

面向缺陷定位的代码搜索引擎

常建明^{1,2,3} 薄莉莉^{1,2,4} 孙小兵^{1,2,4}

- 1 扬州大学信息工程学院 江苏 扬州 225127
2 江苏省知识管理与智能服务工程研究中心 江苏 扬州 225127
3 东南大学软件学院 南京 211189
4 计算机软件新技术国家重点实验室(南京大学) 南京 210023
(carrycrebrith@163.com)

摘要 随着软件项目规模的扩大以及软件复杂性的增加,缺陷修复的难度越来越大。其中,绝大多数缺陷问题都是由代码的错误编写导致的,在软件缺陷修复过程中开发维护人员需要花费大量的时间定位并修改缺陷代码。针对这个问题,对缺陷报告以及对应的项目变更信息整合,根据代码抽象语法树结构信息计算代码块与缺陷报告之间的关系,从而构建缺陷-代码知识库。在此知识库的基础上构建面向缺陷定位的代码搜索引擎,以向用户推荐更全面的缺陷定位信息,包括相似缺陷报告、相关缺陷代码文件以及缺陷代码块,从而帮助开发和维护人员及时有效地定位缺陷。实验结果说明,相比现有的缺陷定位方法,所提方法能够更准确地定位缺陷代码文件,并且可有效定位到代码粒度。

关键词: 缺陷定位;缺陷报告;抽象语法树;代码搜索

中图法分类号 TP311

Code Search Engine for Bug Localization

CHANG Jian-ming^{1,2,3}, BO Li-li^{1,2,4} and SUN Xiao-bing^{1,2,4}

- 1 School of Information Engineering, Yangzhou University, Yangzhou, Jiangsu 225127, China
2 Jiangsu Engineering Research Center Knowledge Management and Intelligent Service, Yangzhou, Jiangsu 225127, China
3 School of Software, Southeast University, Nanjing 211189, China
4 State Key Lab. for Novel Software Technology, Nanjing University, Nanjing 210023, China

Abstract With the evolution and the increased complexity of software project, bug fixing is getting more difficult. During the bug fixing, developers need to spend a lot of time on bug localization and fixing. To evaluate this problem, this paper builds a bug-code database by integrating the bug reports and the corresponding evolution information, and analyzing the relationship between the code block and the bug report. Then, a code search engine is constructed for bug localization based on the bug-code database, which is used for recommending more comprehensive information about similar bug reports, bug related code files, and code blocks. The experiment results show that the proposed approach is more accurate to localize buggy files, and the localization can effectively reach code level.

Keywords Bug localization, Bug report, Abstract syntax tree, Code search

1 引言

软件缺陷是计算机程序或系统产生的一系列错误、瑕疵^[1]。在软件项目开发过程中,缺陷的产生不可避免,并且随

着软件需求的增大,软件缺陷的数量与日俱增,然而修复这些软件缺陷往往会消耗修复人员大量时间成本和人力成本。例如, Eclipse 项目开发过程中共有 145 000 条不同的缺陷报告。据统计,从 Eclipse 项目收集的 9 000 条缺陷报告中,修复单个

到稿日期:2020-11-29 返修日期:2021-04-09

基金项目:国家自然科学基金(61872312, 61972335, 62002309);南京大学计算机软件新技术国家重点实验室资助项目(KFKT2020B15, KFKT2020B16);扬州市校企合作项目(YZU201803);扬州大学“高端人才支持计划”(2019);江苏省“六大人才高峰”高层次人才项目(RJFW-053);江苏省“333”工程;扬州大学畜牧学学科特区学科交叉课题支持项目(yzuxk202015)

This work was supported by the National Natural Science Foundation of China(61872312, 61972335, 62002309), Open Funds of State Key Laboratory for Novel Software Technology of Nanjing University(KFKT2020B15, KFKT2020B16), Yangzhou City-Yangzhou University Science and Technology Cooperation Fund Project(YZU201803), Yangzhou University Top-level Talents Support Program(2019), Six Talent Peaks Project in Jiangsu Province(RJFW-053) and Jiangsu “333” Project and Yangzhou University Cross-disciplinary Project of Animal Science(yzuxk202015).

