自动分析软件缺陷报告间相关性的方法研究 *

李 楠, 王晓博, 刘 超
(北京航空航天大学 软件工程研究所, 北京 100191)

摘 要: 针对缺陷报告相关性分析的研究主要采用计算其文本信息相似度的方法使其查全率和查准率并不理想,提出了一种将结构化信息相似度与文本信息相似度计算相结合的方法,即同时提取出缺陷报告中的文本信息(包括主题和详细描述)以及结构化信息(包括补丁、异常堆栈和代码片段),从缺陷外部表现和内部特征两个角度共同衡量缺陷报告间的相关性。通过对 Eclipse 系统中的 1 000 个缺陷报告进行实验,结果显示,增加结构化信息相似度计算,可以有效地将缺陷报告间相关性分析的查准率和查全率均提高到 90% 左右。

关键词: 相关缺陷报告; 结构化信息; 相似度
中图分类号: TP311.5 **文献标志码**: A **文章编号**: 1001-3695(2010)06-2134-06
doi:10.3969/j.issn.1001-3695.2010.06.040

Approach to automatic analysis for relevance among software bug reports

LI Nan, WANG Xiao-bo, LIU Chao
(*Software Engineering Institute*, *Beihang University*, *Beijing* 100191, *China*)

Abstract: The approaches to the relevance analysis of bug reports were studied based on natural language processing technology, but the precision and recall were hard to improve. The paper proposed the new approach based on both of structure information (including patches, exception stack and code fragments). With these information, the relevance of bug reports detected by using the similarity analysis of the structure and text information. It used 1 000 bug reports form Eclipse to test, and the experimental result shows that it can effectively improve precision rate and recall rate to about 90% by adding similarity analysis of the structure information.

Key words: related bug reports; structure information; similarity

0 引言

随着软件项目规模的不断增长,使得维护工作日趋复杂。在软件生命周期中,通过对其进行不断的测试会产生各种各样的缺陷报告,很多软件系统直接基于这些缺陷报告进行软件维护工作[1]。存储这些缺陷报告并进行合理管理的系统即为软件缺陷管理系统,如常用的开源软件缺陷管理系统 Bugzilla。Bugzilla 能够对软件缺陷进行具体记录,包括缺陷状态跟踪及缺陷修改过程等。它允许用户和开发或测试人员提交缺陷报告,由缺陷分流人员将缺陷报告分配给相关开发人员,由他们对缺陷进行修复,通过此过程保证软件系统的质量[2]。Bugzilla 中定义的缺陷报告包括一些预定义内容,如缺陷主题、缺陷描述、缺陷优先级、严重级、产品版本号、所属构件等。除此之外,开发人员或用户还可以针对缺陷进行讨论和添加附件信息,如补丁、异常堆栈或代码片段等。

在软件开发和不断变更的整个生命周期中,将产生大量的缺陷报告,其中有许多报告是彼此相关的,甚至是重复的。例如 Mozilla 系统在 2003—2005 年产生的约 27 000 个缺陷报告中,重复缺陷报告约占 36.3%;Eclipse 系统在 2001—2007 年中共产生约 213 000 个缺陷报告。在这些缺陷报告中,约 30% 的缺陷报告是具有相关性的,其中约 60% 是重复缺陷报告。缺陷报告间的相关性主要体现在对缺陷的外部表现、产生原因、代码环境或解决方法等方面描述的相似性。这些相关的缺陷报告将被分流人员或开发人员标志为“dependson”或“see bug”,以便于开发人员根据这些关联,将相关的缺陷报告进行统一的分析和修改,以避免对该缺陷的修复而引起其他新的缺陷。缺陷的相关性分析也可以提供新报告缺陷的扩展信息或修复该缺陷的参考方法[3]。除此之外,还有助于将缺陷报告分配或抄送给相关开发人员。

传统的缺陷报告相关性分析是通过人工检测来完成,而大型软件系统在整个生命周期中将产生相当数量的缺陷报告,例如 Eclipse 系统,在 2007 年中向 Bugzilla 提交了约 42 000个缺陷报告,平均一天产生 115 个缺陷报告。在数据庞大的缺陷历史记录中分析缺陷报告间的相关性,需要耗费大量的时间和精力,所以自动分析软件缺陷报告间的相关性具有重要意义。

1 相关的研究现状

缺陷管理系统可以将软件测试过程中发现的缺陷、缺陷信息及缺陷发生的背景信息全面详细地记录下来,构成缺陷信息数据仓库。分析挖掘已有缺陷中的信息及其相关性,有助于指导软件测试工作,对提高缺陷发现率和改善软件质量是一种行

收稿日期: 2009-10-10; **修回日期**: 2009-11-17 **基金项目**: 国家自然科学基金资助项目(90718018);国家“863”计划资助项目(2007AA010302)

作者简介:李楠(1984-),女,天津人,硕士研究生,主要研究方向为软件工程、信息检索(amanda1984@sei.buaa.edu.cn);王晓博(1982-),男,博士研究生,主要研究方向为软件工程、数据挖掘;刘超(1958-),男,教授,博导,研究方向为软件工程、软件测试.

