软件缺陷数据处理研究综述

李 宁 李战怀
(西北工业大学计算机学院 西安 710072)

摘 要 软件缺陷数据是软件质量分析和改进的重要基础数据之一。如何在分析缺陷数据前对缺陷数据进行有效的预处理,如何根据缺陷特征对缺陷数据进行合理分类,如何对缺陷数据进行挖掘以及统计分析,是软件缺陷研究领域面临的问题。详细介绍了缺陷数据预处理、缺陷分类以及缺陷数据挖掘分析3个方面的研究内容、方法和技术,并对这些方法进行了比较和分析,最后提出了几个软件缺陷数据处理研究领域需要进一步研究的问题。

关键词 软件缺陷,重复缺陷,关联缺陷,缺陷分类,文本挖掘,统计分析
中图法分类号 TP311 **文献标识码** A

Overview of Software Defect Data Processing Research

LI Ning LI Zhan-huai
(Computer College, NorthWestern Polytechnical University, Xi'an 710072, China)

Abstract Software defect is important basic data for software quality analysis and improvement. The problems faced by software defect research is how to preprocess noisy defect data effectively before analysis, how to classify defect data according to their characters, and how to mine and analyze them. This paper described the contents, methods and technologies of the above mentioned three problems, then compared and analyzed them, at last proposed several problems of defect data which is worth further studying.

Keywords Software defect, Duplicate defect, Defect association, Defect classification, Text mining, Statistical analysis

关于软件缺陷(Defect)有很多相关的术语,如错误(Error)、缺陷(Defect)、故障(Fault)、失效(Failure)、Bug、问题(Problem)等。特别是有些学者[1,2]对错误、缺陷、故障、失效这4个概念加以严格的区分。本文中仅区分缺陷和失效,根据上下文不同使用到的其他相关术语均与缺陷同义。缺陷是指软件中存在的未满足与期望或者规定用途有关的要求,而失效是指软件运行时出现不正确的行为(输出)。软件缺陷与失效的关系在文献[3]中给出了简单的说明:引起软件失效的主要原因就是软件中存在的各种缺陷。因此,为了提高软件质量,必须对软件缺陷进行控制与分析。

不论是商业软件还是开源软件,软件规模的增长都使得其软件缺陷数据库日益庞大,因此关于软件缺陷的数据处理技术研究逐渐吸引了更多研究者的目光。软件缺陷数据处理主要包括:去除重复无效等噪音缺陷数据、对良好的缺陷数据进行合理分类、对缺陷内容进行挖掘与分析等。软件缺陷数据处理的研究,一方面可以对软件缺陷的管理、控制、预测起重要的指导作用,另一方面也为基于缺陷的软件测试奠定了重要的数据基础。近年来数据挖掘、人工智能、统计学等相关技术的发展,已极大地带动了该领域的研究。

1 软件缺陷数据预处理

软件开发中通常利用缺陷报告记录缺陷数据,有的通过缺陷管理系统(如Bugzilla[4,5]等)记录,有的通过邮件、文件等记录。虽然缺陷报告的形式千变万化,但其处理流程和报告的核心内容都基本相同[6]。软件缺陷报告基本都是用自然语言描述的文本信息,其内容主要包括发现人、发现时间、重要程度、优先级、缺陷概述、缺陷详细现象、重现步骤以及其他相关附件等。

以具有代表性的Bugzilla系统为例。一个缺陷被提交确认(Open)之后,其最终的处理意见(Resolution)可能有如下几种:已修改的(Fixed)、不是问题(Invalid)、无法修改(Wontfix)、以后版本解决(Later)、保留(Remind)、重复(Duplicate)、无法重现(Worksforme)。文献[5]对开源软件FireFox的软件缺陷数据进行了统计分析,分析数据包括2013条缺陷报告。各种缺陷比率如下:Open(33%),Fixed(11%),Duplicate(30%),Worksforme(13%),Invalid(11%),Wontfix(2%),Later/Remind(0%)。其中"不是问题"、"无法重现"和"重复"3类无效缺陷合计比率高达54%,这意味着有超过一半的缺陷报告都浪费了很多宝贵的开发时间且对开发没有意义。商业软件也不例外。笔者参与的一个大型商业软件的第三方测试中,报告的547条缺陷中最终被开发部门认同并修改的有265条,也仅占到48%。由此可知,如果对软件缺陷报告数据事先进行一定的预处理,去除掉重复、无效等缺陷报告,将会为软件开发节约大量宝贵的时间成本。

到稿日期:2008-09-02 返修日期:2008-12-04 本文受国家自然科学基金(60573096)资助。
李 宁(1978-),女,博士研究生,讲师,主要研究方向为软件测试、软件工程、数据库理论与技术等,E-mail:lining@nwpu.edu.cn;李战怀(1961-),男,教授,博士生导师,CCF高级会员,主要研究方向为软件工程、数据库理论与技术等。