通信作者:薄莉莉(lilibo@yzu.edu.cn)

缺陷平均需要 86 天^[2]。

缺陷定位是缺陷修复的前提^[3-4],及时有效地发现并解决软件缺陷问题在软件开发维护过程中至关重要。但随着软件项目规模扩大和复杂度提高,缺陷定位的难度也越来越大。当缺陷修复人员接收缺陷报告进行修复时,即修改程序的部分代码,修复人员不仅需要具备丰富的开发经验,而且需要在阅读完大量的源代码文件后才有可能定位到缺陷的位置。因此,当一个软件项目规模较大时,修复人员往往会耗费大量的时间来定位软件缺陷。

为了帮助修复人员有效定位缺陷,减少缺陷修复时间,研究人员提出了一系列缺陷定位方法,其中主要分为基于信息检索的静态缺陷定位方法^[5-8]和动态缺陷定位方法^[9-11]。例如,Zhang 等^[12]构建了一种定位精度为方法级的静态定位模型 MethodLocator,该模型基于文档词向量^[13]的表示方法,将缺陷报告以及源代码中的函数内容用向量表示,并根据夹角余弦计算方法计算向量之间的相似度,进而定位到有缺陷的函数模块;Hoang 等^[14]结合基于信息检索模型的缺陷定位方法和基于程序频谱的缺陷定位方法^[15],提出了一种网络集群的多模式缺陷定位方法 NetML,通过优化失败的缺陷定位以及对缺陷报告和方法级别程序元素的聚类来实现有效的缺陷定位。然而,这些方法定位到的缺陷位置一般是代码文件和函数模块。当项目的规模较大时,对应的代码文件内代码规模也相对较大,而缺陷定位的精度如果仅限于文件和函数模块,将不利于修复人员阅读、理解相关缺陷代码。

为向缺陷修复人员提供更直观全面的缺陷定位信息,本文提出构建面向缺陷定位的代码搜索引擎。首先,通过整合缺陷报告以及对应代码的变更信息,构建两条缺陷报告间的关系以及缺陷报告与代码文件中缺陷代码块之间的关系,进而构建缺陷-代码知识库。接着,基于该缺陷-代码知识库,针对用户输入的缺陷描述,计算知识库中各个缺陷代码块导致用户描述缺陷的可能性,进而返回给用户最相关的缺陷代码块等缺陷定位信息。

2 研究背景

2.1 抽象语法树

抽象语法树(abstract syntax tree)^[16]是源代码语法结构的树状形式,树中的每一个节点都表示源代码的结构,因其不会表示出语法的每一个细节,所以称其为抽象语法树。

抽象语法树是通过对代码进行词法分析、语法分析和语义分析形成的结构,这样的语法结构并不依赖于原语言的文法和细节,因此借助抽象语法树能够清晰表示程序代码的语法信息。目前,将程序转换成抽象语法树的开源工具有很多,如 Eclipse 中的程序开发工具包 JDT,ANTLR 以及基于 ANTLR 的 gumtree^[17]等。

抽象语法树上节点之间的联系包含代码之间的结构关系,本文为了计算代码块对相关缺陷报告的重要性,以代码抽象语法树结构为基础构建对应代码文件的代码块关系图。该代码块关系图利用对应代码文件抽象语法树结构,能够很好

地表示代码文件内代码块之间的结构联系,并有助于计算出关系图中各个节点对应代码块的重要性。

2.2 中心性度量

中心性^[18]是表示单个节点在网络中处于中心地位的程度,节点中心性越高,其在网络中就越重要。目前有 4 种衡量节点中心性的指标,分别为度中心性、紧密中心性、间接中心性以及特征向量中心性。

紧密中心性(Closeness Centrality, CC)为图中各节点到其他节点的最短路径之和的倒数,表示图中节点到其他节点的难易程度。节点紧密中心性值越大,其在图中的位置越靠近中心点,其重要性也就越大。当节点与其他节点之间不存在路径时,本文规定该节点与其他节点的路径长度为无穷大,该节点对应的紧密中心性值为 0,表明其远离中心点。

间接中心性(Betweenness Centrality, BC)又称中介中心性,为相同端节点内经过该节点的最短路径总数除以该段节点内最短路径总数的比率。间接中心性表示图中该节点对其他节点的传播控制能力,节点间接中心性值越大,表明经过该节点的最短路径越多,其重要性就越大。

本文构造代码块关系图来表示单个代码文件内修改代码块之间的关系,关系图中的节点表示对应修改代码块。结合代码块关系图结构,本文选取紧密中心性和间接中心性作为衡量图中节点重要性的指标。在代码块关系图中,节点对应的中心值越高,表示该节点对应修改的代码块与其他修改代码块之间的关联越多,则缺陷的产生与该修改代码块之间的相关性就越大。

