面向缺陷分析的软件库挖掘方法综述*)

刘英博[1,2] 王建民[2]

(清华大学计算机科学与技术系 北京 100084)[1] (清华大学软件学院 北京 100084)[2]

摘 要 缺陷分析是软件工程领域内一个重要的课题,软件开发过程中的历史信息(缺陷记录、各个版本的源代码等)为缺陷分析这一课题提供了很有价值的经验数据。如何有效地利用这些数据进行缺陷分析,是软件库挖掘研究所面临的挑战。本文从统计方法和程序分析方法两个主要方面介绍了软件开发的历史信息是如何被用来进行缺陷分析的。

关键词 软件库挖掘,软件缺陷分析,软件工程

Review of Defect Analysis-oriented Software Repositories Mining Methods

LIU Ying-Bo[1,2] WANG Jian-Min[2]

(Department of Computer Science and Technology, Tsinghua University, Beijing 100084)[1]

(School of Software, Tsinghua University, Beijing 100084)[2]

Abstract Defect analysis is an important topic in the field of software engineering. The history information of software development (like defect records, different version of source code etc.) have provided valuable empirical data for this topic. How to utilize this data to better analysis the defects of software is a challenge faced by the research of Mining Software Repositories. This review introduces two approaches of using development history information to analysis software defects. The first approach is statistical defect analysis and the other approach is program analysis based defect analysis.

Keywords Mining software repositories, Software defect analysis, Software engineering

1 引言

软件开发尤其是大型软件开发一直是一件非常复杂而艰巨的任务。为了有效地管理软件从需求到废弃的整个生命周期中各个重要环节,软件开发团队使用了大量辅助系统,以便有效记录开发过程中的重要活动和这些活动产生的数据。典型的系统包括:软件配置管理系统(如 CVS[1], Subversion[2])、软件的缺陷管理系统(如 BugZilla[3])、软件团队的开发人员之间新闻(Newsgroups)组以及邮件列表(Mailing List)等等,这些系统综合在一起,形成了开发项目的软件库[1)](Software Repositories)。随着时间的推移和软件项目规模的不断扩大,在这些系统内部逐渐积累了大量和软件开发相关的历史信息。对这些信息进行分析、整理,将为软件设计、复用、维护以及对软件开发的新理念的检验等各个方面提供重要的经验数据,其潜在的价值已经逐渐为研究人员和实践者认识。软件库挖掘(Mining Software Repositories)就是以这些数据为目标进行分析的研究,主要内容包括:

- 从软件库中提取高质量数据的各种方法以及用来评价数据质量的指导原则;
- 对数据交换的格式、元模型以及框架性的工具的提议,以便数据共享、数据重用、可研究成果可重复;
- 对软件变更模式挖掘和分析;
- 用来评价与模拟缺陷发生情况和软件稳定性的模型;
- 对大型软件开发项目中开发过程和开发人员之间协作关系的理解;
- 用来辅助开发人员进行组件复用的搜索策略;
- 对长时间开发的大型软件项目的案例分析;
- 可以在社区内共享的评价指标。

与其他软件库挖掘方面的综述[4,5]不同,本文着眼于缺陷分析,从统计方法和程序分析方法两个方面介绍软件库挖掘方法在缺陷分析方面的应用,并且展望了软件库挖掘在缺陷分析方面的发展方向。对软件库挖掘的其他方面感兴趣的读者,可以参阅这方面的网站[6]和相关的综述[7,8]。

在进一步介绍之前,还需要指出一点:对于缺陷这个概念的定义,不同的研究有着不同的理解。单从缺陷的表达上,就存在着很多方式,例如缺陷(Defect)、故障(fault)、失效(failure)、错误(error)、问题(problem),还有大家经常提到的Bug。虽然有的研究人员已经意识到这个问题,并且在一开始就明确地指定其研究的范围,不同的缺陷概念还是在不同的研究工作中被重叠着使用,很难明确地将它们区分开来[9]。例如,有针对开发过程出现在源码版本库中的缺陷[10~12];有针对使用过程中发现缺陷的[13~16],等等。在本文的讨论中,并不给缺陷这一概念下一个统一的定义,而只是根据研究的内容,尽可能地把不同的研究中表示的缺陷概念区分开来,以免产生歧义。