1.1 重复软件缺陷报告的检测

重复缺陷通常有两种：1）现象本身完全重复；2）现象不同，但却由同一原因导致，修改一个缺陷后，另一个重复缺陷无须修改便自然消失了。第一种重复缺陷相对比较容易判断，相关研究也较多[7-11]，主流方法是基于向量空间模型（Vector Space Model，简称 VSM）的文本相似度计算。对于第二种重复缺陷，目前已有的研究中尚没有比较好的检测方法。

L. Hiew[7]指出，对于重复缺陷有两种处理方法：1）阻止报告重复缺陷，这要求在缺陷提交的时点进行检测；2）通过缺陷报告分拣（triage）将重复的缺陷报告标示出来。实际中目前广泛采用的是第二种方法。P. Runeson 等人[8]提出了一种用自然语言处理（Natural Language Processing，简称 NLP）检测重复缺陷的方法。文献[8]利用该方法对 Sony Ericsson Mobile Communications 软件缺陷报告进行了案例验证，结果表明约有 2/3 缺陷报告被正确地检测到。

P. Runeson 等人所用的自然语言处理技术是信息检索（information retrieval）技术中的核心技术，主要处理如下。

1）分词（tokenization）：分析每个句子中的单词，将其分离。主要通过空格来分词，其中关于标点的分词处理是比较容易出问题的环节。

2）分析单词的原形（stemming）：恢复相关单词的原形单词，这样可以消除单词变形带来的差异。

3）去除停用词（stop words removing）：自然语言中有很多没有实际含义的共通词，例如 the，this，when 等，所以通过一个 stop words list 将这些词去除。除了通用的 stop words list 之外，还添加了软件缺陷报告中常出现的停用词，例如 attach，log 等。

4）建立向量空间模型：根据每个缺陷报告中的分离单词建立 VSM。每个缺陷报告是一个文档，每个单词是文档中的一维，每个单词出现的频率是该点的各轴坐标。如图 1 所示，两个文本 textA 和 textB 经过前 3 步的处理之后，分别仅含有 computer 和 science 两个单词。其出现频率见图中标识。建立了如图 1 所示的 VSM。

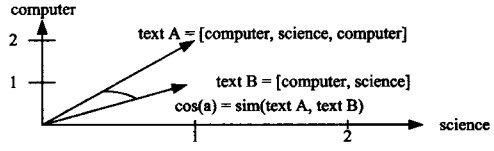

图 1 余弦法计算相似度示意图

5）文本相似度计算：在 VSM 中 3 种主要的相似性计算方法分别是 Cosine，Dice，Jaccard，其中应用最普遍的是 Cosine（余弦法）。如图 1 所示，textA 和 textB 在此二维空间中的连线夹角的余弦值就表示相似度值。相似度是一个[0，1]之间的数字，0 表示完全不同，越接近 1 表示相似度越高，1 则表示完全相同。

6）分别计算出一个新的缺陷报告与已有缺陷报告的相似度，然后将相似度最高的 n 个缺陷列出，由分拣者进一步人工判断是否重复。n 也称为 top list size，可以自由调整。当 n 越大时，找到真正重复缺陷的概率就越大。

由于 P. Runeson 等人提出的方法对重复缺陷的检出率还较低，X. Y. Wang 等人[9]在其研究基础之上，提出了一种将自然语言信息与执行信息结合起来检测重复缺陷的新方法。通过对 Firefox 缺陷库的验证对比表明，该方法可以检测到 67%～93%的重复缺陷，而仅使用自然语言信息只能检测到 43%～72%的重复缺陷，检索正确率大幅度提高。

X. Y. Wang 等人提出的方法中，自然语言信息指缺陷报告本身记录的各种文本等信息，执行信息则是指根据缺陷报告的重现步骤重现缺陷时程序自动记录的执行日志等信息。

同一缺陷报告的自然语言可能不同，但若执行日志相同，也可认为两者之间存在重复的可能性。该方法的主要步骤如下。

1）计算自然语言信息相似度（Natural Language based Similarities，简称 NL-S）：与 P. Runeson 等人提出的方法相同，利用 VSM 模型计算文本相似度。

2）计算执行信息相似度（Execution information based Similarities，简称 E-S）：对重现缺陷时程序记录的日志文件，利用计算 NL-S 的方法，以类中的方法或者函数为粒度进行向量空间化，然后计算其文本相似度。