之有效的方法[4]。

当前国内外对于缺陷报告的分析研究主要包括:

a)对软件缺陷数据的分析。软件开发过程中的历史信息为缺陷分析提供了很有价值的经验数据,需要利用有效方法对这些数据进行如下缺陷分析[5]:

(a)将缺陷报告中的属性值按照一元数据和多元数据进行分析,并说明缺陷数据分析对软件质量保证、项目管理和过程改进具有重要意义[6]。

(b)通过文本挖掘。统计分析等方法,对软件缺陷预处理、缺陷分类和缺陷数据挖掘[7]。

b)软件缺陷分类的研究如下:

(a)缺陷分类是缺陷管理的基础。当前国内研究主要使用正交缺陷分类,并在正交缺陷分类的基础上制订出适合软件组织自身情况的软件缺陷分类方法[8]。

c)自动检测重复缺陷报告如下:

(a)通过缺陷报告的主要属性和文本描述构造分类器,对 Mozilla 系统缺陷报告进行实验,实验结果显示该方法可以自动过滤 8% 的重复缺陷报告[9]。利用缺陷报告间的文本相似度进行文本聚类[10],并通过为分流人员提供候选分配人员列表的方式实现合理分配缺陷报告,实验结果显示查全率和查准率均为 70% 左右[11]。

(b)通过缺陷报告文本描述和缺陷执行信息(如缺陷重现步骤等)两者来共同检测重复缺陷报告,并对 Firefox 系统的缺陷报告进行实验,实验结果显示,对比仅采用文本相似度分析过滤 43% ~72% 的重复缺陷报告,增加缺陷执行信息分析可以过滤 67% ~93% 的重复缺陷报告[12]。

d)自动分流缺陷报告。通过提取缺陷报告的文本描述、组件、操作系统、硬件、软件版本、相关代码所属的开发人员等八个属性来构造一个候选器,为分流人员提供可能适合修复该缺陷的开发人员候选列表[13],并对 Eclipse 系统缺陷报告进行实验,结果显示该方法查准率达到 80% 左右,帮助分流人员对缺陷报告进行合理分配[14]。

综上所述,当前对于缺陷报告间相关性的研究主要以计算缺陷报告间的文本相似度为主,并取得了显著的研究成果,但由于查全率和查准率并不十分理想,造成了一定的局限性。

本文在当前研究的基础上,提取缺陷报告中的另一项有价值的信息——结构化信息,包括补丁、异常堆栈和代码片段。这项信息存储于附件列表或嵌套于文本描述中。通过统计,含有这些信息的缺陷报告具有相当数量,例如 Eclipse 系统 2001—2007 年中提交的约 213 000 个缺陷报告中,含有结构化信息的缺陷报告约占 35% 左右,补丁信息约占 16%,异常堆栈信息约占 14%,代码片段约占 5%,这些结构化信息从不同角度记录了程序中与缺陷报告相关的某些内部特征[15]。

对比文本信息,结构化信息具有以下特点:a)更能反应缺陷的产生原因和代码环境,而不受自然语言多义性的影响;b)更能反映缺陷内部的本质,而这些可能是文本描述中没有提到的[16]。本文利用文本相似度分析技术来处理缺陷报告的主题和详细描述,并结合对缺陷报告中的结构化信息的相似度分析,两者共同衡量缺陷报告间的相关性。

文本主要贡献:a)说明仅使用文本信息来分析缺陷报告间的相关性存在局限,本文提取缺陷报告中另一项有价值的信息——结构化信息,包括补丁、异常堆栈和代码片段;b)利用文本相似度分析技术来处理缺陷报告的主题和详细描述,并结合对缺陷报告中的结构化信息的相似度分析,通过这两种相似度从缺陷外部和内部共同衡量缺陷报告间的相关性。c)通过设置不同参数进行实验,并对结果进行分析比较。

本文通过实例来说明缺陷报告间的相关性以及需要结构化信息相似度与文本信息相似度计算相结合来分析缺陷报告间的相关性的原因,介绍缺陷报告中的结构化信息及其提取方法,文本信息和结构化信息相似度的计算方法,以及如何通过这两种相似度从缺陷外部和内部来共同衡量缺陷报告间的相关性。