*)973 计划资助项目(2002CB312006)、国家自然科学基金资助项目(60373011)。**刘英博** 博士研究生,主要从事企业建模、工作流、软件仓库挖掘等技术的研究;**王建民** 教授、博导、CCF 会员,主要研究方向:组织信息系统、数据库与工作流技术、软件度量与测试、数据网格与数据。

1) 目前对软件库的概念并没有一个明确的界定。严格地说,软件库应该包含那些能够记录软件开发历史信息的所有系统,这些历史信息并不局限于开发这一个方面,例如源代码版本、缺陷等。只是因为其他方面的历史信息目前很难收集到,所以未予以考虑。

2 统计缺陷分析方法

统计缺陷分析利用软件开发过程中遗留下来的历史数据，从统计的角度对系统进行预测或者评价。目前的统计缺陷分析的研究中，主要以预测性的缺陷分析为主，也有研究着眼于描述性缺陷分析，其应用主要有：

- 预测软件可能存在缺陷数量；
- 预测软件残留缺陷的数量；
- 预测模块内包含的缺陷数目；
- 预测文件中包含的缺陷数目；
- 预测软件的缺陷密度(缺陷数/代码量等)；
- 评价各种度量是否和缺陷之间存在关联关系；
- 评价发布时软件是否满足缺陷控制的要求。

在分析手段上，统计缺陷分析主要使用回归分析的方法进行缺陷预测。回归分析是通过多个输入变量预测输出变量最常用的统计方法，在缺陷预测方面使用回归分析的例子非常多[14,15,17,18]。在实际操作过程中，研究人员发现，作为输入的变量之间往往存在着相关性[9]，同时不同软件开发项目，由于采用的开发方法、管理的手段和面向的领域不同，缺陷数量和属性度量之间的关系也是不同的，选择不同的度量来预测缺陷效果差异很大[13]，所以在回归模型确定以后，需要采用相关分析的方法[19]或者主成分分析的方法[18,13,16]找到和项目特征匹配的度量。在预测模型方面，除了经典的回归分析以外，也有使用贝叶斯方法[9]、数字神经网络[20]进行分析的。

2.1 属性选择

属性选择是运用回归分析之前的一个重要步骤。在早期的研究中，Ohlsson 和 Aberg 就提出应该适当地选择度量的方式来进行缺陷的预测，他们希望通过研究找到哪些因素会影响各个模块中缺陷的数目，进而通过这些因素确定那些容易出错的模块。他们运用 Alberg 图的方法评价不同的度量作为预测模型输入的准确性[15]。通过在一个爱立信的程控交换机项目上的试验，他们发现能够通过统计模型建立缺陷数量和设计数据之间的关系，在他们调查的项目中，还发现20%的模块包含了60%的缺陷。但是，他们的研究并没有使用历史数据，而主要是针对爱立信交换机产品，所以不足以做出一般性的结论。

在文[13]中，微软研究院的 Nagappan 等人提出了如下四个假设：

H1：软件实体内的复杂度增加，会与发布以后的产品缺陷数目存在联系；

H2：存在一套通用的复杂度度量，用来检验 H1 对应的假设；

H3：存在一种组合的度量，来预测项目中的新实体[2)]对发布以后的缺陷数目的影响；

H4：从 H3 中获取的度量可以用来预测其他项目中存在缺陷的实体。

从这四个假设出发，他们对微软的五类产品：IE6 的 HTML 绘制模块、IIS W3 Server core 的应用加载模块、Process Messaging Component、DirectX 和 NetMeeting 的缺陷数据和历史版本的代码复杂度之间进行了关联分析，发现：在这五类产品中，复杂度的度量与产品发布以后的缺陷存在关联关系，但是没有一套通用的复杂度度量能够适应所有的这五个产品。然后，他们又使用主成分分析法对每一个项目中的各种度量进行了相关性分析，以便为每一个项目确定一套组合的复杂性度量，并用这一套复杂性度量构造回归模型，来预测缺陷。发现，用主成分分析获得的复杂性度量集构造的回归模型能够有效地预测未来的缺陷。最后，他们分别用每一个项目里建立的缺陷预测回归模型对其他的项目进行了分析，发现不同项目的预测模型之间很难通用。作为结论，他们提出：在使用代码复杂度构造回归模型预测产品缺陷时，不应该盲目地采用复杂度度量构造的回归模型来预测产品的缺陷，应该先根据项目开发的历史数据对各种复杂度度量的组合进行验证，然后确定一套对项目特点有意义的复杂度度量，再构造一套回归模型来预测缺陷。

