting Graph Matching(ECGM)算法计算的新的度量元,这两个度量元分别是类的排名和演化成本,通过这两个度量元,我们可以标出最重要的和变化最频繁的类;D'Ambros 等人[8]讨论了代码演化和软件缺陷的关系,他们提出把代码度量元的变化作为预测软件缺陷的度量元.

软件演化会导致代码的变化,其在代码层次上主要体现在两个方面:一个是新增需求所变动的代码,另一个是修改需求所变动的代码.本文根据一个 Java 类是否存在演化历史,把类分为新增类(new)和已有类(old)两种,以代码类度量元刻画其品种的本质特征.对于演化数据,后者自身有演化数据;前者自身虽没有演化数据,但由于软件组件间的依赖性以及存在于同样的开发和运行环境,新增的类也必然受到该软件品种一些固有演化模式的影响.本文主要针对已有类,对于新增类的分析是下一步的工作之一.本文首先建立基于软件缺陷信息的软件演化矩阵来分析软件中类的演化过程,针对该种软件已有 Java 类的演化过程,提出了两类代码演化度量元:一类是与演化模式相关的度量元,它们是粗粒度的度量,主要是在版本级别上度量 Java 类的变化,比如类存在的时间\出现缺陷的概率和类持续不出现缺陷的时间;另一类是代码变更度量元,它们是细粒度的度量元,主要是在代码级别上度量两个连续版本中代码属性的变化,比如代码行数的变化和代码复杂度的变化等.

本文利用 6 个著名开源软件的数据建立缺陷预测模型,进行案例研究.实验结果表明:在代码、代码变更和演化模式的 7 种组合度量元建立的模型中,仅用演化模式度量元的模型在召回率和 F 值上有着最好的预测性能;代码和演化模式的组合度量元建立的模型在 AUC(area under ROC-curve,ROC 曲线下面积)上的表现最好,也就是说,有着更好的召回率和更低的假正率;7 种组合度量元在精确度上差别不显著.此外,除代码度量元外,最近版本数据建立的模型比全部历史数据建立的模型具有更好的预测性能.这说明在软件进化历程中,最近的演化轨迹对产品的影响更大,这也符合自然界进化的规律.代码类度量元表征产品的本质特征随时间的影响不大.

本文第 1 节介绍软件演化和缺陷预测相关研究工作.第 2 节给出软件演化度量元.第 3 节介绍使用的数据集、数据预处理方法、预测方法和性能评价指标.第 4 节给出实验与结果分析;实验分析软件演化对已有类缺陷预测性能的影响;使用 6 个开源软件作为数据集,4 种常用的方法作为建模方法,分析不同软件演化度量元对缺陷预测性能的影响,并与代码度量元进行对比,从而验证软件演化度量元的有效性.最后总结全文并讨论下一步的研究工作.

1 相关工作

1.1 软件缺陷预测研究

本文之前的研究[9]调研了 2005 年~2012 年间 219 篇缺陷预测论文,这些论文关注 10 类不同的预测目的,例如预测易出缺陷模块、缺陷数目、缺陷目睹、缺陷优先级、缺陷严重程度等.调研结果表明:预测易出缺陷模块,也就是预测一个代码模块是否存在缺陷,是研究人员最关注的预测目的.因此,本文关注在预测 Java 代码类是否存在缺陷.除此之外,该调研论文还发现:代码类和设计类度量元是缺陷预测中最常用的两类度量元,并且获得了较好的预测性能.因此,本文选择常用的代码和设计类度量元作为基准度量元.

1.2 软件演化研究

软件需要不断演化去适应用户或市场的需求,在其漫长的演化过程中累积了大量的制品和数据,这些制品刻画了该软件品种的本质特征,而过程数据必然可以反映出其作为一种物种的演化轨迹.自软件演化这个概念

被提出来,软件演化特征现在已经成为多版本软件产品的一个重要特征.Gall 等人[10]分析了电信产品多个版本的结构,他们发现:虽然系统的整个开发过程比较稳定,但随着产品的演化,不同的版本体现了完全不同的特征,比如模块的规模和改变的比率.

经过多年的研究,研究者已经提出了很多方法来形象化地刻画软件的演化过程,比如:Gall 等人[11]提出了基于时间、软件系统结果和一种属性值的表示方法来检测软件的历史信息;Lanza[12]提出了一种演化矩阵的方法,演化矩阵代表了软件产品的各个类文件的演化,矩阵的每一行代表了一个类文件,每一列表示该类文件在不同版本中的变化;Wu 等人[13,14]使用光谱来研究软件的演化;Girba 等人[15]根据软件在运行过程中的一系列需求,提出了演化分析方法 Hismo,通过可视化的手段对软件历史行为进行建模,从而刻画软件演化过程;Robbes[16]为软件中操作和信息的变化进行建模,以此来监控软件的运行.本文结合软件缺陷信息和演化矩阵[12]得到了基于软件缺陷信息的软件演化改进矩阵来分析软件的演化过程.

1.3 软件演化引入缺陷问题的研究

对于演化的软件产品,特征化软件的演化行为可能对软件缺陷的预测有改进作用.现在很多软件演化分析技术要么集中在源代码,要么集中在其他与源代码有间接关系的数据源上,这些分析已经被用在了缺陷预测上.

2000 年,Graves 等人[17]提出改变相关度量元,比如,修改的次数(比如文件的年龄、修改的规模等)比代码度量元(如麦凯布圈复杂度)有着更好的缺陷预测效果.2005 年,Nagappan 和 Ball[18]成功地在相对代码变化度量元基础上,使用统计回归方法去预测软件缺陷密度.他们的实验结果表明:与绝对代码变化度量元相比,相对代码变化度量元对于缺陷密度的预测有着更好的预测能力.同年,Sliwerski 等人[19]通过分析指出,变更规模和时间对缺陷引入有影响.2006 年,Bell 等人[20]采用负的二项回归方法去预测更容易产生缺陷的文件(20%的文件,它们包含 75%的软件缺陷).他们使用了文件的年龄和它们的变化历史、编码语言等作为预测变量.实验结果表明:与只是用代码行作为预测变量相比,使用改变相关度量元能够显著地提高缺陷的预测精度(大概两倍).2007 年,Zimmermann 等人[21]把 Eclipse 项目的 3 个版本缺陷数据与代码关联起来,并且给出了代码的规模和复杂度相关的度量数据.他们的分析表明,复杂度度量元与发布前和发布后的缺陷有显著的相关性.Aversano 等人[22]基于文本分类的思想,以代码变更作为学习和预测对象,预测一个代码变更是否会引入缺陷.2008 年,Moser 等人[4]对比了代码度量元和改变相关度量元的预测能力,结果表明:对于 Eclipse 数据来说,改变相关度量元比代码度量元更有效.D'Ambros[23]使用历史的依赖性和缺陷信息去分析软件的演化,把缺陷作为演化的实体,使用从源代码中检测到的设计缺陷来预测未来可能出现缺陷的位置.Kpodjedo 等人[5−7]提出了两个新的度量元来预测缺陷,它们是从错误-修正匹配图算法(error-correcting graph matching algorithm,简称 ECGM)中得来——类的排序(class rank,简称 CR)和演化成本(evolution cost,简称 EC).这两个度量元度量了最重要和改变最多的类文件.Kim 等人[24]针对文件级变更,分析了一条代码变更是否会引入缺陷,从 3 个不同的源(变更)提取度量数据,然后建立预测模型.2009 年,Abreu 等人[25]发现,开发人员之间过于频繁地交流会增加变更引入缺陷的风险.2010 年,D'Ambros[8]分析了源代码的演化和软件缺陷的关系.他提出使用源代码的改变这个度量元来预测版本发布后的缺陷,并且得出:相比源代码度量元,源代码的改变度量元具有更好的和稳定的解释性和预测能力.2011 年,Eyolfson 等人[26]发现,时间对缺陷引入有影响.Rahman 等人[27]发现,代码的所有权和开发人员的经验对变更质量有影响.2013 年,Shivaji 等人[28]使用属性选择方法对属性进行降维,从而提高基于代码变更的缺陷预测性能.