2　相关缺陷报告

在本章中,以 Eclipse 系统中的相关缺陷报告为例,解释需要结构化信息相似度与文本信息相似度来共同衡量缺陷报告间的相关性的原因。缺陷报告中的文本信息通常包括缺陷主题和详细描述两方面,为简要起见,在这里仅显示缺陷主题。

本文定义缺陷报告间的相关性主要包括:a)重复缺陷报告,如缺陷报告#212109 和#212114;b)在问题现象、产生原因、代码环境或解决方法等方面相关的缺陷报告,这种关联被标志为"dependson"或"see bug":(a)文本描述相似,如缺陷报告#208821 和#211243;(b)文本描述不相似,但结构化信息具有一定相似度,如缺陷报告#188103 和#190945。

缺陷报告#212109 和#212114 是一对已被标志为"duplicate"的重复缺陷报告,两者的缺陷主题如下:

212109:[Automation][Acceptance]NPE is thrown out when previewing report

212114:[Automation][Regression]NPE is thrown out when preview a report containing a hidden chart

可见两者具有很多共同词汇,如"[Automation][Acceptance]""NPE""thrown out"等,通过文本相似度分析可以检测出这对缺陷报告是重复的。

208821 :Change SDK to use individual source bundles

211243 :Migrate org. junit4 to individual source bundle

这两个缺陷报告的文本描述具有一定的相似度,两者已被开发人员标志为"dependson"。通过文本相似度分析可以检测出这种相关性。

188103 :[1.5][compiler] Inappropriate usage of HashSet

190945:[1.5][compiler] failure to compile complex generic code </short_desc >

可见两者的文本描述不相似,但通过比较两者的异常堆栈信息,发现是相似的,即两者产生问题的原因或代码环境相关。在#188103 中,已被分流人员标志出"see bug 190945"。

通过以上相关缺陷报告实例显示,文本信息和结构化信息对于缺陷报告间的相关性分析同等重要:a)文本信息通常描述缺陷的外部表现,而结构化信息则反映缺陷的内部特征;b)由于自然语言通常包含多义性和不确定性,而结构化信息通常是确定的数据[17],采用文本信息相似度和结构化信息相似度相结合的方法,从缺陷外部和内部来共同分析缺陷报告间的相关性。

3　缺陷报告间的相关性分析方法

如果新提交的缺陷报告与其他已有缺陷报告存在相关性，开发人员需要根据这些关联，过滤掉重复缺陷报告，或者对于这些相关的缺陷进行统一的分析和修改，以避免对该缺陷的修复而引起其他新的缺陷。

本章将具体介绍缺陷报告间的相关性的分析方法，包括：a）提取缺陷报告中的结构化信息；b）通过计算缺陷报告间的文本相似度和结构化信息相似度，从缺陷外部和内部来共同衡量缺陷报告间的相关性。

3.1　缺陷报告中的结构化信息及其提取方法

结构化信息大部分以附件形式存储于缺陷报告中，或嵌套于文本描述中，主要包括补丁、异常堆栈和代码片段。对于以附件形式存储的，可直接获得；对于嵌套于文本描述中的，需要采用相应的方法将它们提取出来[18]。

a）补丁：用来更新原文件或修复问题的一段代码。通常更新或修改多个文件。以统一的“diff”格式出现，标准格式为 UNIX 命令 diff -u*N* file1 file2（*N* 为数字，file1、file2 为文件名）。结构如图 1 所示。

Patch Index：惟一标志每个子补丁区。

Patch Header：包括原文件和新文件的文件名、版本、日期以及 Revision Control System（RCS）文件名。

Patch Hunks：描述文件内部的改变。包括一个 Hunk Header 和多个 Hunk Lines。

Hunk Header：原文件被修改的位置，以“@@”作为起始和终止符。

Hunk Lines，即包括加入行：由单一的“+”开始，表示在原始文件的给定位置上加入这些行；删除行：由单一的“-”开始，表示在原始文件的给定位置上删除这些行。

补丁信息提取：

（a）首先寻找标志符“Index:”，如果没有则表示没有补丁信息；如果仅找到一个，则从“Index:”开始到结束都作为潜在补丁区域；若找到多个“Index:”，则每两个“Index:”间的信息作为潜在的补丁区域。