这些工作都说明属性的选择对于缺陷分析而言是非常重要的。下面我们来看看如何使用回归分析的方法来进行缺陷预测。

2.2 回归缺陷分析

为了说明回归分析预测缺陷的基本原理，首先看看经典的线性回归模型在缺陷预测过程中的应用，然后讨论线性回归模型的不足以及人们提出的两种主要改进模型：广义线性模型和负二项回归模型。最后介绍其他统计理论在缺陷预测方面的应用。

普通的线性回归模型假设：通过若干次观察，某个随机变量的观察值会随着不同的输入向量 x 变化，这样就可以得到由多次观察值组成的向量 y 和输入向量组成的矩阵 X。如果每次观察得到的结果是独立的并且都服从标准差为 σ 的正态分布，那么 y 的期望 $\mu=E(y)$ 满足等式：$\mu=X\beta$。如果通过参数估计得到系数 β，就可以得到下一个输入变量对应的输出估计。

对应到缺陷预测的应用，y 表示对目标(可能是类、文件、模块或者是整个软件)多次观察到的缺陷数目，X 表示每次观察目标所具备的属性(例如代码数、复杂度、开发语言的代号、版本号等等)，而系数 β 则是参数估计的目标。

线性回归模型的问题在于它对输出变量的正态独立同分布假设在实际应用中大都不成立[9]，所以人们提出采用广义线性模型(Generalized Linear Models)来预测缺陷[14]。广义线性模型可以描述输出变量服从其他分布的情况，并且放宽了对每次观察要求标准差不变的限制。和线性回归模型不同，广义线性模型假设观测到的输出变量服从某种指数分布，(例如泊松分布、二项分布、gamma 分布等)。在线性回归模型中 y 的均值 $\mu=E(y)$；在广义线性模型中 y 的均值由非线性的连接函数(Link Function)表示，即 $E(y)=g(\mu)=X\beta$。通过选择分布的假设以及连接函数，可以改进模型对缺陷预测的准确程度。

在文[14]中，Grave 等人使用泊松分布假设，并且取连接函数 $g(x)=\log_e x$ 来预测一个程控交换机项目缺陷。在预测的过程中，他们使用了更改记录库和版本库的信息，其分析的故障主要来源于集成和测试阶段。在使用历史信息的方法上，他们将版本库的信息转换为代码的寿命进行度量，并且发现用这个信息作为度量要比直接使用代码行数预测更能反映缺陷数量。在进一步的实验中，他们加入了更改记录信息并且结合时间加权，结果发现这种加权时间度量更准确地反映了缺陷的发展趋势。

2) 本文中新实体不但代表新的模块，同样代表相同模块中的新版本。

Ostrand 等人在对大型软件系统的缺陷分布进行了分析以后，提出采用负二项回归模型(Negative Binomial Regression Model)来预测下一个版本内各个文件中可能包含的缺陷数量[18]，其基本思想如下：

负二项回归模型假设输出值 y 代表缺陷的数目，当输入向量 x 确定以后，y 服从均值为 λ 泊松分布。对于任意的文件 i，将其属性视为输入向量 x_i，其包含的缺陷数目视为输出值 y_i，那么 λ_i 代表了 y_i 确定以后的条件的均值：$\lambda_i = \gamma_i e^{\beta' x_i}$

其中 γ_i 本身是一个均值为 1、服从 gamma 分布的随机变量，其方差 σ^2 是需要估计的参数。σ^2 越大，那么估计的错误准确性就越低。然而，随着 x_i 对 y_i 解释能力的增加，σ^2 会逐渐减小。所以，为了能够确定 λ_i，负二项回归模型需要估计系数 β 和方差 σ^2。

在输入向量 x_i 的选择上，Ostrand 等人根据缺陷在系统中分布的情况[18]采用了代码行、文件是否是新建的、文件是否改变、文件的年龄(即在哪些版本中出现)、上一个版本里面发现错误的数目、编程语言和版本作为输入变量，然后用极大似然估计的方法对一个历时 4 年、具有 17 个版本的库存系统和一个历时 2 年、具有 9 个版本的配置预览系统的单元测试结果进行了预测，发现在这两个系统中 20%的文件分别包含了 71%和 90%的缺陷[18]。