2014 年,原子等人[29]提出了面向细粒度源代码变更的缺陷预测方法.他们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,并提出一种伴随概念回顾的动态窗口学习机制来实现长时间的稳定预测.

本文是在 Java 类级上预测易出缺陷的类,使用两种类型的演化度量元:一种是演化模式相关的度量元,另一类是代码变更类的度量元,并与传统的代码度量元做对比,从而验证演化度量元是否对缺陷预测性能有改进作用.现有相关研究大多针对哪些属性对缺陷引入有影响或改进现有缺陷预测模型性能,而本文考虑了两种不同类型的演化度量元,不仅验证了演化相关度量元是否比传统代码度量元预测效果好,而且分析了哪类演化度量元预测效果相对较好.

2 软件演化度量元

在软件漫长的演化历史中,软件的过程数据可以反映出其作为一种物种的演化轨迹,因此,本文结合演化矩阵[12]和缺陷信息提出了基于软件缺陷信息的改进的 Java 类演化矩阵来分析软件演化过程.该改进的演化矩阵展示了软件中 Java 类的演化轨迹.矩阵的每一行代表一个类,每一列代表一个演化版本,虚框代码该类在该版本下存在缺陷.

图 1 给出了演化矩阵的例子.演化矩阵展示了 8 个类在 5 个演化版本中的变化.

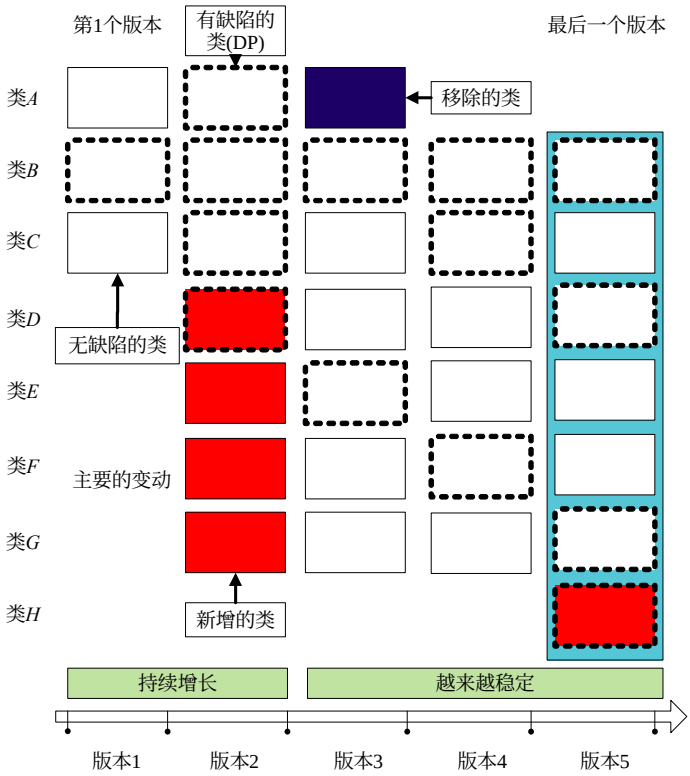


Fig.1 An improved evolution matrix based on defect information

图 1 基于软件缺陷信息的改进的软件演化矩阵

本文缺陷预测的粒度为 Java 代码中的类(class),主要研究 Java 代码类是否存在缺陷的预测方法.根据 Java 类是否存在演化历史,本文将 Java 类分为新增类和已有类两种,以代码类度量元(见表 1)刻画其品种的本质特征.表 1 中给出了常用的 20 个代码度量元,这些度量元在以往的研究[30,31]中使用过.这些代码度量元根据度量的方面不同,可以分为 5 类:复杂度、耦合性、内聚性、继承性和规模.

因为新增的 Java 类在历史版本中从来没有出现过,不存在其演化轨迹;而已有的 Java 类在已有的历史版本中曾经存在过,在历史版本中存在演化轨迹.因此,本文主要针对已有 Java 类作分析,对于新增类的分析会是未来工作之一.从图 1 中可以看出已有类在之前版本中的演化轨迹.以版本 5 为例:类 *B*~类 *G* 在版本 1~版本 4 中有着演化轨迹;而类 *H* 是没有的,原因是类 *H* 是新增的类,不存在于之前任意历史版本中.针对已有类,本文提出了两类代码演化度量元来刻画软件的演化过程.

(1) 演化模式相关的度量元,它们是粗粒度的度量,主要是在版本级别上度量 Java 类的变化,比如类存在的时间\出现缺陷的概率和类持续不出现缺陷的时间;

(2) 代码变更度量元,它们是细粒度的度量元,主要是在代码级别上度量两个连续版本中代码属性的变

化,比如代码行数的变化和代码复杂度的变化等.

这些演化度量元会在下面两节中详细介绍.表 2 中给出了本文中使用的度量元的汇总.

Table 1 Code metrics

表 1 代码类度量元

类型	度量元	简写	描述
复杂性	平均方法复杂度	*amc*	比如 Java 字节代码的数目
	平均圈复杂度	*avg_cc*	一个类的平均圈复杂度
	最大圈复杂度	max_*cc*	同一种方法中方法的最大圈复杂度
耦合性	传出耦合度	*ca*	使用该具体类的其他类的个数
	方法间的耦合度	*cbm*	与所有继承方法耦合的新的/重新定义的方法的总数
	对象之间的耦合	*cbo*	类与其他类的耦合度,即类调用其他类中方法或接口的次数
	输入耦合度	*ce*	使用某类的其他类的个数
	继承耦合度	*ic*	与一个已给类相耦合的父类的个数
内聚性	类间内聚度	*cam*	每种方法中方法参数的不同类型个数的总合 除以整个类中不同方法参数类型和方法数目的乘积
	类内聚缺乏度	*locm*	内聚性缺乏度,即对某一实例变量无共同引用的方法对数
	另一种 类内聚缺乏度	*locm*3	如果 m 是方法个数,a 是一个类中属性的个数,$\mu(a)$是访问一个属性的方法个数,那么 $lcom3=\left(\left(\frac{1}{a}\sum_j^a \mu(a_j)\right)-m\right)\Big/(1-m)$
继承性	继承树的深度	*dit*	类在继承树中位置的深度
	聚合的度量	*moa*	该度量元是统计用户定义类中数据声明个数
	功能抽象的度量	*mfa*	一个类继承的方法数加上该类成员方法访问的方法
规模	代码行数	*loc*	度量代码量
	类孩子的数目	*noc*	度量一个类直系后代的个数
	类的响应度	*rfc*	响应类中一个消息所涉及的方法数
	公用方法的个数	*npm*	统计一个类中声明为公有的方法个数
	每个类的加权方法数	*wmc*	类中方法的个数(假设为所有的方法设置相同的权重)
	数据存取度量	*dam*	私有(受保护)属性在所有属性中所占比率

Table 2 Metrics used in our study

表 2 本文中使用到的度量元

度量元		缩写	个数	描述
基准度量元	代码度量元	CM	20	度量复杂度、耦合性、内聚性、继承性和规模
软件演化度量元	演化模式相关度量元	EP	3	度量软件演化过程中类的演化模式
	代码变更度量元	{*Delta*,*Churn*}	40	度量连续两个版本间代码的变化,包括两类度量元

2.1 演化模式相关度量元

演化模式相关的度量元主要是在版本级别上刻画一个 Java 类在历史版本中的演化轨迹.从图 1 中可以看出:为了描绘 Java 类的演化特征,首先需要了解 Java 类存在的时间,比如在版本 5 中,一共有 6 个已有类,其中,类 *B* 和类 *C* 存在了 4 个版本,类 *D*~类 *F* 和类 *G* 存在了 3 个版本;其次需要了解 Java 类可能出现缺陷的概率,例如在版本 5 中,类 *B* 在前 4 个历史版本中都存在缺陷,类 *C* 在 4 个历史版本中的两个中存在缺陷等,这些类出现缺陷的概率可能会影响到下一个版本中出现缺陷的可能性;一个 Java 类最后出现缺陷的位置也可能影响下一个版本中缺陷的出现,可能越邻近的版本出现缺陷,那其下一个版本中出现缺陷的概率越大.因此,我们选择了相应的 3 个度量元来刻画软件演化过程中类的演化模式.