3）检测潜在的重复缺陷报告：将 NL-S 和 E-S 的平均值作为每个缺陷报告的综合相似度。另外做出几个重要的假定，首先定义两个置信度阈值（Credibility Threshold）$CT_{NL\text{-}S}$ 和 $CT_{E\text{-}S}$，分别表示 NL-S 和 E-S 的置信度阈值。

- 当 NL-S 和 E-S 都高于其对应的 $CT_{NL\text{-}S}$ 和 $CT_{E\text{-}S}$ 时，将其放入第一类，用 NL-S 和 E-S 的平均值计算排序；
- 当 NL-S＞$CT_{NL\text{-}S}$，但 E-S＜$CT_{E\text{-}S}$ 时，将其放入第二类，仅用 NL-S 计算排序；
- 当 E-S＞$CT_{E\text{-}S}$，但 NL-S＜ $CT_{NL\text{-}S}$ 时，将其放入第三类，仅用 E-S 计算排序。
- 其他情况下，将其放入第四类，用 NL-S 和 E-S 的平均值计算排序；

由于并非总是可以获得执行信息，总体排序将以 NL-S 为主导，总排序优先级如下：第一类＞第二类＞第三类＞第四类。

4）显示重复缺陷报告列表：根据 top list size，将相似度最高的 n 个重复缺陷报告列出。

P. Runeson[8]等人和 X. Y. Wang[9]等人都是利用 Bugzilla 记录的缺陷进行研究验证。Bugzilla 记录的缺陷中的自然语言信息主要是指概述（Summary）和详细描述（Description）。这两部分的信息是否同等重要，是否需要分别处理或者做不同的加权处理，P. Runeson 等人并没有特别提到。而 X. Y. Wang 等人针对两者用各种不同权重的实验进行了验证。最终实验表明仅使用自然语言时，Summary * 2＋Description 的效果更好一些，若只使用 Summary，则效果最差。但与之相反，如果同时使用自然语言信息和执行信息，当 top list size 变大时，只使用 Summary，效果是最好的。因此在计算相似度时，应该根据具体情况决定 Summary 和 Description 的权值。

1.2 其他无效软件缺陷报告的检测

其他无效缺陷主要包括“不可重现”和“不是问题”这两类缺陷。笔者在实际参与的商业软件测试中，对这两类缺陷报告进行了详细的原因分析。造成“不可重现”缺陷的主要原因有：重现步骤叙述过于简单或被省略；报告者没有注意到该缺陷出现的特有条件，因此缺陷报告中没有明确指出；该缺陷本

身可能与操作时点(例如多线程产生的一些资源互锁等)有关,难以重现。造成"不是问题"缺陷的主要原因有:报告者自身理解或操作错误导致误报;报告者认为某些地方不正确或者不合理,而开发者认为设计本就如此,因此不是问题无需修改。

由以上的原因分析可知,这两类无效缺陷的判定主观性较强,目前关于如何检测这两类无效缺陷的研究基本没有。但文献[5]的分析表明,这两类无效缺陷的比率高达24%,因此展开这方面的研究也是必要的。

2 软件缺陷数据分类

软件缺陷分类是分析软件缺陷的基础,可以采用不同的分类技术[12-14]按照不同的分类体系对软件缺陷进行分类。目前已有很多成熟的分类体系,例如文献[15]中介绍的Putnam等人提出的简单分类法;国家军用标准GJB2437的软件错误分类方法;Thayer等人提出的软件错误性质分类方法;IBM提出的正交缺陷分类(Orthogonal Defect Classification,简称ODC)等。本文仅详述一些重要的分类体系。

2.1 软件缺陷正交分类

ODC是软件缺陷分类中应用最广泛,且最具代表性的一种分类体系,也是很多分类的扩展基础。

IBM定义的最新ODC法[16]中,用8个属性特征对缺陷进行描述,包括发现缺陷的活动(Activity)、缺陷引发事件(Trigger)、缺陷影响(Impact)、缺陷载体(Target)、缺陷类型(Defect Type)、缺陷限定词(Qualifier)、缺陷来源(Source)、缺陷年龄(Age)。

该方法的最大优点在于,可以根据大量缺陷分类后产生的缺陷统计数字,结合缺陷所属信息(所属子系统、模块、特性)进行多维度正交分析,能快速准确定位软件主要质量问题的区域,从而进行有针对的测试或者实施其他改进措施。

2.2 特定范围缺陷分类

特定范围内的软件缺陷分类对于研究该范围内的软件审查(Review/Inspection)、测试(Test)活动有重要的指导作用。

D. Kelly等人[17]为提高审查效率,基于ODC提出了对Defect Type更加详细的ODC-CC分类,并在文献[17]附录列出了详尽的缺陷分类目录。

UML(Unified Modeling Language)已经成为软件开发中标准的设计语言,它用多种图形表现各种设计的内容。UML图中的缺陷不同于传统的代码/文档中的缺陷,C. F. J. Lange等人[18]提出了对UML图中的缺陷的一种分类,将UML文档设计中的典型缺陷归纳为8种。