在文[9]中，Fenton 等人深入细致地调查了以往的缺陷预测方面的研究，并且在总结了这些研究的基础上提出了一系列关于使用统计方法进行缺陷预测的问题。这些问题有的涉及到概念理解的模糊，有的涉及到统计模型本身的缺陷，也有的涉及到错误的研究方法。在此基础上，他们提出采用贝叶斯信念网(Bayesian Belief Networks)进行缺陷预测，并且列举了 BBN 应用的方法，但是没有涉及到过多细节。

在文[20]中，Frank Padberg 等人提出采用数字神经网络预测软件文档(而非代码)中的错误，他们希望能够通过学习文档在审核过程中发现的缺陷，预测软件的文档在经过审核以后残留的缺陷数目。他们首先使用关联分析确定两个文档的输入属性，然后使用贝叶斯学习的方法来确定神经网络的误差约束，最后他们采用数字神经网络比较了各种不同数目的隐藏单元的非线性回归模型的效果。虽然在他们的研究中用来对神经网络进行训练的样本只是 0-1 缺陷矩阵，但是他们最后指出，当存在一个缺陷跟踪系统的时候，这种模型能够通过学习缺陷在文档中发现的情况有效地估计文档残留的缺陷数目。

最后，在利用历史信息进行预测方面，Ahmed E. Hassan 等人提出了一种简单易行的开发过程中缺陷预测方法。虽然其方法和回归模型没有太大关系，但是由于这些研究的目的非常相似，所以也在这里一并介绍。Hassan 的方法借用了缓冲区的思想[21]，不同于前面介绍的缺陷数量的预测方法。他们的结果表现为一个模块的优先列表，这个列表中包含了最有可能出现缺陷的模块。而这个列表中应该存放哪些模块，则由一个启发性的规则确定。他们提出了四个启发性规则：最频繁修改的模块(MFM)、最近修改的模块(MRM)、最频繁修复的模块(MFF)和最近修复的模块(MRF)。在试验中，他们利用这四个启发性规则在 Postgre DBMS、KDE K Desktop Environment、KOffice Office Productivity Suite、FreeBSD 操作系统、OpenBSD 操作系统、NetBSD 操作系统六个开源项目上进行了测试。通过比较命中率，他们发现：在四个启发性规则中，MFM 和 MFF 的命中率最好；当取缓冲区大小相当于 20%～30%的总模块时，获得的命中率和计算付出的代价之比最高。

3 程序缺陷分析方法

统计缺陷分析着眼于宏观缺陷估计和描述，而程序缺陷分析则着眼于从微观的角度理解缺陷的本质，然后根据缺陷特征发掘缺陷，这些方法大都体现了研究人员对开发过程和程序本身的深刻理解。在各种程序缺陷分析研究中，有的单独采用版本库进行缺陷分析，有的采用版本库结合进行缺陷库进行分析，也有单独采用缺陷库进行分析。本节从无缺陷信息的程序分析和有缺陷信息的程序分析介绍软件库挖掘在缺陷分析方面的应用。

3.1 无缺陷信息的程序缺陷分析方法

无缺陷信息的程序分析方法仅仅采用版本和源代码进行分析研究。这样的研究从源代码版本和源代码本身的变化出发，不考虑缺陷库。通过对程序的理解，自动发现导致缺陷的错误。使用这种方法进行的研究，研究人员一般对于缺陷的性质和发生频繁程度都有一个非常清醒的认识，然后他们在针对某一类缺陷在程序中进行查找，这样的分析可以非常准确地定位缺陷，但是适用面比较窄。如果针对的缺陷在软件里面不是频繁出现，那么分析所付出的代价将会大于其收益，使效果大打折扣。

在文[22]中，Williams 等人调查了 Apache 的 CVS 检入说明以后发现：在 Apache 的开发过程中存在大量由于忽略函数返回值检验(Return Value Check)引起的错误，所以他们利用源代码静态检查器(Static Checker)对 Apache 2.0 的 Httpd 模块进行了分析，并且在分析的过程中利用 CVS 的更改记录提高监测出错误的准确率。其后他们又对长达 6 年 Wine(实现 MS Windows API 的开源项目)进行了试验，同样发现利用 CVS 的历史更改信息有助于提高静态分析的准确性[11]。