3 面向缺陷定位的代码搜索引擎实现

3.1 实现框架

图 1 给出了面向缺陷定位代码搜索引擎实现的整体流程,其主要分为两个部分:缺陷-代码知识库的构建以及面向缺陷定位的代码搜索。在知识库构建部分,本文整理 Bugzilla 平台上项目的缺陷报告信息以及缺陷报告对应代码的变更信息;接着,对缺陷报告文本内容进行自然语言预处理,同时提取源代码文件中的修改代码块并构建源代码文件抽象语法树,其中本文规定代码文件中连续修改的代码行为一个代码块,若只修改一行代码,则对应代码块表示该行代码;再利用余弦相似度计算缺陷报告文本之间的相似度,在源代码文件抽象语法树的基础上构建代码块关系图并计算图中各个节点对应代码块的重要性,进而根据缺陷报告与缺陷报告之间的相似度关系以及缺陷报告与代码块的重要性关系,构建缺陷-代码知识库。面向缺陷定位的代码搜索部分,当用户输入缺陷描述信息后,引擎对用户输入的描述信息进行自然语言预处理,并将其与缺陷-代码知识库中的缺陷报告文本进行匹配。根据文本之间的余弦相似度返回相似缺陷报告列表,并根据用户输入的缺陷描述与知识库中缺陷报告之间的相似度以及该缺陷报告对应代码块的重要性,计算知识库中缺陷代码块对于用户输入缺陷描述内容而含有缺陷的可能性,返回相关缺陷定位信息,包括缺陷代码块、缺陷代码文件,

以实现缺陷定位。

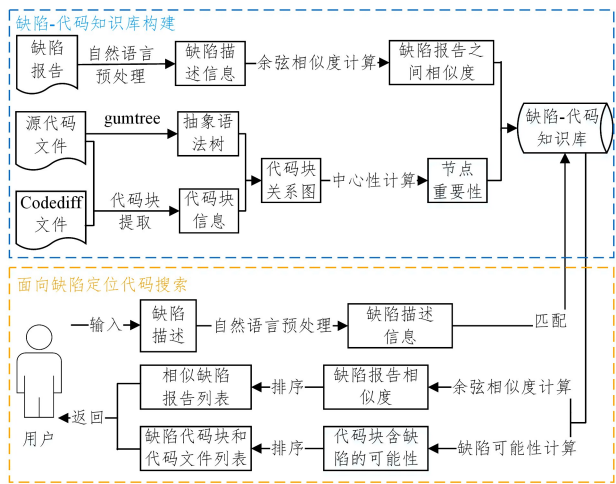


图1 面向缺陷定位代码搜索引擎的实现流程图
Fig.1 Implementation of code search engine for bug localization

3.2 缺陷代码知识库构建

本节介绍了面向缺陷定位的代码搜索引擎实现的第一阶段内容,即缺陷-代码知识库构建过程,主要包括数据预处理、知识库中的实体关系构建、知识库构建 3 个步骤。

3.2.1 数据预处理

数据预处理主要是提取知识库构建数据,依次分为数据获取、json 格式抽象语法树构建以及缺陷代码块提取 3 个部分。

(1)数据获取部分。通过从 Bugzilla 缺陷管理平台以及 Github 项目仓库中分别爬取缺陷报告信息以及缺陷报告对应的代码变更信息,构建缺陷-代码知识库。其中缺陷报告信息包括缺陷报告 ID、缺陷总结、缺陷描述和评论信息,代码变更信息包括与缺陷报告相关的 codediff 文件以及 codediff 文件对应的修改前 java 代码文件。由于缺陷报告信息中往往包含缺陷产生的原因、解决办法等重要信息,为了能够有效利用这些信息,本文使用 nltk 工具包^[19]对缺陷报告文本进行小写处理、分词、去除停用词、词干化等自然语言预处理,统一缺陷报告的文本格式,其中停用词选取 nltk 提供的 179 个常用英文停用词。

(2)json 格式抽象语法树构建部分。为利用抽象语法树中的源代码语法结构建立代码块关系图,在数据预处理过程中,使用 gumtree 作为抽象语法树转换工具,将 java 代码文件转换成对应 json 数据格式抽象语法树。以图 2(a)所示的代码为例,其对应 json 格式抽象语法树的部分结构如图 2(b)所示。其中,语法树中各个节点都有“type”“typeLabel”“pos”“length”“children”属性,部分节点还含有“label”属性。“children”属性是其对应的嵌套孩子节点信息,为空则表明该节点为叶子节点;“pos”属性是语法树节点在源代码文件中的字符位置;“label”属性表示该节点对应的代码内容,包括变量名、注释信息等,它与“length”属性配对;“length”属性表明对应代码内容的长度;“typeLabel”属性表示对应的代码行、函数模块、类的结构信息。