(1) 类的年龄(age).该度量元是指一个类存在的时间,以版本为度量单位.

(2) 类出现缺陷的可能性(probability of a class has defects,简称 PCD).该度量元度量一个类在当前版本以前百分之多少的版本存在缺陷:

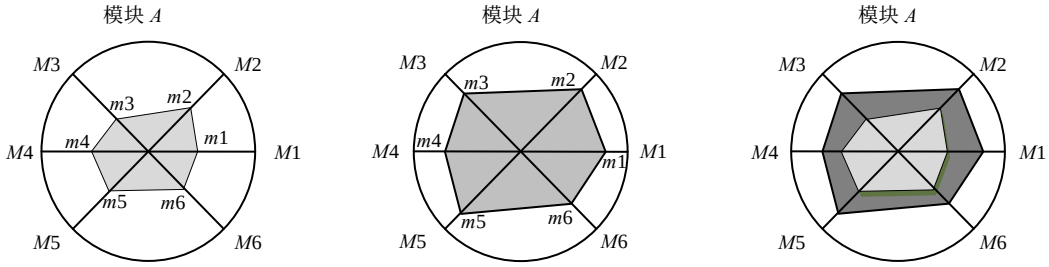
$$PCD=\frac{\text{一个类出现缺陷的版本数}}{\text{类的年龄}} \tag{1}$$

(3) 类连续不含有缺陷的周期(duration).该度量元是度量当前版本的一个类有多长时间没有出现过缺陷.

例如在图 1 中,对于版本 5 中的类 *D* 来说,它不含有缺陷的周期是 2,因为在版本 5 相邻的连续两个版本中,类 *D* 不存在缺陷,所以周期是 2.

2.2 代码变更度量元

软件演化带来代码的变化,代码的变化可能体现在代码度量元的变化上.图 2 给出了软件演化中,一个软件模块代码属性的变化.模块 *A* 的属性由 6 个代码度量元度量:*M*1,*M*2,*M*3,*M*4,*M*5 和 *M*6.图 2 中给出了模块 *A* 在连续两个版本中属性值的变化情况,其中,图 2(b)中灰色部分代表代码属性的变化.对于代码的变更,本文提出两类度量元来度量——代码相对变化(code delta)和代码绝对变化(code churn).这两类度量元曾被 Hall 和 Munson[32] 用来度量元软件的演化.因为代码相对变化和代码绝对变化是在 20 个代码度量元基础上计算得到的,因此不再详细介绍每个代码变更度量元.



(a) 版本 1 (b) 版本 2 (c) 版本 2 与版本 1 的差异

Fig.2 An example of code changes[8]

图 2 代码属性变化的一个例子[8]

(1) 代码相对变化度量元(delta)

代码相对变化度量相邻两个版本代码属性的变化.一个类 *i* 在属性 *j* 上的代码相对变化表示为:

$$Delta_{ij}(n)=Metric_{ij}(n)-Metric_{ij}(n-1) \tag{2}$$

$Metric_{ij}(n)$是类 *i* 在版本 *n* 中属性 *j* 的取值,$Metric_{ij}(n-1)$是类 *i* 在版本 *n*−1 中属性 *j* 的取值.本文标记 20 个代码度量值 *amc*,*avg_cc*,…的相对变动为 *Damc*,*Davg_cc* 等.20 个代码度量值的定义见表 1.

(2) 代码绝对变化度量元(churn)

代码绝对变化被 Kpodjedo 等人[5]使用过,它是代码相对变化的绝对值.一个类 *i* 在属性 *j* 上的代码绝对变化表示如下:

$$Churn_{ij}(n)=|Delta_{ij}(n)|=|Metric_{ij}(n)-Metric_{ij}(n-1)| \tag{3}$$

本文标记 20 个代码度量值 *amc*,*avg_cc*,…的绝对变动为 *Camc*,*Cavg_cc*,*C*max_*cc* 等.

3 研究案例

3.1 数据集

为使实验更加接近多版本缺陷预测的实际应用环境,本文实验中选择 Promise[33]开源项目作为数据集,训练集和测试集都在同一个产品中,表 3 给出了选择的 6 个开源项目数据,它们含有至少 4 个版本数据,其中,第 1 个版本作为基准版本.选择的理由主要有:

(1) 开源项目的数据容易获取并且是免费的.除此之外,缺陷预测研究领域有大量公开的已整理好的开源项目数据集可以使用;

(2) 商业项目出于安全、保密、市场竞争等原因,极少有软件开发组织乐意贡献自己组织内商业项目的历史数据.

Table 3 Datasets used in our study

表 3 本文中所使用的数据集

项目名称	项目描述
ANT	软件自动化构建工具,可以将软件的编译、测试、部署等活动集成在一起
CAMEL	企业软件集成框架
JEDIT	跨平台文本编辑器
POI	开源的 Java 读写 Excel、WORD 等微软 OLE2 组件文档的工具包
XALAN	XLST 处理引擎,将 xml 文档转化为 html,text 等其他文档格式
XECRCES	xml 解析器

实验选取了 6 个开源项目共 26 个版本的数据.所有项目均使用 Java 语言开发,缺陷预测的粒度为 Java 代码中的类(class).对 Java 类的描述,采用 20 个静态代码特征进行度量.对 Java 类的分类标准为:对开源项目,代码发布后 6 个月内有用户提交 bug,则将该类视为易错模块;否则为非易错模块.表 4 中,规模表示数据集中样本的数目,*#DP* 表示易错类的数目,*%DP* 表示易错类占样本总数的比例.每个所选项目的第 1 个版本作为历史版本的第 1 个版本,新增类和已有类是根据所有历史版本来定的.比如 ANT 项目中,1.4 版本的新增和已有类是以历史版本 1.3 为基准的,1.5 版本的新增和已有类是以历史版本 1.3,1.4 为基准的.本文主要针对已有类进行缺陷预测分析.

Table 4 Basic information of datasets used

表 4 数据集的基本信息

产品及其版本号		标记	所有类			已有类			新增类		
			规模	*#DP*	*%DP*	规模	*#DP*	*%DP*	规模	*#DP*	*%DP*
ANT	1.3		125	20	16.0	–	–	–	–	–	–
	1.4	V1	178	40	22.5	125	22	17.6	53	18	34.0
	1.5	V2	293	32	10.9	168	23	13.7	125	9	7.2
	1.6	V3	351	92	26.2	293	73	24.9	58	19	32.8
	1.7	V4	745	166	22.3	355	118	33.2	390	46	11.8
CAMEL	1		339	13	3.8	–	–	–	–	–	–
	1.2	V1	608	216	35.5	262	87	33.2	346	129	37.3
	1.4	V2	872	145	16.6	577	117	20.3	295	28	9.5
	1.6	V3	965	188	19.5	857	171	20.0	108	17	15.7
JEDIT	3.2.1		272	90	33.1	–	–	–	–	–	–
	4.0	V1	306	75	24.5	265	62	23.4	41	13	31.7
	4.1	V2	312	79	25.3	291	77	26.5	21	2	9.5
	4.2	V3	367	48	13.1	291	41	14.1	76	7	9.2
	4.3	V4	492	11	2.2	225	5	2.2	267	6	2.2
POI	1.5		237	141	59.5	–	–	–	–	–	–
	2.0RC1	V1	314	37	11.8	224	26	11.6	90	11	12.2
	2.5.1	V2	385	248	64.4	314	219	69.7	71	29	40.8
	3.0	V3	442	281	63.6	382	261	68.3	60	20	33.3
XALAN	2.4		723	110	15.2	–	–	–	–	–	–
	2.5	V1	803	387	48.2	689	337	48.9	114	50	43.9
	2.6	V2	885	411	46.4	766	349	45.6	119	62	52.1
	2.7	V3	909	898	98.8	861	851	98.8	48	47	97.9
XECRCES	Init		162	77	47.5	–	–	–	–	–	–
	1.2.0	V1	440	71	16.1	116	35	30.2	324	36	11.1
	1.3.0	V2	453	69	15.2	433	62	14.3	20	7	35.0
	1.4.4	V3	588	437	74.3	328	209	63.7	260	228	87.7
平均值			483	169	32.0	391	157	34.0	144	39	30.8
中位数			413	91	23.5	304	82	25.7	99	20	32.2

3.2 数据预处理

以往的研究表明:适当的数据预处理,可以显著提高缺陷预测的性能.本文实验中,主要采用了以下数据预处理操作.