在进一步的研究中，Willanms 等人还对系统调用的顺序等规则进行了挖掘，从软件开发的历史角度判断哪些调用应该按照顺序进行，例如 BeginPaint 应该和 EndPaint 配合使用，HeapAlloc 和 HeapFree 应该一起调用等。通过这样结合源代码的静态分析，确定由于违反调用规则而导致的缺陷[11]。

软件开发过程中，对接口或者类中的方法进行更改往往是导致错误的重要原因[23]。在文[12]中，Weiβ gerber 等人对两类重构操作进行研究：方法更名(Rename Method)和增加参数(Add Parameter)。在面向对象的软件中进行这两类重构操作的时候，如果不注意继承关系将会导致问题。例如，修改接口里面方法或者抽象方法的名字会导致子类编译错误，如果修改父类中的方法将会导致子类同名的方法失去重载关系从而改变系统运行的行为。这种错误不会在编译时表现出来，所以更难发现。为了发现这种不完全的重构操作，作者用一个简单的解释器对每次更改进行分析，获取方法更名和增加参数这样的操作，然后再对更改以后的软件进行扫描，获得有关类和更改的继承关系，进而判断重构操作是否是不一致的。虽然其思路非常好，但是在 JEdit 和 Tomcat 上进行的试验里只分别发现了 5 个和 7 个不一致的重构操作，再加上其代价非常高昂(因为需要为每次更改建立起一个版本快照以便找到继承关系)，所以效果并不理想。

Zeller 在文[24]中提出 Delta Debugging 的概念。Zeller

对 Delta Debugging 的定义是：通过观察更改差异(Delta)，确定程序的异常行为。Zeller 的方法假设一次一次的更改能够逐步地应用到快照版本上，并且回归测试能够自动进行，通过二分法将携带错误的更改分组应用到快照上，然后通过缩小更改的分组大小逐步求精，最后确定到底是哪个更改导致了缺陷。Zeller 为这种分析思想设计了 DD^+ 算法，通过试验验证了该方法能够准确地找到缺陷对应的第一类关联。但是 Zeller 的方法资源消耗非常大，并且要求测试能够自动进行，所以实际应用的难度很大。

3.2 有缺陷信息的程序分析方法

除了配置管理系统中记录的版本以外，另一类缺陷分析的研究工作希望能够在缺陷库里面的缺陷和版本库里面的更改之间建立连接，然后利用这种连接分析缺陷和更改之间的关系。对于一个缺陷而言，和它有关的更改可以分成两类。

第一类：导致了缺陷的更改；

第二类：修复了缺陷的更改。

由于历史的原因，缺陷跟踪系统(如：BugZilla)和配置管理系统(如 CVS)之间没有提供有效的机制来建立更改和缺陷之间的联系。所以，建立缺陷和更改之间的联系需要很多数据预处理工作。Mokus 等人在调查 Apache 和 Mozilla 的检入说明以后发现：利用配置管理系统中的检入说明可以对软件的更改操作进行一个有效的分类，其中包含那些对修复缺陷起作用的更改[25]。从这个发现出发，其他的研究人员采用了文本分析的方法建立缺陷和更改之间的联系。在这方面比较有代表性的工作是 Fischer，Pinzger 和 Gall 等人的研究，他们的方法能够在更改库和缺陷库的记录之间建立起第二类关联[26,27]。Jacek，Zimmermann 和 Zeller 又对他们的方法进行了改进，进而在第二类关联的基础上推测出第一类关联[28]。他们通过对 CVS 库内开发人员遗留下来的检入说明按照一定的规则(例如：正则表达式等)进行文本分析，获取有关缺陷的描述，例如缺陷描述的关键字、缺陷的代号和缺陷的状态等。根据这些代号信息在 BugZilla 库内寻找匹配的缺陷。如果存在代号对应的缺陷，那么就可以建立起缺陷和更改之间的第二类关联。然后，在第二类关联的基础上，找到最近一次更改的具体内容(这在软件配置管理系统内能够很容易实现；例如在 CVS 里，只需要用 diff 命令即可得到更改的集合)，这些内容代表了对修复缺陷起作用的更改，以这些更改的内容为条件从以往的更改记录中查找哪些更改引起了这些内容(这种查找也能够通过配置管理系统实现，如用 CVS 里面的 annotate 命令)，即可建立第一类关联关系。