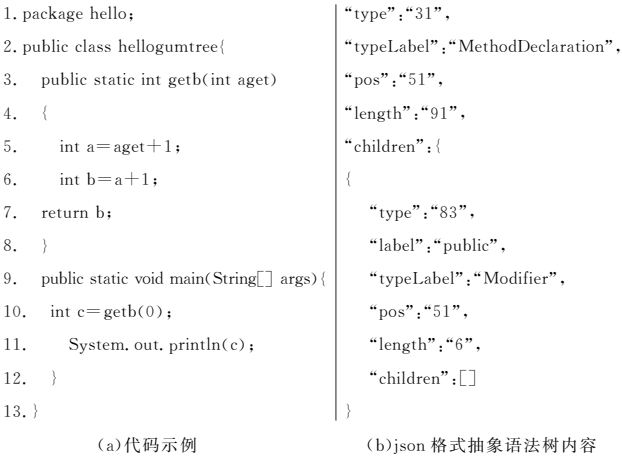


图2 代码转为 json 格式抽象语法树的示例
Fig.2 Example of converting code to abstract syntax tree

(3)缺陷代码块提取部分。本文利用 codediff 文件提取 java 代码文件中删减的代码块信息,并将其作为缺陷-代码知识库中的实体。提取的代码块信息包括代码内容以及所属函数名,其中函数名的获取是通过代码块找到对应抽象语法树节点,再根据“pos”以及“typeLabel”节点属性值找到该节点所属函数对应的定义节点,从而获取相关函数名,函数名获取操作如算法 1 所示。

算法 1 代码块所属函数名获取算法

- 输入:代码块;代码文件;代码文件对应的 json 格式抽象语法树
输出:代码块所属函数模块的函数名
- 1.将代码块 chunk 以及代码文件 codefile 当作字符串处理,获得子串 chunk 在母串代码文件 codefile 中的位置 pos(该 pos 值与抽象语法树各个节点的“pos”属性表达含义相同)。
 - 2.遍历 json 格式抽象语法树,定位到属性“pos”值为步骤 1 中 pos 值的节点区域。
 - 3.遍历步骤 2 中的节点区域,判断其中是否存在“typeLabel”属性值为“MethodDeclaration”的节点,若存在,则遍历该节点的孩子节点,找到孩子节点列表中“typeLabel”属性值为“SimpleName”的节点,并返回该节点的“label”属性值,即函数名。
 - 4.若不存在“typeLabel”属性值为“MethodDeclaration”的节点,则向上遍历 json 格式抽象语法树,直至找到“typeLabel”属性值为“MethodDeclaration”的节点,并同样执行步骤 3 中的函数名获取操作。若遍历到根节点,则函数名为空。

3.2.2 知识库中的实体关系构建

在缺陷-代码知识库中,实体为缺陷报告和缺陷代码块,实体间的关系为缺陷报告之间的相似度联系以及缺陷报告与缺陷代码之间的重要性关系。其中,参照 Zhang 等^[12]采用余弦相似度计算方法计算缺陷报告和源代码方法之间的相似度,本文通过计算预处理后缺陷报告文本信息之间的余弦相似度来表示缺陷报告之间的相似度关系,余弦相似度的计算式如下:

$$\cos \theta = \frac{\sum_{i=1}^n (A_i \times B_i)}{\sqrt{\sum_{i=1}^n (A_i)^2} \times \sqrt{\sum_{i=1}^n (B_i)^2}} = \frac{\mathbf{A} \cdot \mathbf{B}}{|\mathbf{A}| \times |\mathbf{B}|} \quad (1)$$

其中, $\mathbf{A}$ 和 $\mathbf{B}$ 表示两个文本的 n 维词向量, A_i 和 B_i 分别表示 $\mathbf{A}$ 和 $\mathbf{B}$ 词向量中的元素。

缺陷代码块与缺陷报告之间的重要性关系表示该代码块对相关缺陷报告的重要程度,该重要性值计算主要根据缺陷报告对应的缺陷代码文件语法结构,构建代码文件中各个代码块的关系图,从而计算所有代码块对相关缺陷报告的重要性,具体过程如算法 2 所示,主要分为以下 3 个步骤:抽象语法树构建、代码块关系图构建和代码块重要性值计算。

算法 2 缺陷代码块重要性值衡量算法

输入:缺陷报告;缺陷报告相关缺陷代码文件

输出:代码文件中各个缺陷代码块对该缺陷报告的重要性

1. 依次遍历缺陷报告 bugReport 的相关缺陷代码文件 `codefile(i)` ($i \in \text{sum}(\text{codefiles})$)。
2. 获取数据预处理部分代码文件 `codefile(i)` 对应的 json 格式抽象语法树以及 `codefile(i)` 中所有修改的代码块 `chunk`, 若代码块 `chunk` 数量 `sum(chunk)` 不为 0, 则生成 json 格式抽象语法树树状图 `AST(i)`, 否则处理下一个代码文件。