S. G. Vilbergsdottir等人[19]提出关于可用性(Usability)缺陷的分类,该分类也是在ODC方法之上扩展形成。它用12个属性描述一个缺陷,与ODC的8个属性相比,强调了可用性缺陷特有的属性。

其他特定范围内缺陷的相关分类研究还有很多,如文献[20]对Web领域软件缺陷的详细分类、文献[21]对并发缺陷的统计分类、文献[22]对安全缺陷的分类等,本文在此不做介绍。

3 软件缺陷数据挖掘分析

软件缺陷数据预处理、分类的最终目的是为了挖掘、分析缺陷数据,期望发现一些规律或者规则,改进软件开发的某些环节,提高软件质量。目前软件库挖掘(Mining Software Repository)[23]是软件工程中的研究热点之一,而软件缺陷数据挖掘、分析就是其中的一个重要研究内容。软件缺陷数据挖掘、分析的主要研究内容包括挖掘与检测关联的软件缺陷,利用软件缺陷数据对缺陷特征、缺陷变化趋势等进行统计分析等。

3.1 关联软件缺陷的挖掘与检测

软件中的关联缺陷是一种比较普遍的现象,某些缺陷的存在与否可能导致其他缺陷的变化。软件缺陷的关联关系宏观上可分为两类:一类为横向关联关系,例如同一种类型的缺陷如果在某个功能中发生了,那么在其他类似功能中是否也发生呢?另一类为纵向关联关系,例如一个软件有多个互有影响关系的功能模块,那么一个功能中发生某些缺陷的情况下,对另一个关联的功能模块有何影响?消除横向关联缺陷并不需要对缺陷数据库进行挖掘、分析,主要是依据缺陷数据库中的缺陷现象,逐一排查测试类似功能中是否存在该现象即可,因此不是本文关注的重点。对于纵向关联关系缺陷的检测与消除,需要清楚功能模块之间的关系,以及各类缺陷之间存在的联系规则,这些都可以通过对缺陷库的挖掘和分析得到。

Q. B. Song等人[24]提出了利用数据挖掘中的关联规则挖掘Apriori算法来挖掘软件缺陷数据中的关联规则。关联规则是通过置信度(confidence)和支持度(support)进行度量。文献[24]将Apriori算法项集中的元素实例化为每个缺陷的类型(Defect Type),共包括6种:计算缺陷(Computational Defect)、数据值缺陷(Data value defect)、内部接口缺陷(Internal interface defect)、外部接口缺陷(External interface defect)、初始化缺陷(Initialization defect)、逻辑/控制结构缺陷(Logic/control structure defect)。基于NASA提供的SEL Data软件缺陷数据(以模块为单位),设定不同的最小支持度(10%,20%,30%)和最小置信度(30%,40%,50%,60%),共12种组合并应用到5组训练数据集中,最终得到了60个规则集,包含了1000多条关联规则。例如:

$$Defect\{Comput.\} \wedge Defect\{Ini.\} \Rightarrow Defect\{Ex.\ Interface.\}\ @(34.4\%, 75.1\%)$$

该规则表明Comput,Ini以及Ex. Interface这3种缺陷可能同时发生,在整个数据集中有34.4%的数据同时包含了Comput,Ini以及Ex. Interface缺陷。在包含了Comput,Ini的缺陷中,有75.1%的缺陷也同时包含了Ex. Interface缺陷。由此可以告知开发人员或者测试人员,如果一个模块同时发生了Comput和Ini的缺陷,需要确认是否也存在Ex. Interface缺陷。总体来说,该方法利用软件中已有的缺陷数据库挖掘缺陷之间的典型的联系规则,为软件后续开发测试中消除有关联关系的软件缺陷提供了有效的方法。

景涛等人[25]根据缺陷出现的顺序将缺陷之间的关联关系分为两类:第1类关联指如果缺陷A发生,缺陷B一定不会发生或者很难发生;第2类关联指只有缺陷A发生后缺陷B才可能会发生。他们提出用一种缺陷放回的测试方法来检测关联缺陷。所谓的缺陷放回是指每次输入一个测试用例前,按照一定比例放回一个已消除的缺陷。缺陷放回的方法需要与其他测试策略相结合,很多测试策略可以方便地引入

缺陷放回的方法。文献[25]中以带缺陷放回的随机测试策略为例,用 Space 软件作为测试数据进行了验证。结果表明,缺陷放回的测试方法能够 100%地消除第 2 类关联缺陷,并且可以提高第 1 类关联缺陷的检测率。