需要注意的是，由于开发人员习惯(开发人员在修复缺陷时可能不会留下日志)、文本分析本身的不稳定性(文本描述中存在错误的内容从而连接到错误的缺陷上)、更改目的的不确定性(一批更改可能并不仅仅是为了修复缺陷)等问题，使用这些方法建立起来的任何一类关系都不可能是完全的。所以，在建立起关联的同时还要使用一些其他的措施过滤掉不能确定正确性的关联，以保证分析用来进行分析的关联的质量。

即使存在这样的问题，通过上述的关联关系，研究人员仍然发现了很多有趣的现象。例如，在 MOZILLA 项目中，修复一个缺陷又导致另外的缺陷的概率是直接由于更改导致缺陷的概率达 3 倍以上，而在 Eclipse 项目里则小得多。在 MOZILLA 项目里，程序员在星期五引起缺陷的概率最大；而在 Eclipse，则在星期六引起缺陷的概率比较大[28]。

另外，也有采用其他方式识别更改和缺陷之间关系的。例如，采用批量更改的大小来判断更改是否是修复缺陷[18]、利用高级缺陷跟踪系统建立更改和缺陷之间的联系[13,29,17]，这里就不一一列举了。

最后，有少数研究仅仅针对缺陷记录本身进行分析。例如，某个缺陷的解决依赖于另外的缺陷(Depend)，两个报告的缺陷其原因可能是相同的(Duplicate)，在解决一个缺陷的同时可以引用解决另外缺陷的方法(Cite)。Sandusky 等人研究了缺陷之间的关联关系，并且提出采用缺陷报告网络(BRN)的方式来表达这种关联关系。他们希望通过 BRN 能够有效地管理和组织缺陷库[30]。

总结与展望 缺陷分析的目的是避免缺陷。从目前在软件库挖掘领域内的缺陷分析看，主要有两个分支：统计分析和程序分析。图 1 展示了对这些手段的一个主要分类。统计缺陷分析的研究是在理想状态假设下使用统计进行研究的方法，大量研究使用回归模型来预测缺陷。这种预测是否准确，很大程度上取决于数据的质量和属性的选择。在一个实际的软件开发项目中，如果希望统计缺陷分析能够收到良好的效果，就需要用良好的工程管理手段有系统地记录缺陷，并且在建立对缺陷进行描述的统计模型之前，认真收集和整理数据，并且采取措施，选择能够反映项目自身特征的度量集合。

统计缺陷分析也存在着不足。首先，统计缺陷分析方法给出了宏观的缺陷分布情况，但是没有给出缺陷产生的原因。到底软件开发过程中的哪个环节容易导致缺陷，仍需要项目参与人员根据统计数据，结合项目特点以及自身经验予以确定。其次，从统计缺陷分析本身而言，各种回归方法着眼于给历史数据一个很好的解释，但是这个解释能否适应未来的情况，估计谁也无法给出满意的结论。

程序缺陷分析的研究则比较现实，它着眼于能够获得的数据(版本库、缺陷库、源代码等等)，在认识缺陷以后进行分析。程序缺陷分析能否成功地应用，很大程度上取决于研究人员对缺陷的理解。程序缺陷分析面临的另一个问题在于计算开销，几乎所有的方法都要进行大量的前处理工作，例如建立快照、分析文本和程序等等。但是，在阅读文献的过程中，我们也看到，有的研究艺术性地提出了简单实用的方法，在现有工具和人员之间找到了有机的平衡。例如，Hassan 等人提出的缓冲区缺陷预测方法等，这些方法最有潜力为工业界接受并且产生影响。