（b）在潜在的补丁区域中提取 Patch Headers 和 Hunk Header，每两个 Hunk Header 间作为潜在 Hunk 区域，在其中提取 Hunk Lines。

b）异常堆栈：程序执行时所引起的异常列表，用于帮助修复缺陷或定位缺陷产生原因。结构[19]如图 2 所示。

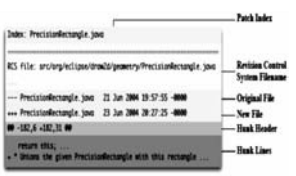

图1　补丁信息示例

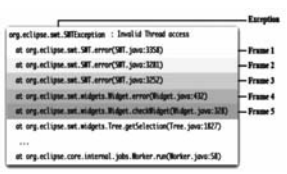

图2　异常堆栈信息示例

它包括：方法调用行：显示方法名和方法所属类名，及其在代码源文件中的行号。异常堆栈信息提取：

（a）采用正则表达式定义常见的异常集合 *E*；

（b）从文本描述中检测是否含有集合 *E* 中的异常，一旦找到，则将其后续行与方法调用行模板进行匹配。

c）代码片段：描述问题、问题产生的代码环境或修复问题的一段代码。结构如图 3 所示。

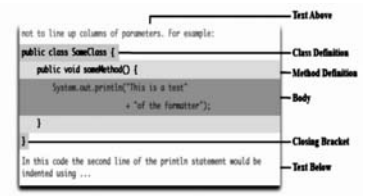

图3　代码片段信息示例

代码片段信息提取：利用具有程序特性的语句，如条件判断“if”“else”或赋值语句等，来寻找嵌套于文本信息中的代码区域。

（a）将类定义（class definition）、方法定义（method definition），条件判断、赋值语句等具有程序特性的关键词作为集合 $K=\{\text{class, if, else, for, while, switch, “=”,}\cdots\}$；

（b）用正则表达式描述方法定义的基本文法格式 *F*；

（c）在缺陷文本描述中寻找是否出现集合 *K* 中的关键词，或符合文法格式 *F* 的字符串，如果找到，则检测周围信息，以发现类定义的区域、方法定义区域或一个程序片段。

综上所述，采用相应方法对嵌套于文本信息中的补丁、异常堆栈、代码片段进行检测和提取，通过随机抽取 Eclipse 系统中的 10 000 个缺陷报告进行实验，发现该提取方法具有很高的准确率，基于此，可以将结构化信息作为自动分析缺陷报告间相关性的重要因素之一，来展开下一步研究。

3.2　相关缺陷报告检索方法

当前研究的基本理论：如果两个缺陷报告是相关的，那么这两个缺陷报告很有可能是相似的。通过采用信息检索的方法自动检测具有相似度的缺陷报告，从而分析缺陷报告间的相关性。

信息检索研究的主要目标是采用相应算法和模型从信息库中得到所需信息。这里的信息主要包括自然语言文本信息和结构化信息[20]。

3.2.1　缺陷报告间的文本信息相似度计算

自然语言处理技术主要包括：

a）标记化。通过去掉字母大写、标点符号、括号等，将一系列词汇流转换为标志符流。每个标志符代表一个单词。

b）提取词干，相同的词汇可以不同的语法形式存在。提取词干的作用在于识别每个词汇的本质信息。

c）去掉停用词。停用词指一些非常普遍使用的词汇，如“the”“that”“what”等，这些词没有特别的含义，对相似度分析作用很小。

d）将文本信息表示为空间向量。将以上操作得到的词汇基于向量空间模型（vector space model）表示为一个多维矩阵，每一个向量代表一个词汇[21]。

e）相似度计算。两个文本间的相似度计算定义为空间向量模型中两个向量间的余玄值[22]。

用以上方法对缺陷报告中的文本信息进行处理，并计算缺陷报告间的文本相似度。在本文中提取缺陷报告的主题描述和详细描述作为缺陷报告的文本信息。

缺陷报告文本信息的标记化、提取词干、去掉停用词处理：

如缺陷报告#204401 的缺陷主题描述：“[Smoke] OutofMemory occurs when preview attached report in Web viewer.”。表 1 为标记化、提取词干和去掉停用词的处理过程。

表 1　标记化、提取词干和去掉停用词的处理过程

缺陷报告文本信息	缺陷主题描述
原文本	[Smoke] OutofMemory occurs when preview attached report in Web viewer
标记化	smoke outofmemory occurs when preview attached report in Web viewer
提取词干	smoke outofmemory occur when preview attach report in Web viewer
去掉停用词	Smoke outofmemory occur preview Web viewer

注:因为在缺陷报告文本描述中,“attach”和“report”也是普遍出现的词,所以也作为停用词处理。

将缺陷报告的主题描述和详细描述两项文本信息按照以上过程处理后,分别建立两个语料库,每个语料库中的所有文本构成一个包含 n 个独立词汇的集合,基于向量空间模型,将语料库中的每个文本表示为一个 n 维向量 v, $v[i]$ 表示第 i 个词汇的权重,这里一般采用逆词频率方法计算词汇权重[23,24]。