关联软件缺陷的挖掘与检测不仅对此类缺陷的消除有重要作用,对软件缺陷预测等方面也会提供有效的信息[26]。

3.2 软件缺陷数据的统计分析

缺陷分析的结果作为重要质量数据反馈到软件开发过程中,可以不断改进软件质量[27],因此以软件缺陷数据为基础进行统计分析的研究很多,本文仅介绍几个有代表性的内容。

3.2.1 When-Who-How 分析

文献[28]以一个早期版本的 Windows 开发为例,提出了用什么时候、谁、怎么样(When-Who-How)三维的方法来分析缺陷数据,以提高软件质量控制的过程。

关于 When 的分析,着重从各个阶段所发现的缺陷比率进行分析。此处的阶段与通常的软件开发阶段不同,主要是软件发布后期的一些里程碑阶段,按照时间顺序依次包括 Preview,Beta1,Beta2,Beta3,RC1,RC2,RTM。结果表明,软件缺陷并不是随着时间推移一直呈递减的收敛趋势,而是 RC1 阶段发现的缺陷数目有明显的上升。该分析结果说明,早期的 Beta1-Beta3 阶段的测试工作还有改善提高的余地。

关于 Who 的分析,将发现缺陷的人分为几类,包括 AppGroup(专门做兼容性测试的小组)、Customer(外部顾客)、InternalCustomer(组织内部的顾客)、Internal(本组件的测试团队)、OtherTestTeams(其他组件的测试团队)、Ingored(其他)。按各类人员发现的缺陷比例来看,OtherTestTeams 的测试效果最好,其次是 Internal。文献[29]中指出,虽然 OtherTestTeams 的测试效果是最好的,但并非任意的 OtherTestTeam 都能有好的测试效果。实际中对具体某个模块只有少数测试团队才有较大贡献,这些团队就构成了该模块的测试伙伴(Test Buddies)。例如,对某模块共有 10 个 OtherTestTeam 参与测试,前 5 个测试团队发现的缺陷占所有发现缺陷的 90%以上,因此对于该模块的测试伙伴就只有前 5 个团队。该分析表明,如果对每个模块找到合适的测试伙伴,测试效果一定会事半功倍。

关于 How 的分析,主要是分析哪些方式是发现缺陷较多的,哪些方式还需要改进。分析表明,该软件中各种方式发现的缺陷比率实际如下:常规测试>静态源代码工具分析>代码/文档审查>顾客发现>压力性能测试>组内构建系统测试>编译版本测试>其他。该分析结果中客户发现缺陷的比率较高,这表明开发过程中没有尽早更好地反映出顾客的使用情况,这将是一个需要改进的问题。

When-Who-How 分析法,除了单独分析每一点之外,也可以将该三维组合进行分析。例如可以分析哪个团队在哪个阶段使用何种方式将是最好的,根据分析结果调整软件质量控制全过程。

3.2.2 软件缺陷变化趋势分析

随着软硬件、测试技术工具等发展更新,经常发生的软件缺陷也在日益发生变化。文献[30]通过对 29,000 条 Bugzilla 中开源软件的缺陷进行分析,发现几个新的变化趋势:1)由于一些有效内存检测工具的广泛使用,内存相关的缺陷有所减少;2)在内存缺陷中,类似于空指针引用错误这类简单缺陷的数目比率较高,这意味着内存相关的检测工具还没有发挥其全部的能量;3)语义相关的缺陷较多,都是程序特有的并且难以修正,这表明应该在设计、测试以及修正此类问题上投入更多的精力;4)安全性缺陷日益增加,它们常常会引起严重的后果。这些整体趋势的变化为软件缺陷的预防和测试都起到了指导作用。

3.2.3 软件缺陷数据与其他数据结合的分析

软件缺陷数据仅仅是软件开发过程中产生的部分数据,如果能与其他更多软件开发过程中的数据结合进行分析,对软件质量的提高和控制将起到更加重要的作用。

软件缺陷预测是软件缺陷研究领域的重要内容之一,研究成果已有很多[31-35]。王青[31]等人将软件缺陷预测领域的各种技术方法进行了全面的总结、归纳。软件预测主要就是用软件缺陷数目、程序代码行数、体积、软件复杂度、软件开发时间、模块分布等各种信息进行综合分析,得出各种预测模型,以预测将开发的软件的缺陷数目以及分布状况。

软件缺陷报告内容与代码的历史版本变化数据相结合,可以从代码修改的历史中,找出做什么样的修改易引起什么样的错误等[36,37];也可以从修改代码的历史者以及缺陷数据中联合分析什么样的缺陷应该分配给谁来修改效果最好[38],什么时候修改效果最好[39]等。