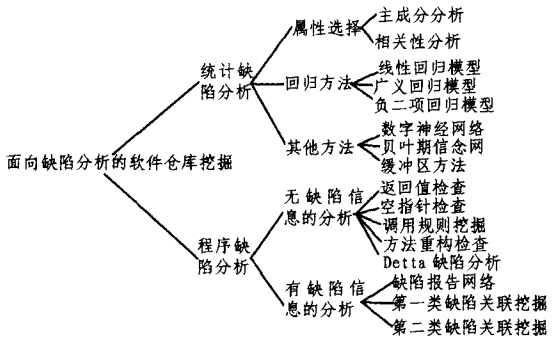

图 1 各种方法的分类情况

对于整个基于历史数据的缺陷分析而言，共同面临的问题还是在于数据质量的问题[31]。例如，很多统计缺陷分析只能忽略缺陷在开发过程中的差异来进行预测。而实际上，开发过程中不同阶段产生的缺陷在分布上是完全不同的，例如

(下转第 11 页)

05-05
4 Kavantzas N, Burdett D, Rizinger G, et al. Web Service Choreography Description Language Version 1.0. http://www.w3.org/TR/ws-cdl-10/, 2005-11
5 Brogi A, Canal C, Pimentel E, et al. Formalizing Web Service Choreographies. Electronic Notes in Theoretical Computer Science, 2004, 105: 73～94
6 Arkin A, Askary S, Fordin S, et al. Web Service Choreography Interface (WSCI) 1.0. http://www.w3.org/TR/wsci/, 2002-08
7 Foster H, Uchitel S, Magee J, et al. Model-Based analysis of Obligations in Web Service Choreography. In: Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006), USA, 2006. 149～157
8 Yeung W L, Wang Ji, Dong Wei. Verifying Choreographic Descriptions of Web Services Based on CSP. In: Proceedings of the IEEE Services Computing Workshops (SCW'06), USA, 2006. 97～104
9 Busi N, Gorrieri R, Guidi C, et al. Towards a formal framework for Choreography. In: Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2005), USA, 2005. 107～112
10 Gorrieri R, Guidi C, Lucci R. Reasoning about interaction patterns in Choreography. In: Proceedings of 2nd International Workshop on Web Services and Formal Methods (WS-FM '05), France, 2005. 333～348
11 Mendling J, Hafner M. From WS-CDL Choreography to BPEL Process Orchestration. wi.wu-wien.ac.at/home/mendling/publications/TR06-CDL.pdf
12 Milner R. Communicating and Mobile Systems: The Pi Calculus. Cambridge: Cambridge University Press, 1999
13 Victor B, Moller F. The Mobility Workbench-A Tool for the Pi-Calculus. In: Proceedings of the 6th International Conference on Computer Aided Verification, USA, 1994. 428～440
14 Sangiorgi D. A Theory of Bisimulation for the pi-Calculus. Acta Informatica, 1996, 3(1): 69～97
15 Salaün G, Bordeaux L, Schaerf M. Describing and Reasoning on Web Services Using Process Algebra. In: Proceedings of the IEEE International Conference on Web Services (ICWS'04), USA, 2004. 43～50

(上接第4页)

在开发阶段提交的大量缺陷是编码的错误导致(返回值检查、空指针引用等)的,在发布以后提交的缺陷则大多是由于功能设计导致的缺陷或者逻辑错误[10]。程序缺陷分析的研究也面临同样的问题,最典型的就是缺陷和更改之间没有关联。虽然通过文本分析这样的启发式方法能够建立缺陷和更改之间的关联,但是如果开发项目的管理没有保障,开发人员仅仅根据自己的习惯添加说明,那么这种方法的准确性和完备性就无法得到保证。