(1) 数据采样

由于缺陷分布的 2-8 规律,一般缺陷预测数据集中易错模块所占的比率为 20%左右.这种数据的非均衡分布会影响缺陷预测模型的预测性能,为降低其影响,在进行模型训练前需要对训练数据进行重采样处理.Menzies 等人[34]和 He[30,35]对常用的采样方法进行了对比,发现欠抽样(under-sampling)能够显著提高缺陷预测的性能.基于此,本文实验中使用 WEKA 工具对训练集进行欠抽样处理.设有缺陷预测训练集 D,其中,易错模块数目为 M,非易错模块数目为 N.本文实验中使用的欠抽样,采用有放回随机抽样来抽取和易错模块相同数量的模块数(N'),即 $N'=M$.

(2) 数据规范化

为避免不同度量元特征取值范围的差异给预测模型造成影响,实验中将数据集的特征都规范化到[0,1]区间.假设输入数据集 D,其中的每个特征为 M_i.首先,计算 M_i 取值的最小值 $\min(M_i)$和最大值 $\max(M_i)$;然后,对于 D 中每条样例 I_j,计算规范化后的 M_i 的值为

$$I'_{ji}=\frac{I_{ji}-\min(M_i)}{\max(M_i)-\min(M_i)} \tag{4}$$

(3) 数据离散化

实验中,采用 10-等频分箱方法对连续性度量元进行数据离散化.假设输入数据集为 D,其中的每个特征为 M_i.首先,将所有样例按 M_i 的取值升序排列;然后,计算每条样例 I_j 的离散化 M_i 取值为

$$I'_{ji}=10\times index(I_j)/size(D) \tag{5}$$

其中,$index(I_j)$表示样本 I_j 的排列下标,“/”操作为向下整除操作.

3.3 预测方法

实验中,选用了 4 种常用的分类预测算法:朴素贝叶斯(NB)[36]、J48 决策树(J48)[37]、随机森林(RF)[38]和二元逻辑回归(BLR).本文实验对预测模型的选择基于如下理由:

(1) 在本文之前的研究中[9],调研了 2005 年~2012 年间的缺陷预测模型,发现朴素贝叶斯(NB)、J48 决策树(J48)、随机森林(RF)和二元逻辑回归(BLR)是预测易出缺陷模块最常用的建模方法;

(2) Menzies 和 Lessmann 等人通过对比实验发现:朴素贝叶斯模型和随机森林模型在静态代码度量数据上表现良好,预测性能高于支持向量机等复杂模型[39,40];

(3) 逻辑回归模型和朴素贝叶斯模型是缺陷预测相关研究中使用最广泛的两种对比基准[41−45].J48 决策树方法在很多研究中亦得到很好的预测性能[46,47].

不同的方法在不同数据集上有不同的表现,在其他数据集上表现好的方法在本文中所用数据集就不一定好.本文侧重在度量元对缺陷预测性能的改进,选择预测方法是为了构建模型来验证不同度量元的性能,期望找到对不同预测方法都敏感的度量元,即在不同的预测方法下预测性能都比较好的度量元.因此,本文选择了以上常用并且性能较好的 4 种预测方法.

3.4 模型评价

3.4.1 评价方法

本文是针对基于软件演化历史的易出缺陷模块预测.在软件演化过程中,新需求不断加入,对开发环境、工具的熟悉程度,个人技能的不断提高,开发团队人员不断更迭,这些外界环境的变化导致软件自身特征和软件缺陷的生成规律也在不断改变.Ekanayake 等人[48]发现,时间邻近的实例与时间相距较远的实例相比具有更好的预测性能.为了验证软件演化历史是否能够提高缺陷预测性能,并且时间邻近的实例是否有更好缺陷预测性能,本文采用两种跨版本验证方法(即,在某一项目版本的历史数据上训练缺陷预测模型后对该项目的后续版本进行预测).

(1) ALL.该类是使用所有的历史版本的缺陷数据来预测下一个版本的缺陷:$\{V_1,V_2,\ldots,V_{n-1}\}\rightarrow V_n$;

(2) PRE.该类是使用前一个版本的缺陷数据来预测下一个版本的缺陷 $V_{n-1}\leftarrow V_n$.

3.4.2 评价指标

本文使用精确度(precision,简称 P)、召回率(recall,简称 R)、*F*-值(F)和 ROC 曲线下面积(AUC)来评价缺陷预测模型的性能.软件缺陷预测问题是典型的二分类预测问题(易错或非易错),相关研究中常将易错模块视为正类样本(positive),将非易错模块视为负类样本(negative),缺陷预测结果共分 4 类,见表 5.

Table 5 Confusion matrix

表 5 混淆矩阵

		预测值	
		易错	非易错
真实值	易错	TP(true positive)	FN(false nagative)
	非易错	FP(false positive)	TN(true nagative)

(1) 精确率(precision)

精确率也称真正率(True positive rate),表征预测模型识别的正类样本中正确的比率,精确率指标越接近 1 越好.

$$precision = \frac{TP}{TP + FP} \tag{6}$$

(2) 召回率(recall)

召回率也称检测率(probability of detection,简称 PD)或灵敏性(sensitivity),表征预测模型对正类样本的识别程度,召回率指标越接近 1 越好.

$$recall = PD = \frac{TP}{TP + PN} \tag{7}$$

(3) *F* 值(*F*-measure)

F-measure 是对 Recall 和 Precision 的综合.

$$F\text{-}measure = \frac{(1+\beta^2) \times recall \times precision}{\beta^2 \times recall \times precision} \tag{8}$$

其中,β用于调节比重,β越小 Recall 的比重越小.当β为 1 时,Recall 和 Precision 的比重相同.缺陷预测研究中,通常默认β取值为 1.*F*-measure 取值越高,表明预测模型的性能越好.

(4) ROC 曲线下面积[49]

ROC 曲线方法是一种比较两个分类模型的有用可视工具,它不受数据非均衡分布的影响,在缺陷预测研究中被广泛使用.ROC 曲线越靠近左上角越好,表明预测结果的召回率高且假正率低.为方便模型性能的比较,本文通过计算 AUC 来评估模型的准确率.面积越接近 0.5,对应的模型准确率越低,而完全准确地模型的面积为 1.

4 实验与结果分析

本节分析了已有类的缺陷预测模型.为了验证软件演化度量元是否能够提高缺陷预测性能,本文设置使用代码度量元建立的缺陷预测模型为基准模型,并在 6 个公开数据集上验证软件演化度量元的有效性.