软件缺陷数据与测试、修改缺陷的时间数据结合进行分析,可以预测哪类软件缺陷的修改需要花费多少修改时间[24,40,41],以便于管理者进行时间计划。

4 关于软件缺陷处理研究的展望

软件缺陷数据处理的研究已经发展了很多年,对软件开发起到积极的作用。但是已有的研究中还有以下的不足有待进一步研究解决。

1)重复、无效软件缺陷报告的自动检测

现有的重复报告缺陷检测技术如果仅利用缺陷文本信息,准确率就比较低。而利用文本信息与执行信息相结合,虽然可以提高准确率,但由于执行信息需要单独获取,成本增加、操作复杂而且有可能获取不到执行信息。因此需要一种低成本、操作简单又检测准确率高的方法。文献[42]对缺陷报告从语言学的角度进行了分析,各种词性的不同单词的权重有所不同,可从词性统计等思路解决软件缺陷报告的重复缺陷检测问题。文献[43]中给出了重复数据的一些处理方法。软件缺陷数据也是一种软件数据,因此也可以借鉴该文中的方法展开研究。

"不可重现"和"不是问题"这两类无效缺陷报告所占比率较大,目前却还没有相关的研究,可以通过对缺陷报告一定格式的形式化检测以及与需求的比较等方法尝试解决该问题。

2)针对黑盒测试可检出的缺陷进行分类

软件测试是消除软件缺陷的重要手段,而对于任何一个软件的测试,黑盒测试都占很大比率[44]。但是,目前尚没有专门针对黑盒测试检出的典型缺陷进行分类的研究。如果不进行细化的分类,黑盒测试就会缺少系统全面且有针对性地消除缺陷的依据,因此展开对黑盒测试的典型缺陷进行细致的分类研究,将为基于缺陷的测试技术研究奠定重要的基础。

3)软件缺陷数据的挖掘、分析

目前已有的软件缺陷数据的关联规则挖掘、统计分析等

虽然对提高软件质量起到了一定的帮助作用，但依然不能满足实际软件开发中对数据挖掘与分析的进一步期待。例如模块之间的关联关系挖掘分析也可以通过文献[24]的方法继续研究。如果能够用准确且明显的方式找出关联的软件缺陷、模块联系等，必将对基于关联关系以及组合等的软件测试技术产生重要的影响，因此非常值得进一步深入研究。

参 考 文 献

[1] Parhami B. Defect, Fault, Error, …, or Failure[J]. IEEE Transactions on Reliability IEEE Transactions on Reliability, 1997,46(4):450-451

[2] 梁成才，章代雨，林海静. 软件缺陷的综合研究[J]. 计算机工程，2006,32(19):88-90

[3] Fenton N, Neil M. A critique of software defect prediction models[J]. IEEE Transactions on Software Engineering, 1999, 25(5):675-689

[4] http://www.bugzilla.org

[5] Anvik J, Hiew L, Murphy C C. Coping with an Open Bug Repository[C]// Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange. 2005:35-39

[6] Mockus A, Fielding R, Herbsleb J. Two case studies of open source software development Apache and Mozilla[J]. ACM Transactions on Software Engineering and Methodology (TOSEM),2002,11(3):309-346

[7] Hiew L. Assisted Detection of Duplicate Bug Reports[D]. University of British Columbia, Canada, 2006

[8] Runeson P, Alexandersson M, Nyholm O. Detection of Duplicate Defect Reports Using Natural Language Processing[C]// Proceedings of the 29th International Conference on Software Engineering. 2007:499-510

[9] Wang X Y, Zhang L, Xie T, et al. An Approach to Detecting Duplicate Bug Reports Using Natural Language and Execution Information[C]// Proceedings of the 30th International Conference on Software Engineering. 2008:461-470

[10] Jalbert N, Weimer W. Automated Duplicate Detection for Bug Tracking System[C]// Proceedings of the International Conference on Dependable Systems and Networks. 2008:1-10

[11] Sandusky J, Gasser L, Ripoche G. Bug Report Networks: Varieties, Strategies, and Impacts in a F/OSS Development Community [C] // Proceedings of International Workshop on Mining Software Repositories. 2004:80-84

[12] Han J W, Kamber M. Data Mining Concepts and Techniques [M]. Beijing: China Machine Press, 2005:223-260

[13] Cubranic D, Murphy G C. Automatic bug triage using text categorization[C] // Proceedings of Software Engineering and Knowledge Engineering. 2004:92-97

[14] Podgurski A, Leon D, Francis P, et al. Automated Support for Classifying Software Failure Reports[C] // Proceedings of the 25th International Conference on Software Engineering. 2003: 465-475