参 考 文 献

1 Ximbiot. Version Management with CVS. http://ximbiot.com/cvs/manual/.
2 Collins-Sussman B, Fitzpatrick B W, Pilato C M. Version Control with Subversion. 1 ed. http://svnbook.red-bean.com/en/1.1/svn-book.html, 2005
3 BugZilla. Homepage. http://www.bugzilla.org/
4 Godfrey M, Dong X, Kasper C, et al. Four Interesting Ways in Which History Can Teach Us About Software. In: 1st International Workshop on Mining Software Repositories(MSR-04), 2004
5 Kagdi H, Collard M, Maletic J. Towards a Taxonomy of Approaches for Mining of Source Code Repositories. In: 2nd International Workshop on Mining Software Repositories (MSR 2005), Saint Louis, Missouri, USA, 2005
6 MSR. Mining Software Repositories Homepage. http://msr.uwaterloo.ca
7 Hassan A E, Holt R C, Mockus A. Report on MSR 2004. In: International Workshop on Mining Software Repositories, 2004
8 Hassan A E, Holt R C, Diehl S. Report on MSR 2005. In: International Workshop on Mining Software Repositories, 2005
9 Fenton N E, Neil M. A critique of software defect prediction models. Software Engineering, IEEE Transactions on, 1999, 25: 675～689
10 Williams C C, Hollingsworth J K. Bug Driven Bug Finders. In: 1st International Workshop on Mining Software Repositories (MSR 2004), Edinburgh, Scotland, 2004
11 Williams C C, Hollingsworth J K. Automatic mining of source code repositories to improve bug finding techniques. Software Engineering, IEEE Transactions on, 2005, 31: 466～480
12 Görg C. Weißgerber P. Error Detection by Refactoring Reconstruction. In: 2nd International Workshop on Mining Software Repositories, St Louis, Missouri, 2005
13 Nagappan N, Ball T, Zeller A. Mining Metrics to Predict Component Failures. In: 3rd International Workshop on Mining Software Repositories (MSR 2006), Shanghai China, 2006
14 Atkins D L, Ball T, Graves T L, et al. Using version control data to evaluate the impact of software tools: a case study of the Version Editor. Software Engineering, IEEE Transactions on, 2002, 28: 625～637
15 Ohlsson N, Alberg H. Predicting fault-prone software modules in telephone switches. Software Engineering, IEEE Transactions on, 1996, 22: 886～894
16 Khoshgoftaar T M, Allen E B, Halstead R, et al. Detection of fault-prone software modules during a spiral life cycle. In: Proceedings of the 1996 IEEE Conference on Software Maintenance, ICSM, Monterey, CA, USA, Nov. 1996
17 Gefen D, Schneberger S L. The non-homogeneous maintenance periods: a case study of software modifications. In: Software Maintenance 1996, Proceedings., International Conference on, 1996
18 Ostrand T J, Weyuker E J, et al. Predicting the Location and Number of Faults in Large Software Systems. Software Engineering, IEEE Transactions on, 2005, 31: 340～355
19 Fenton N E, Pfleeger S L. Software Metrics: A Rigious & Practical Approach: PWS Publishing Company, a division of Thomson Learning, 1997
20 Padberg F, Ragg T, Schoknecht R. Using machine learning for estimating the defect content after an inspection. Software Engineering, IEEE Transactions on, 2004, 30: 17～28
21 Hassan A E, Holt R C. The Top Ten List: Dynamic Fault Prediction. In: 21st IEEE International Conference on Software Maintenance (ICSM 2005), 2005
22 Willianms C C, Hollingsworth J K. Bug Driven Bug Finders. In: 1st International Workshop on Mining Software Repositories (MSR 2004), Edinburgh, Scotland, 2004
23 Schwaber C, Orlov L M, Zetie C, et al. The Forrester Wave(tm): Process-Centric Software Configuration Management. Cambridge, MA 02139 USA: Forrester Research Inc, November 14, 2005
24 Zeller A. Yesterday, My Program Worked. Today, It Does Not. Why? 1999
25 Mockus A, Votta L G. Identifying Reasons for Software Changes using Historic Databases. In: International Conference on Software Maintenance, San Jose, California, 2000
26 Fischer M, Pinzger M, Gall H. Analyzing and relating bug report data for feature tracking. In: Reverse Engineering, 2003. WCRE 2003. Proceedings. 10th Working Conference on, 2003
27 Fischer M, Pinzger M, Gall H. Populating a Release History Database from version control and bug tracking systems. In: Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on, 2003
28 Sliwerski J, Zimmermann T, Zeller A. When Do Changes Induce Fixes? In: 2nd International Workshop on Mining Software Repositories (MSR 2005), St Louis, Missouri, 2005
29 Purushothaman R, Perry D E. Toward understanding the rhetoric of small source code changes. Software Engineering, IEEE Transactions on, 2005, 31: 511～526
30 Sandusky R J, Gasser L, Ripoche G. Bug Report Networks: Varieties, Strategies, and Impacts in a F/OOS Development Community. In: 1st International Workshop on Mining Software Repositories (MSR 2004), Edinburgh, Scotland, 2004
31 Zimmermann T, WeiBgerber P. Preprocessing is a Prerequisite for Any Analysis of CVS Data. In: 1st International Workshop on Mining Software Repositories (MSR), Edinburgh, UK, 2004