逆词频率,在自然语言处理中用于描述词汇的重要性。它与文本语料库宽度(即词汇总数),和该词汇出现的次数有关。它基于假设:在文本中出现次数越少的词,对于文本相似度计算越重要。如果一个词汇在文本中多次出现,那么该词对相似度分析影响很小,它的权重值通常被设定得很小。

在实验中在缺陷报告文本描述的相似度计算中,使用了逆词频率作为词汇权重计算方法,但通过结果显示,使用逆词频率对缺陷报告间相关性分析的结果影响很小。

同一语料库中的两个文本向量 v_1 和 v_2 间的相似度定义为余玄相似度:

$$\text{similarity} = \cos(\theta) = \frac{v_1 \cdot v_2}{|v_1| \times |v_2|}$$

通过以上方法得到缺陷报告间主题描述和详细描述的相似度后,通过两者来度量缺陷报告间的文本相似度。

首先分别设定缺陷主题的相似度阈值(*TT*)和详细描述的相似度阈值(*DT*),并根据这两个阈值以及缺陷主题的相似度(*TS*)和详细描述的相似度(*DS*)将缺陷报告划分为三类:

(a)*TS* 和 *DS* 分别超过 *TT* 和 *DT* 的值,即 $TS \geqslant TT$ 且 $DS \geqslant DT$;

(b)*TS* 和 *DS* 都没有超过 *TT* 和 *DT* 的值,即 $TS < TT$ 且 $DS < DT$;

(c)*TS* 和 *DS* 两者中有且仅有一个超过其对应阈值,即 $TS \geqslant TT$ 且 $DS < DT$,或 $DS \geqslant DT$ 且 $TS < TT$。

以上三种情况下,缺陷报告间文本相似度(*TDS*)定义为

$$\begin{cases} \dfrac{(TS + DS)}{2}, & TS \geqslant TT \text{ 且 } DS \geqslant DT \text{ 或 } TS < TT \\ \text{且} & DS < DT \\ TS, & TS \geqslant TT \text{ 且 } DS < DT \\ DS, & DS \geqslant DT \text{ 且 } TS < TT \end{cases}$$

3.2.2　缺陷报告间的结构化信息相似度计算

得到提取出来的三种类型的结构化信息后,通过比较它们的相似度特征来计算结构化信息间的相似度。

1)补丁　以图 2 为例,采用以下三项信息作为补丁的相似度特征:

a)以“Index:”作为起始标志符的 Patch Index,这项信息通常为被修改或更新的原文件名,如“Index: precisionRectangle.java”;

b)表示源文件被修改位置的 Hunk Header,如“@@ -182,6 +182,31”;

c)表示源文件被修改内容的 Hunk Lines,包括以“+”起始的“加入行”和以“-”起始的“删除行”。

2)异常堆栈　以图 3 为例,本文中选取异常堆栈栈顶的五个调用序列作为异常堆栈的相似度特征。

3)代码片段

(a)对明确给出代码片段所属文件名的,提取其文件名作为相似度特征之一。

(b)对于没有标志出所属文件名的代码片段,提取代码片段的相关信息,包括包名、类名、接口名、方法名、方法返回值类型、方法参数(参数个数、类型、参数名)等作为相似度特性。如图 4 所示的代码片段示例,获得其类名“someClass”、方法名“someMethod”和方法返回值类型“void”作为相似度特征。

3.2.3　度量缺陷报告间的相关性

通过以上方法得到了缺陷报告间的文本相似度和结构化信息相似度(对于含有结构化信息的缺陷报告而言),采用以下方法度量缺陷报告间的相关性:

首先分别设定缺陷报告文本信息的相似度阈值(*TDT*)和结构化信息的相似度阈值(*AT*)的值,并根据这两个阈值以及文本相似度(*TDS*)和结构化信息的相似度(*AS*)度量相关性:

$$\text{缺陷报告间}\begin{cases} \text{相关}, & TDS \geqslant TDT \text{ 或 } AS \geqslant AT \\ \text{不相关}, & TDS < TDT \text{ 且 } AS < AT \end{cases}$$

4　实验及结果分析

为了验证本文提出的缺陷报告间的相关性分析方法的有效性,本章设计了相应实验,并对实验结果进行了分析。

实验采用 Eclipse 系统 2007 年 11 月和 12 月提交的 1 000 个缺陷报告作为实验数据,首先采用对缺陷报告中的文本信息进行相似度计算来分析缺陷报告间的相关性,得出实验结果并加以分析。在此基础上,增加对缺陷报告中结构化信息的提取和相似度分析,通过文本相似度和结构化信息相似度来共同分析缺陷报告间的相关性。