[15] 聂林波，刘孟仁. 软件缺陷分类的研究[J]. 计算机应用研究，2004,21(6):84-86

[16] IBM Centre for Software Engineering, Details of ODC [EB/OL]. http: // researchweb. watson. ibm. com/softeng/ODC/DETODC. HTM

[17] Kelly D, Shepard T. A case study in the use of defect classification in inspections[C] // Proceedings of the 2001 Conference of the Centre for Advanced Studies on Collaborative Research. 2001:1-7

[18] Lange C F J, Chaudron M R V. Effects of defects in UML models: an experimental investigation[C] // Proceedings of the 28th International Conference on Software Engineering. 2006: 401-411

[19] Vilbergsdottir S G, Hvannberg E T, Law E L C. Classification of usability problems (CUP) scheme: augmentation and exploitation[C] // Proceedings of the 4th Nordic Conference on Human-computer Interaction. 2006:281-290

[20] Ma P, Tian P. Web error classification and analysis for reliability improvement[J]. Journal of Systems and Software, 2007, 80(6): 795- 804

[21] Lu S, Park S, Zhou Y Y. Learning from mistakes a comprehensive study on real world concurrency bug characteristics[C] // Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems. 2008:329-339

[22] OWASP, Top 10 2007 (The Ten Most Critical Web Application Security Vulnerabilities) [EB/OL]. http: // www. owasp. org/index. php/Top_10_2007, 2007

[23] 刘英博，王建民. 面向缺陷分析的软件库挖掘方法综述[J]. 计算机科学，2007,34(9):1-4

[24] Song Q B, Shepperd M, Cartwright M, et al. Software Defect Association Mining and Defect Correction Effort Prediction[J]. IEEE Transactions on Software Engineering, 2006, 32(2):69-82

[25] 景涛，江昌海，胡德斌，等. 软件关联缺陷的一种检测方法[J]. 软件学报，2005,16(1):18-28

[26] Vandecruys O, Martens D, Baesens B, et al. Mining software repositories for comprehensible software fault prediction models [J]. Journal of Systems and Software, 2008, 81(50):823-839

[27] Jalote P, Agrawal N. Using defect analysis feedback for improving quality and productivity in iterative software development [C] // Proceedings of the Information Science and Communications Technology. 2005:701-713

[28] Jalote P, Munshi R, Probsting T. The When-Who-How analysis of defects for improving the quality control process[J]. Journal of Systems and Software, 2007, 80(4):584-589

[29] Jalote P, Munshi R, Probsting T. Components Have Best Buddies[C]// Proceedings of The 9th International SIGSOFT Symposium on Component-Based Software Engineering. 2006:310-319

[30] Li Z M, Tan L, Wang X H, et al. Have things changed now? - An empirical study of bug characteristics in modern open source software[C]// Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability. 2006:25-33

[31] 王青，伍书剑，李明树. 软件缺陷预测技术研究[J]. 软件学报，2008,19(7):1565-1581

[32] Kapoor K, Bowen P P. Predicting defect-prone software modules using support vector machines[J]. The Journal of Systems and Software, 2008, 81(5):649-660

[33] Fenton N, Neil M, Marsh W, et al. Predicting software defects in varying development lifecycles using Bayesian nets[J]. Information and Software Technology, 2007, 49(1):32-43

（下转第 78 页）

70.71m略大。由此可得，当 x 的值为 50m 时，R 取 70.71m 最好。而且由推论 1 可得，对单个节点而言，传输半径越大（当大于86.2m时），进行一次收发所消耗的能量更大（将用式(10)进行能耗计算，增长更快），所以 R 取 70.71m。为了消除传输半径内的过渡区间的影响，应消除与处于传输半径边缘的节点通信的不稳定性。在工程实施中，取 R 值为 75m，实施效果较为理想。

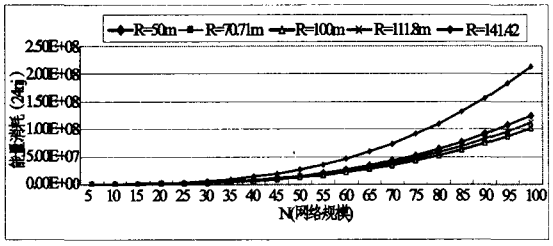

图 3　网络规模与一次全网查询能耗关系图

结束语　本文以信道能耗衰减模型、传输半径模型和数据融合能耗模型 3 个性质定理形式给出了在大规模农田部署中的能耗计算办法。通过这些模型可以对部署间距 x、传输半径 R 及数据包尺寸对网络总能耗的影响进行理论计算，得出在部署间距 x 确定的情况下，通过性质 2 的不同计算模式的 S_{sm_L} 最小值分析，结合性质 1、性质 3 的能耗总量计算，可以求解出最佳的发射半径。实例分析得出了在本文的工程背景下，若部署间距为 50m，则发送半径取 75m 为最佳的结论。这些理论计算所得的能耗模型和传输特性能够为部署能量有效的无线传感器网络拓扑提供指导，为实现无线传感器网络的高层通信协议提供理论基础，类似情况的传感器网络也可以参考本文方法进行部署规划。