4.1 度量元效力分析

在 6 个项目数据基础上对 63 个度量元(20 代码度量元+3 演化相关度量元+40 代码变更度量元)进行 Pearson 相关性分析,结果表明,度量元之间存在相关性.例如 *rfc* 与 *wmc* 有很强的正相关,其中,相关系数为:ANT(0.896),CAMEL(0.885),JEDIT(0.866),POI(0.859),XALAN(0.862),XERCES(0.904).为了去除度量元间相关性对缺陷预测性能的影响,本文选择了 WEKA[50]中的 CFS(CfsSubsetEval)[51]属性选择算法和 BestFirst[52]搜索方法去寻找对缺陷预测最重要的度量元,其中,

- CFS 属性选择算法综合考虑单一属性的预测值和属性间的重复度,根据属性子集中每一个特征的预测能力以及它们之间的关联性进行评估,然后挑选那些与缺陷有高度关联但相互之间关联程度却较低

的属性.也就是说,CFS 属性选择算法既考虑了属性间的相关性,又考虑了属性与缺陷之间的相关性,因此本文中没有再做相关分析;

- BestFirst 搜索方法是回溯的贪婪搜索.在 CFS 属性选择方法中,对于每个数据,选择的度量元被赋值为 1,否则为 0.本文采用两种跨版本验证,共 22 组实验,属性选择结果见表 6.

Table 6 Results of metrics selection

表 6 属性选择结果

项目	训练集	测试集	属性选择后的度量元
ANT	V1	V2	*ce,moa,cloc,cmfa*
	V2	V3	*rfc,ce,lcom3,cbm,cdam,cmoa,awmc,amax_cc*
	V1,V2	V3	*rfc,ce,moa,ic,cdam,awmc,anoc,arfc,aloc,acam,amax_cc*
	V3	V4	*wmc,rfc,loc,moa,cam,clcom,cloc,cavg_cc,age*
	V1,V2,V3	V4	*wmc,rfc,ce,lcom3loc,moa,crfc,clcom,clcom3,cdam,amax_cc,age*
XALAN	V1	V2	*noc,rfc,lcom,loc,dam,amc,crfc,cmfa,anpm,alcom3,duration*
	V2	V3	*noc,rfc,lcom3,loc,cam,amc,crfc,ccam,duration,pcd*
	V1,V2	V3	*rfc,lcom3,loc,amc,crfc,duration*
CAMEL	V1	V2	*noc,cbo,dam,cbm,amc,max_cc,avg_cc,cwmc,cnoc,clcom,cca, cnpm,camc,ace,anpm,adam,duration,pcd*
	V2	V3	*npm,clcom3,duration,pcd*
	V1,V2	V3	*noc,npm,max_cc,cdit,cca,cloc,anpm,duration,pcd*
JEDIT	V1	V2	*dit,rfc,crfc,cce,cmoa,cmax_cc,awmc,anpm,aamc,duration,pcd*
	V2	V3	*rfc,npm,ccbo,cnpm,adam,duration,pcd*
	V1,V2	V3	*rfc,ce,ccbo,cmax_cc,arfc,amoa,duration,pcd*
	V3	V4	*ce,npm,lcom3,max_cc,clcom,cmoa,cmax_cc,arfc,duration,pcd*
	V1,V2,V3	V4	*rfc,npm,lcom3,moa,*max_cc,cnpm,cmax_cc,arfc,duration,pcd
POI	V1	V2	*dit,lcom,avg_cc,cnpm,alcom3,aloc,amoa,amax_cc*
	V2	V3	*dit,lcom3,dam,cbm,ccam,duration*
	V1,V2	V3	*lcom3,pcd*
XERCES	V1	V2	*cam,cnoc,cmoa,cmfa,ccbm,duration,pcd*
	V2	V3	*ca,ce,lcom3,dam,moa,ic,cbm,avg_cc,ccbo,ccam,camc,age*
	V1,V2	V3	*ca,ce,lcom3,ic,cbm,avg_cc,cwmc,cnoc,cdam,cmoa,acam,duration,pcd*

根据度量元在 22 组实验中出现的次数进行排名,前 10 个最重要的度量元为 *Duration*(score=14),*PCD*(score=12),*rfc*(score=11),*lcom*3(score=10),*ce*(score=8),*moa*(score=6),*npm*(score=5),*loc*(score=5),*cbm*(score=5),*Crfc*(score=5)和 *Cmoa*(score=5).实验结果可以看出:36%的所选度量元属于两类软件演化度量元——2 个属于软件模式相关度量元,2 个属于代码变更度量元,其中,2 个演化模式相关度量元排在前 2.除此外,与规模、耦合性、内聚性和继承相关的度量元也在前 10 中出现.

针对选出来的前 10 个最重要的度量元,我们做了度量元与缺陷间的 Spearman 相关性分析,结果显示,该 10 个度量元与一个类是否有缺陷在置信度(双测)为 0.01 时相关性是显著的.其中,*Duration*,*lcom*3 和一个类是否有缺陷是负相关的,其他都是正相关的.

4.2 不同度量元的预测性能分析

针对已有类的演化特征,本文提出了两类软件演化度量元.为验证演化度量元的性能,本文在 6 个开源软件产品上建立缺陷预测模型——ANT,CAMEL,XALAN,JEDIT,XERCES 和 POI,并且使用召回率、精度、*F*-值和 AUC 来评价预测性能.本文把使用代码度量元的缺陷预测模型作为对比的基准.表 7 给出了仅使用代码度量元(*CM*)、演化模式相关度量元(*EP*)和代码变更度量元(*delta*,*churn*)的缺陷预测模型的性能.从表 7 中可以观察到:

(1) 不管是从性能平均值来看还是中位数来看,仅使用演化模式相关度量元可以有效地提高预测的召回率和 *F*-值;

(2) 演化模式相关度量元在精确度 *P* 和 AUC 上与代码度量元差别不大;在精确度 *P* 上,演化模式相关度量元有着更小的标准差(*Stdev*),也就意味着,演化度量元比代码度量元在精确度 *P* 上更稳定.在 AUC 上,虽然演化度量元的标准差不是最小的,但是与代码变更度量元的标准差的差距不大,并且比代码度量元要小,因此,演化模式相关度量元在 AUC 上较稳定;

(3) 使用代码变更度量元的预测性能在精确度 *P* 上相差不大,但在召回率 *R*、*F* 值和 AUC 上要低于代码度量元和演化模式相关度量元;除此之外,在统计学上,代码变更度量元的预测性能在召回率和 *F* 值上显著低于演化模式相关度量元.

综上所述,与代码和代码变更度量元相比,演化模式度量元有着相对较好的预测性能.

Table 7 Performance of models built by *CM*, *EP*, *Delta* and *Churn*

表 7 仅使用 *CM*,*EP* 和{*Delta*,*Churn*}的缺陷预测模型的性能

	方法	*CM*				*EP*				{*Delta*,*Churn*}			
		P	*R*	*F*	AUC	*P*	*R*	*F*	AUC	*P*	*R*	*F*	AUC
平均值	NB	0.53	0.55	0.46	**0.69**	0.50	**0.64**	**0.53**	0.65	**0.54**	0.40	0.33	0.64
	BLR	0.48	0.54	0.43	**0.65**	**0.50**	**0.67**	**0.53**	**0.65**	0.49	0.49	0.35	0.63
	J48	**0.53**	0.53	0.46	**0.65**	0.48	**0.65**	**0.51**	0.64	0.52	0.46	0.35	0.61
	RF	0.49	0.54	0.45	**0.65**	**0.50**	0.58	**0.50**	**0.65**	**0.50**	0.48	0.35	0.63
标准差	NB	0.31	0.24	0.21	0.22	**0.25**	**0.20**	0.21	**0.14**	0.27	0.29	**0.19**	**0.14**
	BLR	0.30	0.22	**0.19**	0.18	**0.25**	**0.21**	0.21	0.15	0.26	0.30	**0.19**	**0.11**
	J48	0.29	0.23	**0.20**	0.19	**0.26**	**0.21**	0.22	0.14	0.30	0.29	**0.20**	**0.09**
	RF	0.29	**0.21**	**0.19**	0.19	**0.25**	0.23	0.22	0.15	0.27	0.31	**0.19**	**0.11**
中位数	NB	0.48	0.56	0.46	**0.78**	0.51	**0.63**	**0.54**	0.68	**0.54**	0.37	0.35	0.64
	BLR	0.41	0.54	0.41	**0.70**	0.51	**0.68**	**0.57**	0.68	**0.52**	0.49	0.37	0.61
	J48	0.48	0.45	0.48	**0.69**	0.46	**0.64**	**0.49**	0.65	**0.54**	0.50	0.34	0.61
	RF	0.48	**0.55**	0.44	**0.74**	**0.55**	0.54	**0.50**	0.66	0.53	0.48	0.36	0.61