实验数据:

a)采用的实验数据是开源软件 Eclipse 系统的缺陷报告。由于 Eclipse 系统包含很多模块,并被不同类型的用户所使用,实验数据集具备了多样性。

b)分别抽取 2007 年 11 月和 12 月提交的前 1 000 个缺陷报告,因为太近时期提交的缺陷报告可能存在一些错误未被改正。

建立实验:

实验 1　通过文本相似度分析缺陷报告间的相关性。

对缺陷报告的主题和详细描述进行文本相似度计算,分别采用以下三种阈值设定来进行实验,相关性分析结果的查全率和查准率如表 2 和表 3 所示。

(a)设定 $TT = 0.60$, $DT = 0.50$,相关缺陷相似度阈值为 0.55;

(b)设定 $TT = 0.50$, $DT = 0.40$,相关缺陷相似度阈值为 0.45;

(c)设定 $TT = 0.40$, $DT = 0.30$,相关缺陷相似度阈值为 0.35。

表 2　实验 1 查全率结果

	方案一			方案二		
阈值设定	(a)	(b)	(c)	(a)	(b)	(c)
查全率/%	43	61	79	43	62	81

注:方案一没有调整词汇特征项的权重,方案二采用逆词频率作为词汇特征项的权重。

表 3　实验一查准率结果

	方案一			方案二		
阈值设定	(a)	(b)	(c)	(a)	(b)	(c)
查准率/%	91	84	66	91	86	67

方案一没有调整词汇特征项的权重,方案二采用逆词频率作为词汇特征项的权重。

通过结果分析得到:方案二比方案一的查全率和查准率相差很少,说明通过逆词频率方法设计词汇权重并不能从根本上提高分析缺陷报告间相关性的准确度。阈值设定(a)虽然使检索相关缺陷报告的查准率达到了 90% 以上,但查全率却比较低,仅有 40% 左右;而阈值设定(c)虽然使查全率达到 80% 以上,但查准率却也降低到 60% 左右,说明随着缺陷主题相似度和详细描述相似度阈值的降低,虽然可以找到更多的相关缺陷从而提高查全率,但同时也将一些实际上并不相关的缺陷报告检索出来,从而降低了查准率。笔者认为由于某些不相关的缺陷报告的文本描述中存在共有词汇,或文本描述没有反映出缺陷的本质特征,造成了缺陷报告相关性分析结果不够理想。

实验 2 在实验 1 的基础上,增加对结构化信息相似度计算来共同分析缺陷报告间相关性。

基于实验 1 中表现出的仅用文本信息相似度来分析缺陷报告间相关性的不足,在实验 2 中增加了对缺陷报告中的结构化信息(包括补丁、代码片段、异常堆栈)的相似度(*AS*)的计算,通过文本相似度和结构化信息相似度来共同分析缺陷报告的相关性。

当文本相似度(*TDS*)超过了文本相似度阈值(*TDT*),或者结构化信息相似度(*AS*)超过了结构化信息相似度阈值 *AT*,则认为缺陷报告间是有相关性的。分别采用以下三种阈值设定来进行实验,相关性分析的查全率和查准率结果如表 4 和表 5 所示。

(a)设定 $TT=0.60, DT=0.50, TDT=0.55, AT=0.60$;

(b)设定 $TT=0.50, DT=0.40, TDT=0.45, AT=0.55$;

(c)设定 $TT=0.40, DT=0.30, TDT=0.35, AT=0.50$。

表 4　实验 2 查全率结果

	方案一			方案二		
阈值设定	(a)	(b)	(c)	(a)	(b)	(c)
查全率/%	84	88	88	84	89	90

方案一没有调整词汇特征项的权重,方案二采用逆词频率作为词汇特征项的权重。

表 5　实验 2 查准率结果

	方案一			方案二		
阈值设定	(a)	(b)	(c)	(a)	(b)	(c)
查全率/%	91	86	71	91	87	73

方案一没有调整词汇特征项的权重,方案二采用逆词频率作为词汇特征项的权重。

通过实验结果可见,文本信息相似度和结构化信息相似度共同分析缺陷报告间的相关性可以得到较高的查全率和查准率。如图 4 和 5 所示,实验 2 在结果的查全率和查准率方面对比实验 1 都有所提高,尤其在查全率方面如阈值设定(a)中,实验 2 在保证查准率在 90% 左右的基础上,将查全率从 40% 提高到 85% 左右。这主要是由于结构化信息更能反映缺陷的本质特征,包括产生原因和代码环境,且避免了自然语言多义性对分析结果的不利影响。