此外，值得一提的是，本文的融合模型仅考虑包头融合，未考虑数据本身的融合，如果再对数据进行必要的融合传输，可以进一步节约能耗，延长整个网络的生命周期。同时，关于阈值 d_0 的计算，我们也仅考虑了比较理想的情形，随着实际情况的变化，尤其需要针对无线发射装置的变化改变各项参数的设置。而且需要注意的是，实际与理论计算有一定的误差，需要在应用中进行标定和修正。

参 考 文 献

[1] Wang Ning, Zhang Naiqian, Wang Maohua. Wireless sensors in agriculture and food industry—Recent development and future perspective[J]. Computers and Electronics in Agriculture, 2006, 50(1): 1-14

[2] Ephremides A. Energy concerns in wireless networks[J]. IEEE Wireless Communications, 2002(8): 48-59

[3] Rodoaplu V, Meng T H. Minimum Energy Mobile Wireless Metworks [J]. IEEE J. Select. Areas Communications, 1999, 17(8): 1333-1334

[4] Li L, Halpen J Y. Minimum Energy Mobile Wireless Networks Revisited [C]// IEEE International Conference on Communications(ICC), 2001. http: // www. cs. cornell. edu/Info/People/halpern/papers/icc01. ps

[5] Wattenhofer R. Distributed Topology Control for Power Effi - cient Operation in Multihop Wireless AdHoc Networks[C]// IEEE INFOCOM, 2001. https: // eprints. kfupm. edu. sa/35651/1/35651. pdf

[6] Li L. Analysis of a Cone - based Distributed Topology Control Algorithm for Wireless Multihop Networks[C]// ACM Symposium on Principle of Distributed Computing (PODC), 2001. http: // ieeexplore. ieee. org/

[7] Lin K, Zhao H. Energy Prediction and Routing Algorithm in Wireless Sensor Networks [J]. Journal on Communications, 2006, 27(5): 21-23

[8] Wang A, Heinzelman W B, Sinha A, et al. Energy-scalable protocols for battery-operated microsensor networks[J]. Journal of VLSI Signal Processing, 2001, 29(3): 223-237

[9] 朱红松，孙利民，徐勇军，等. 基于精细化梯度的无线传感器网络汇聚机制及分析[J]. 软件学报，2007，18(5)：1138-1151

(上接第 25 页)

[34] Nagappan N, Ball T, Zeller A. Mining Metrics to Predict Component Failures[C] // Proceedings of the 28th International Conference on Software Engineering. 2006: 452-461

[35] Boetticher G D. Nearest neighbor sampling for better defect prediction[J]. ACM SIGSOFT Software Engineering Notes, 2005, 30(4): 1-6

[36] Livshits B, Zimmermann T. DynaMine Finding Common Error Patterns by Mining Software Revision Histories[C] // Proceedings of 13th International Symposium on Foundations of Software Engineering (ESEC/FSE'05). 2005: 296-305

[37] Weissgerber P, Diehl S. Mining Version Histories to Guide Software Changes[J]. IEEE Transactions on Software Engineering, 2005, 31(6): 429-445

[38] Anvik J, Hiew L, Murphy G C. Who Should Fix This Bug[C] // Proceedings of the 28th International Conference on Software Engineering. 2006: 361-370

[39] Śliwerski J, Zimmermann T, Zeller A. When Do Changes Induce Fixes[C]// Proceedings of the 2005 International Workshop on Mining Software Repositories[C]. 2005: 1-5

[40] Weiβ C, Premraj R, Zimmermann T, et al. How Long Will It Take to Fix This Bug[C] // Proceedings of the Fourth International Workshop on Mining Software Repositories. 2007: 1-8

[41] Hooimeijer P, Weimer W. Modeling bug report quality[C] // Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering. 2007: 34-43

[42] Ko A J, Myers B A, Chau D H. A Linguistic Analysis of How People Describe Software Problems [C] // Proceedings of the Visual Languages and Human-Centric Computing. 2006: 127-134

[43] 韩京宇，徐立臻，董逸生. 数据质量研究综述[J]. 计算机科学，2008，35(2)：1-5

[44] Schroeder P J, Korel B. Black-box test reduction using input-output analysis[C] // Proceedings of the 2000 ACM SIGSOFT International Symposium on Software Testing and Analysis. 2000: 173-177