为了验证组合度量元的预测性能是否优于单独的度量元,本文给出了代码(*CM*)、演化模式(*EP*)、代码变更度量元(*Delta*,*Churn*)及其组合度量元,共 7 组,即,3 种度量元的排列组合 $(C_3^1+C_3^2+C_3^3)$.这些度量元的预测性能如图 3 所示.7 组度量元及其组合分别如下:

(1) *M*1: *CM*, *M*2: *EP*, *M*3: {*Delta*,*Churn*};

(2) *M*4: *CM*+*EP*, *M*5: *CM*+{*Delta*,*Churn*}, *M*6: *EP*+{*Delta*,*Churn*};

(3) *M*7: *CM*+*EP*+{*Delta*,*Churn*}.

图 3 中给出了不同度量元组合在不同方法下的平均性能情况.

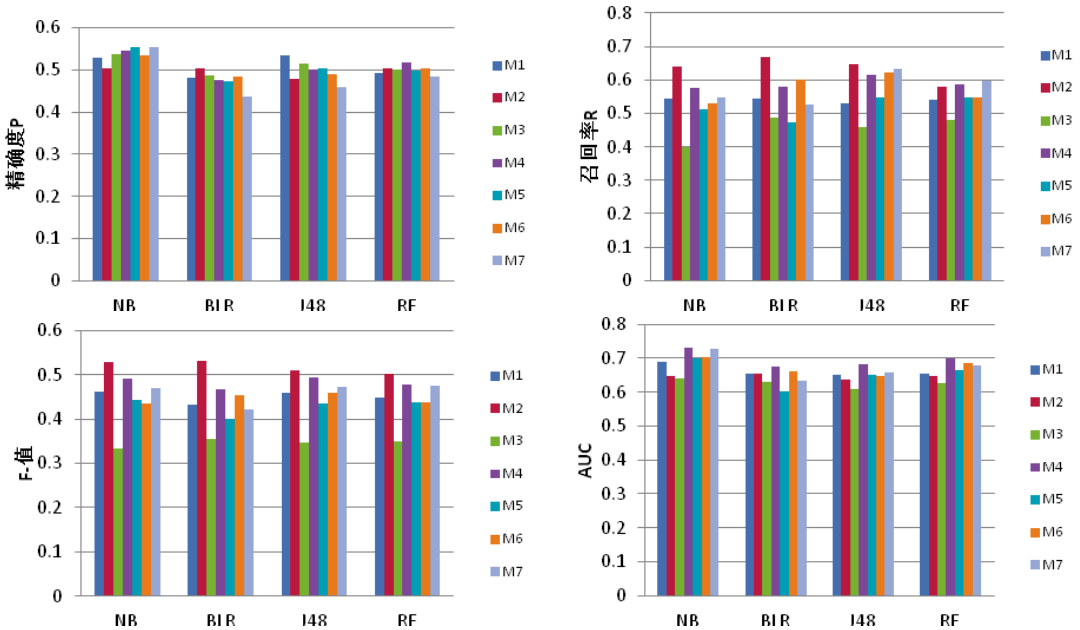


Fig.3 Performance of models built by various metrics combinations

图 3 使用不同度量元组合的模型性能

从中可以看出:

(1) 在精确度 *P* 上,7 组度量元组合在不同的方法下,性能各有不同,性能间的差距不显著:当使用朴素贝叶斯(NB)时,*M*7(*CM+EP+*{*Delta*,*Churn*})有着最好的预测性能;当使用二元逻辑回归时,*M*2(*EP*)有着最好的预测性能;当使用 J48 决策树时,*M*1(*CM*)有着最好的预测性能;当使用随机森林(RF)时,*M*4(*CM+EP*)有着最好的预测性能;

(2) 在召回率 *R* 上,在朴素贝叶斯、二元逻辑回归和 J48 决策树算法上,*M*2(*EP*)都有着最好的预测性能;在随机森林算法上,虽然 *M*7(*CM+EP+*{*Delta*,*Churn*})有着最好的性能,但 *M*2 与 *M*7 的差距也不大.即演化模式相关的度量元在不同的建模方法下都有着较好的召回率;

(3) 在 *F*-值上,不管在什么方法上,*M*2(*EP*)都有着相对最好的预测性能.即演化模式相关的度量元在不同的建模方法下都有着最好的 *F*-值;

(4) 在 AUC 上,不管在什么方法上,*M*4(*CM+EP*)都有着较好的预测性能.即代码度量元和演化模式度量元的组合有着较高的召回率且较低的假正率;

(5) 除此之外,代码变更度量元(*M*3)在召回率 *R*、*F* 值和 AUC 上的预测性能比其他度量元相对较差.

4.3 演化历史中数据集选择对预测性能的影响分析

本文选择历史版本数大于 1 的产品来验证使用前一个版本数据(PRE)的预测效果是否比使用所有版本数据(ALL)的预测效果要好,并且选择演化模式相关度量元和代码与演化相关的组合度量元来分析.表 8 给出了其平均性能的对比.

Table 8 Comparison of performance of models built on ALL/PREVIOUS data

表 8 使用所有和前一个版本数据的缺陷预测模型的性能对比

模型	度量元	方法	ALL				PRE			
			P	*R*	*F*	AUC	*P*	*R*	*F*	AUC
*M*1	*CM*	NB	0.617	**0.598**	0.505	0.787	**0.625**	0.593	**0.515**	**0.790**
		BLR	**0.558**	**0.603**	**0.473**	0.738	0.541	0.518	0.422	0.668
		J48	**0.589**	0.583	**0.500**	0.731	0.564	**0.633**	0.480	**0.736**
		RF	0.559	0.581	**0.476**	**0.724**	**0.565**	**0.591**	0.460	0.722
*M*2	*EP*	NB	**0.542**	0.602	0.515	0.678	0.510	**0.733**	**0.557**	**0.684**
		BLR	**0.537**	0.654	0.521	0.687	0.502	**0.798**	**0.568**	**0.693**
		J48	**0.498**	0.620	0.486	0.657	0.477	**0.812**	**0.543**	0.651
		RF	**0.538**	0.502	0.471	0.673	0.512	**0.718**	**0.556**	**0.684**
*M*3	{*Delta*,*Churn*}	NB	0.510	0.448	0.333	0.649	**0.546**	**0.497**	**0.400**	**0.681**
		BLR	0.519	**0.508**	0.379	**0.664**	**0.520**	0.472	**0.392**	0.599
		J48	0.523	**0.484**	0.382	**0.638**	**0.529**	0.469	**0.391**	0.604
		RF	0.513	**0.488**	**0.369**	**0.646**	**0.537**	0.410	0.360	0.641
*M*4	*CM+EP*	NB	0.616	0.601	0.516	0.783	**0.635**	**0.616**	**0.543**	**0.813**
		BLR	0.546	0.600	**0.495**	**0.736**	**0.562**	**0.625**	0.483	0.714
		J48	**0.545**	**0.666**	**0.520**	0.717	0.517	0.660	0.471	**0.744**
		RF	0.577	0.616	0.499	**0.755**	**0.582**	**0.645**	**0.511**	0.750
*M*5	*CM+*{*Delta*,*Churn*}	NB	**0.606**	0.560	0.470	0.779	0.601	**0.576**	**0.487**	**0.789**
		BLR	**0.541**	0.515	0.418	**0.662**	0.515	**0.527**	**0.418**	0.646
		J48	**0.559**	0.533	0.443	**0.706**	0.555	**0.553**	**0.466**	0.706
		RF	**0.569**	0.587	0.462	**0.730**	0.557	**0.608**	**0.468**	0.713
*M*6	*EP+*{*Delta*,*Churn*}	NB	0.534	0.549	0.453	0.703	**0.557**	**0.623**	**0.514**	**0.757**
		BLR	**0.514**	0.648	0.501	0.661	0.507	**0.667**	**0.522**	**0.676**
		J48	**0.517**	0.674	**0.523**	0.659	0.464	**0.778**	0.518	**0.664**
		RF	0.514	0.581	0.464	0.687	**0.527**	**0.609**	**0.510**	**0.707**
*M*7	*CM+EP+*{*Delta*,*Churn*}	NB	0.611	0.572	0.493	0.777	**0.626**	**0.597**	**0.524**	**0.815**
		BLR	0.497	0.545	0.437	0.687	**0.532**	**0.623**	**0.486**	**0.705**
		J48	0.527	0.634	**0.494**	0.693	**0.546**	**0.640**	0.483	**0.746**
		RF	0.552	0.632	**0.507**	0.735	**0.578**	**0.650**	**0.507**	**0.767**