在实验的 1 000 个缺陷报告中,通过统计得到三种结构化信息所占比例,以及它们对查全率的提高,将三种结构化信息对相关性分析的贡献定义为:该结构化信息对查全率的提高与含有该结构化信息的缺陷个数的比值,如表 6 所示。

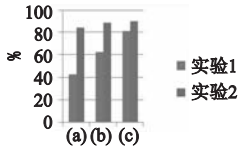

图4　查全率比较

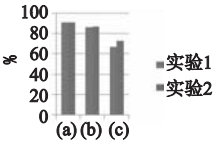

图5　查准率比较

表 6　三种结构化信息含量和对实验结果的贡献

	补丁	代码片段	异常堆栈
含有该信息的缺陷报告个数	159	89	137
该信息对查全率的提高/%	18.6	7.8	14.6
该信息的贡献/%	0.116 981	0.08 764	0.106 569

其中代码片段的贡献最小,原因主要是由于某些代码片段没有标志出所属源文件名,且其代码信息对相似度分析没有帮助。通过分析发现,这是由于这些代码片段不是来自 Eclipse系统的文件,也不是来自缺陷提交者对系统原文件的修改,而是由缺陷报告提交者编写的测试代码等,如图 6 所示。

这里类名 A、X 等,以及方法名 foo 是缺陷报告提交者自己随意命名的,所以无法帮助分析代码片段间的相似度。

5　结束语

本文提出一种自动分析软件缺陷报告间相关性的方法,采用缺陷报告中的文本信息相似度和结构化信息(包括补丁、代码片段和异常堆栈)相似度,从缺陷外部和内部来共同衡量缺陷报告间的相似度,从而分析缺陷报告间的相关性。本文从 Eclipse 系统中抽取了 1 000 个缺陷报告作为实验数据,结果显示:对比仅使用文本信息相似度来分析缺陷报告相关性的结果,增加结构化信息相似度计算可以在保证查准率 90% 左右的条件下,将查全率从 40% 提高到 85% 左右。

缺陷报告间相关性的自动分析可以帮助开发人员根据这些关联,将相关的缺陷报告进行统一的分析和修改,以避免对该缺陷的修复而引起其他新的缺陷。缺陷的相关性分析也可以提供新报告缺陷的扩展信息或修复该缺陷的参考方法,从而有利于提高软件测试的效率和软件质量。

本文提及的自动分析缺陷报告间相关性的技术也存在一些不足,例如对于代码片段信息,一些由缺陷提交人员编写的测试代码等,由于命名是随意的,会对代码片段间相似度的计算形成影响;另外,相关性阈值的设定问题也是影响查全率和查准率的重要因素,对此笔者将对不同的系统的缺陷报告进行实验,以便给用户较合理的阈值设定意见。

参考文献:

[1] CANFORA G, CERULO L. How software repositories can help in reoslving a new change request[C]//Proc of Workshop on Empirical Studies in Reverse Engineering. 2005.

[2] ANVIK J, HIEW L, MURPHY G C. Who should fix this bug[C]//Proc the 28th of International Conference on Software Engineering. New York: ACM Press, 2006:361-370.

[3] FISCHER M, PINZGER M, GALL H. Analyzing and relating bug report data for feature tracking[C]//Proc of the 10th WCRE IEEE. Washington DC: IEEE Compnter Society, 2003:90-99.

[4] 韩卫岗,周红建,赵禄丰. 软件缺陷信息分析研究[J]. 计算机工程与设计,2008,29(13):3381-3383,3387.

[5] 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):1-4,11.

[6] 刘海 郝克刚. 软件缺陷数据的分析方法及其实现[J]. 计算机科学,2008,35(8):262-264.

[7] 李宁,李战怀. 软件缺陷数据处理研究综述[J]. 计算机科学,2009,36(8):21-25,78.

[8] 尹相乐,马力,关昕. 软件缺陷分类的研究[J]. 计算机工程与设计,2008,29(19):4910-4913.

[9] JALBERT N, WEIMER W. Automated duplicate detection for bug tracking systems[C]//Proc of IEEE International Conference on Dependable Systems and Networks with FTCS and DCC, DSN. 2008: 52-61,24-27.

[10] CUBRANIC D, MURPHY G C. Automatic bug triage using text classification[C]//Proc of SEKE. [S.l.]: KSI Press, 2004: 92-97.

[11] ANVIK J K. Assisting bug report triage through recommendation [R]. [S.l.]: University of British Columbia, 2007.