从表 7 中可以观察到:

(1) 对于代码度量元(*CM*)来说,时间邻近的演化历史对预测性能的影响不明显;

(2) 对于演化模式相关度量元,时间邻近的演化历史降低了 0.025 左右的精确度,但是提高了召回率、*F*-

值和 AUC 上的预测性能.其中,召回率最高提高了 0.216,*F*-值最高提高了 0.085,AUC 的提高不大;

(3) 对于代码变更度量元,时间邻近的演化历史提高了预测的精确度和 *F*-值,但是减低了召回率和 AUC;

(4) 对于代码和演化相关度量元的组合,时间邻近的演化历史对预测性能有改进作用,主要针对精确度和召回率;

(5) 对于代码和代码变更度量元的组合,时间邻近的演化历史可以提高预测的召回率和 *F*-值,并且在精确度和 AUC 上差别不大;

(6) 对于演化模式相关的度量元和代码变更度量元的组合,时间邻近的演化历史对除精确度外的预测性能有改进作用,且最大改进了 0.104 的召回率、0.061 的 *F*-值和 0.054 的 AUC;

(7) 对于 3 种度量元的组合,时间邻近的演化历史对精确度、召回率、*F*-值和 AUC 都有改进作用,且最高改进了 0.035 的精确度、0.078 的召回率、0.049 的 *F*-值和 0.053 的 AUC.

综上所述,案例研究表明:除代码度量元外,最近版本数据建立的模型比全部历史数据建立的模型具有更好的预测性能.这说明在软件进化历程中,最近的演化轨迹对产品的影响更大.这也符合自然界进化的规律.而代码类度量元表征产品的本质特征,随时间的影响不大.

5 结束语

本文提出了一种基于软件演化历史的缺陷预测方法,该方法的目的是预测一个 Java 类是否存在缺陷.本文把软件的演化过程看成一个物种的进化过程,通过改进的软件演化矩阵来模拟 Java 类的演化过程.根据一个 Java 类是否存在演化历史,把 Java 类分为新增类和已有类,其中,已有类有自身演化数据而新增类没有.本文主要是针对已有类进行缺陷预测.已有类是指在历史版本中曾存在过的 Java 类,它们有着自身的演化历史数据.针对该类的特点,本文提出两类代码演化度量元来度量已有类的演化,它们分别从粗粒度和细粒度上分析软件演化过程中版本级别和代码级别上软件属性的变化.案例研究选择了 6 个著名开源软件项目和 4 种常用的软件缺陷预测建模方法建立缺陷预测模型,实验结果验证了软件演化度量元的有效性.

在未来的研究中,将进一步开展以下几个方面的工作.

(1) 扩展数据源.本文选用的数据集都是 PROMISE 数据库中版本数大于 4 的 Java 项目,具有一定的局限性.在更多不同类型的连续项目上验证演化度量元的有效性,是下一步工作的重点;

(2) 完善演化度量元集.本文中使用两类与演化相关的度量元:一类是在版本级别上类的演化模式,另一类是软件代码变更相关的度量元.未来计划通过深入分析软件演化的特征来提出更多度量元,从而完善现有演化度量元集;

(3) 提取与新增类相关的度量元.本文中仅分析了软件演化对已有类的影响,下一步将深入分析由哪些因素影响着新增类的演化,从而提高新增类缺陷预测的性能;

(4) 应用在实际项目中.本文已经验证了演化度量元的有效性,下一步计划将实验结果应用于实际的项目中.用演化模式相关度量元来预测易出缺陷模块,帮助软件人员更好的检测和移除软件缺陷.

References:

[1] Halstead M. Elements of Software Science. New York: Elsevier North-Holland. 1977.

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

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

[4] 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 30th Int'l Conf. on Software Engineering. 2008. 181−190. [doi: 10.1145/1368088.1368114]

[5] Kpodjedo S, Galinier P, Antoniol G. A google-inspired error correcting graph matching algorithm. Technical Report, EPM-RT-2008-06, Ecole Polytechnique de Montreal, 2008.

[6] Kpodjedo S, Ricca F, Galinier P, Antoniol G. Not all classes are created equal: Toward a recommendation system for focusing testing. In: Proc. of the Int'l Workshop on Recommendation Systems for Software Engineering. New York, 2008. 6−10. [doi: 10.1145/1454247.1454250]

[7] Kpodjedo S, Ricca F. Evolution and search based metrics to improve defects prediction. In: Proc. of the 1st Int'l Symp. on Search Based Software Engineering. 2009. 23−32. [doi: 10.1109/SSBSE.2009.24]

[8] D'Ambros M. On the evolution of source code and software defects [Ph.D. Thesis]. Universita Della Svizzera Italiana of Washington, 2010.

[9] Wang D. Research on the analysis and prediction techniques of defect and its removal [Ph.D. Thesis]. Graduate University of Chinese Academy of Sciences, 2014 (in Chinese).

[10] Gall H, Jazayeri M, Klosch R, Trausmuth G. Software evolution observations based on product release history. In: Proc. of the Int'l Conf. on Software Engineering. 1997. 160−166.

[11] Gall H, Jazayeri M, Riva C. Visualizing software release histories: The use of color and third dimension. In: Proc. of the Int'l Conf. on Software Maintenance. IEEE Press, 1999. 99−108. [doi: 10.1109/ICSM.1999.792584]

[12] Lanza M. The evolution matrix: Recovering software evolution using software visualization techniques. In: Proc. of the 1st Workshop on Principles of Software Evolution. New York: ACM Press, 2001. 37−42. [doi: 10.1145/602461.602467]

[13] Wu J, Holt R, Hassan A. Exploring software evolution using spectrographs. In: Proc. of the 11th Working Conf. on Reverse Engineering. IEEE Press, 2004. 80−89. [doi: 10.1109/WCRE.2004.20]

[14] Wu JW, Spitzer CW, Hassan AE, Holt RC . Evolution spectrographs: Visualizing punctuated change in software evolution. In: Proc. of the 7th Int'l Workshop on Principles of Software Evolution. ACM Press, 2004. 57−66.