[12] WANG Xiao-yin, ZHANG Lu, ANVIK J, *et al*. An approach to detecting duplicate bug reports using natural language and execution information[C]//Proc of the 30th International Conference on Software Engineering. New York: ACM Press, 2008:461-470.

[13] WEISS C, PREMRAJ R, ZIMMERMANN T, *et al*. How long will it take to fix this bug[C]//Proc of the 4th International Workshop on Mining Software Repositories. Washington DC: IEEE Computer Society, 2007.

[14] ANVIK J. Automating bug report Assignment[C]//Proc of the 28th International Conference on Software Engineering. New York: ACM Press,2006:937-940.

[15] BETTENBURG N, JUST S, SCHRTER A, *et al*. Quality of bug reports in Eclipse[C]//Proc of OOPSLA Workshop on Eclipse Technology Exchange. New York: ACM Press, 2007: 21-25.

[16] BETTENBURG N, JUST S, SCHRTER A, *et al*. What makes a good bug report[C]//Proc of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York: ACM Press, 2008:308-318.

[17] KO A J, MYERS B A, CHAU D H. A linguistic analysis of how people describe software problems[C]//Proc of IEEE Conference on Visual Language and Human-Centric Computing. 2006:127-134.

[18] BETTENBURG N, PREMRAJ R, ZIMMERMANN T, *et al*. Extracting structural information from bug reports[C]//Proc of the 5th Working Conference on Mining Software Repositories. New York: ACM Press,2008:27-30.

[19] BETTENBURG N, PREMRAJ R, ZIMMERMANN T, *et al*. Duplicate bug reports considered harmful... really[C]//Proc of the 24th IEEE International Conference on Software Maintenance. 2008:337-345.

[20] BAEZA-YATES R, RIBEIRO-NETO B. Modern information retrieval [M]. New York: Addison Wesley, 1999.

[21] GUNN S R. Support vector machines for classification and regression [R]. [S.l.]: University of Southampton, Faculty of Engineering, Science and Mathematics; School of Electronics and Computer Science, 1998.

[22] RUNESON P, ALEXANDERSON M, NYHOLM O. Detection of duplicate defect reports using natural language processing[C]//Proc of ICSE. Washington DC: IEEE Computer Society,2007:499-510.

[23] 庞剑锋,卜东波, 白硕. 基于向量空间模型的文本自动分类系统的研究与实现[J]. 计算机应用研究,2001,18(9):23-26.

[24] ALLAN J, ASLAM J, BELKIN N, *et al*. Challenges in information retrieval and language modeling[J]. ACM SIGIR Forum, 2003,37(1):31-47..

(上接第 2126 页)

[12] MARCUELLO P, GONZÁLEZ A. Thread-spawning schemes for speculative multithreading[C]//Proc of the 8th International Symposium on High-performance Computer Architecture. Washington DC: IEEE Computer Society, 2002: 55-64.

[13] XEKALAKIS P, IOANNOU N, CINTRA M. Combining thread level speculation helper threads and runahead execution[C]//Proc of the 23rd International Conference on Supercomputing. New York: ACM Press,2009: 410-420.

[14] SACK P. SESC: SuperESCalar simulator [EB/OL]. (2004-12-20). http://iacoma.cs.uiue.edu/~pau/sack/sescdoc/sescdoc.pdf.

[15] MADRILES C, GARCÍA-QUIÑONES C, NCHEZ J. Mitosis: a speculative multithreaded processor based on precomputation slices[J]. IEEE Trans on Parallel and Distributed Systems, 2008, 19(7): 914-925.

[16] LIU Wei, TUCK J, CEZE L, *et al*. POSH: a TLS compiler that exploits program structure[C]//Proc of the 11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. New York: ACM Press,2006: 158-167.

[17] BHOWMIK A, FRANKLIN M. A general compiler framework for speculative multithreading[C]//Proc of the 14th Annual ACM Symposium on Parallel Algorithms and Architectures. New York: ACM Press, 2002: 99-108.

[18] 刘圆,安虹,汪芳,等. 利用连续两阶段在线剖析优化多线程推测执行[J]. 小型微型计算机系统,2009,30(3):385-390.

[19] DU Zhao-hui, LIM C C, LI Xiao-feng, *et al*. A cost-driven compilation framework for speculative parallelization of sequential programs [C]//Proc of ACM SIGPLAN Conference on Programming Language Design and Implementation. New York: ACM Press, 2004: 71-81.

[20] 安虹,王莉,王耀彬. 针对子程序结构的线程级推测并行性分析[J]. 小型微型计算机系统,2009,30(2):230-235.