[15] Grba T, Ducasse S. Modeling history to analyze software ecolution. Journal on Software Maintenance and Evolution: Research and Practice, 2006,18(3):207−236. [doi: 10.1002/smr.325]

[16] Robbes R, Lanza M. A change-based approach to software evolution. Electronic Notes in Theoretical Computer Science, 2007,166: 93−109. [doi: 10.1016/j.entcs.2006.06.015]

[17] 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]

[18] Nagappan N, Ball T. Use of relative code churn measures to predict system defect density. In: Proc. of the 27th Int'l Conf. on Software Engineering. 2005. 284−292. [doi: 10.1109/ICSE.2005.1553571]

[19] Sliwerski J, Zimmermann T, Zeller A. When do changes induce fixes? ACM Sigsoft Software Engineering Notes, 2005,30(4):1−5. [doi: 10.1145/1082983.1083147]

[20] Bell RM, Ostrand TJ, Weyuker EJ. Looking for bugs in all the right places. In: Proc. of the Int'l Symp. on Software Testing and Analysis. New York: ACM Press, 2006. 61−72. [doi: 10.1145/1146238.1146246]

[21] Zimmermann T, Premraj R, Zeller A. Predicting defects for eclipse. In: Proc. of the 3rd Int'l Workshop on Predictor Models in Software Engineering. 2007. 9. [doi: 10.1109/PROMISE.2007.10]

[22] Aversano L, Cerulo L, Grosso CD. Learning from bug-introducing changes to prevent fault prone code. In: Proc. of the Int'l Workshop on Principles of Software Evolution. New York: ACM Press, 2007. 19−26. [doi: 10.1145/1294948.1294954]

[23] D'Ambros M. Supporting software evolution analysis with historical dependencies and defect information. In: Proc. of the Int'l Conf. on Software Maintenance. 2008. 412−415. [doi: 10.1109/ICSM.2008.4658092]

[24] Kim S, Whitehead 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]

[25] Abreu R, Premraj R. How developer communication frequency relates to bug introducing changes. In: Proc. of the Joint Int'l Workshop on Principles of Software Evolution. New York: ACM Press, 2009. 153−158. [doi: 10.1145/1595808.1595835]

[26] Eyolfson J, Tan L, Lam P. Do time of day and developer experience affect commit bugginess? In: Proc. of the 8th IEEE Working Conf. on Mining Software Repositories. New York: ACM Press, 2011. 153−162. [doi: 10.1145/1985441.1985464]

[27] Rahman F, Devanbu PT. Ownership, experience and defects: A fine-grained study of authorship. In: Proc. of the 33rd Int'l Conf.on Software Engineering. New York: ACM Press, 2011. 491−500. [doi: 10.1145/1985793.1985860]

[28] Shivaji S, Whitehead 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]

[29] 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]

[30] Jureczko M, Madeyski L. Towards identifying software project clusters with regard to defect prediction. In: Proc. of the 6th Int'l Conf. on Predictive Models in Software Engineering. New York: ACM Press, 2010. 9−19. [doi: 10.1145/1868328.1868342]

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

[32] Hall GA, Munson JC. Software evolution: Code delta and code churn. The Journal of Systems and Software, 2000,54:111−118. [doi: 10.1016/S0164-1212(00)00031-5]

[33] Menzies T, Caglayan B, He Z, Kocaguneli E, Krall J, Peters F, Turhan B. The PROMISE repository of empirical software engineering data. West Virginia University, Department of Computer Science, 2012. http://promisedata.googlecode.com

[34] Menzies T, Turhan B, Bener A, Gay G, Cukic B, Jiang Y. Implications of ceiling effects in defect predictors. In: Proc. of the 4th Int'l Workshop on Predictor Models in Software Engineering. New York: ACM Press, 2008. 47−54. [doi: 10.1145/1370788.1370801]

[35] He Z. Research on the data shift problem of software defect prediction and cost estimation [Ph.D. Thesis]. Graduate university of Chinese Academy of Sciences, 2014 (in Chinese with English abstract).

[36] Han J, Kamber M Wrote; Fan M, Meng XF, Trans. Data Mining: Concepts and Techniques. 2nd ed., Beijing: China Machine Press, 2007 (in Chinese).

[37] Quinlan JR. Introduction of decision trees. Machine Learning, 1986,1:81−106.

[38] Breiman L. Random forests. Machine Learning, 2001,45(1):5−32. [doi: 10.1023/A:1010933404324]

[39] 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]

[40] Menzies T, Greenwald J, Frank A. Data mining static code attributes to learn defect predictors. IEEE Trans. on Software Engineering, 2007,33(1):2−13. [doi: 10.1109/TSE.2007.256941]

[41] Turhan B, Bener A, Menzies T. Regularities in learning defect predictors. In: Proc. of the 11th Int'l Conf. on Product Focused Software Development and Process Improvement. 2010. 116−130. [doi: 10.1007/978-3-642-13792-1_11]

[42] Rahman F, Posnett D, Devanbu P. Recalling the "imprecision" of cross-project defect prediction. In: Proc. of the ACM SIGSOFT Symp. on Foundations of Software Engineering. 2012. 61. [doi: 10.1145/2393596.2393669]

[43] Rahman F, Devanbu P. How, and why, process metrics are better. In: Proc. of the 35th Int'l Conf. on Software Engineering. 2013. 432−441. [doi: 10.1109/ICSE.2013.6606589]

[44] Zimmermann T, Nagappan N, Gall H. Cross-Project defect prediction: A large scale experiment on data vs. domain vs. process. In: Proc. of the 7th Joint Meeting of the European Software Engineering Conf. and the ACM SIGSOFT Symp. on the Foundations of Software Engineering. 2009. 91−100. [doi: 10.1145/1595696.1595713]

[45] Ma Y, Luo GC, Zeng X, Chen A. 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]

[46] Seliya N, Khoshgoftaar TM, Hulse JV. Predicting faults in high assurance software. In: Proc. of the 12th Symp. on High-Assurance Systems Engineering. 2010. 26−34. [doi: 10.1109/HASE.2010.29]

[47] Rodriguez D, Ruiz R, Cuadrado-Gallego J, Aguilar-Ruiz J. Detecting fault modules applying feature selection to classifiers. In: Proc. of the IEEE Conf. on Information Reuse and Integration. 2007. 667−672. [doi: 10.1109/IRI.2007.4296696]

[48] Ekanayake J, Tappolet J, Gall H, Bernstein A. Tracking concept drift of software projects using defect prediction quality. In: Proc. of the 6th IEEE Working Conf. on Mining Software Repositories. Washington: IEEE Computer Society, 2009. 51−60. [doi: 10.1109/MSR.2009.5069480]

[49] Jiang Y, Cukic B, Ma Y. Techniques for evaluating fault prediction models. Journal of Empirical Software Engineering, 2008,13(5): 561−595. [doi: 10.1007/s10664-008-9079-3]

[50] Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P, Witten IH. The WEKA: A data mining software: An update. SIGKDD Explorations, 2009,11(1):10−18. [doi: 10.1145/1656274.1656278]

[51] Rodriguez D, Ruiz R, Cuadrado-Gallego J, Aguilarruiz J, Garre M. Attribute selection in software engineering datasets for detecting fault modules. In: Proc. of the 33rd EUROMICRO Conf. on Software Engineering and Advanced Applications. 2007. 418−423. [doi: 10.1109/EUROMICRO.2007.20]

[52] Hall M. A correlation-based feature selection for machine learning [Ph.D. Thesis]. Hamilton: The University of Waikato, 1999.

附中文参考文献:

[9] 王丹丹,软件缺陷产生和移除的因素分析及预测技术研究[博士学位论文].北京:中国科学院研究生院,2014.

[29] 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,2012,25(11):2499−2517. http://www.jos.org.cn/1000-9825/4559.htm [doi: 10.13328/j.cnki.jos.004559]

[35] 何治民.软件缺陷预测和成本估算中数据漂移问题的研究[博士学位论文].北京:中国科学院研究生院,2014.

[36] Han J, Kamber M,著;范明,孟小峰,译.数据挖掘:概念与技术.第 2 版,北京:机械工业出版社,2007.

王丹丹(1985－),女,山东枣庄人,博士,助理研究员,主要研究领域为缺陷数据分析与预测.

王青(1964－),女,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为软件过程方法与技术,经验软件工程.