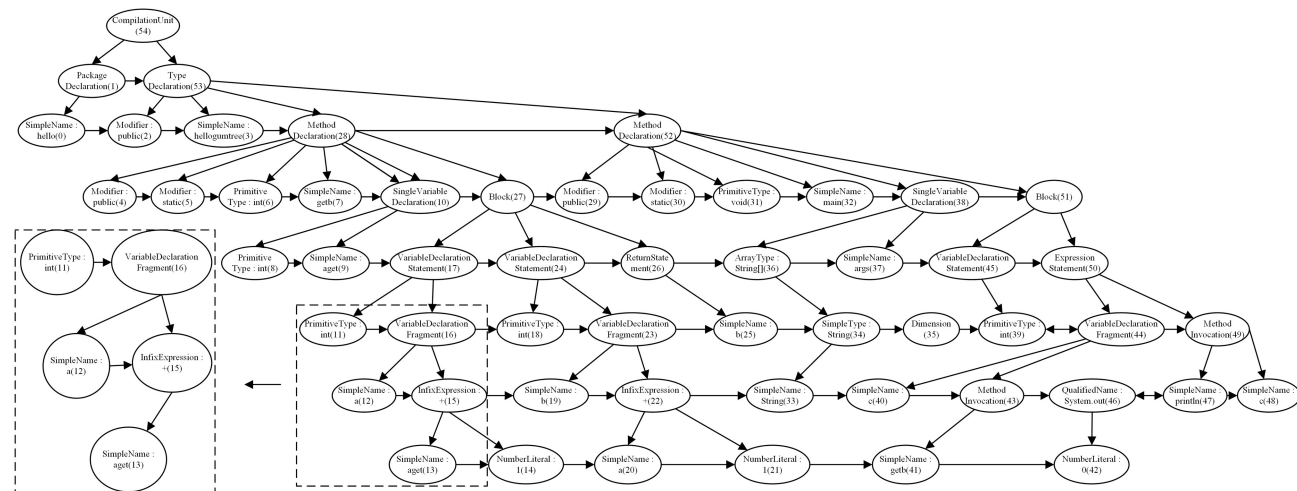


图 3 代码抽象语法树

Fig. 3 Code abstract syntax tree

(2)代码块关系图的构建过程。将代码块以及代码文件中的整个代码内容当作字符串处理,定位子串(代码块)在母串(代码文件)的位置,将位置的数值与抽象语法树中节点的“pos”属性相匹配,进而找到代码块在抽象语法树中的位置区域,根据“label”和“typeLabel”属性进一步判断代码块在抽象语法树的具体位置。并在新建抽象语法树的基础上依次遍历其子树结构和兄弟节点,判断其他代码块节点是否存在父子或兄弟关系,进而构建代码块关系图,代码块关系图的具体构建过程如算法 3 所示。以图 2 所示的代码为例,假设修改代码行为 3,6,7,9,10,根据本文对代码块的定义,代码块 1 为第 3 行,代码块 2 为第 6 行和第 7 行,代码块 3 为第 9 行和第 10 行,在图 3 所示的语法树结构基础上形成的代码块关系如图 4 所示。

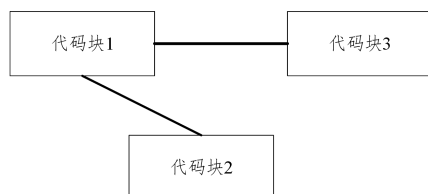


图 4 代码块关系图

Fig. 4 Relational graph of code chain

3. 遍历 $AST(i)$, 依次找到各个代码块 $chunk(j) (j \in \text{sum}(chunk))$ 在抽象语法树 $AST(i)$ 中对应的子树 $subAST(j)$, 并保留子树 $subAST(j)$ 的位置, 利用抽象语法树 $AST(i)$ 树状结构生成新的代码块 $chunk$ 之间的关系图, 图中节点 $node(j)$ 对应 $chunk(j)$ 。
4. 遍历代码块关系图, 计算各个节点 $node(j)$ 的紧密中心性 $CC(j)$ 和间接中心性 $BC(j)$, 以 $CC(j)$ 和 $BC(j)$ 的和值 $\text{sum}(CC(j), BC(j))$ 来表示对应 $chunk(i)$ 对于该缺陷报告 $bugReport$ 的重要性值。

(1) 抽象语法树的构建过程。由于代码模块化、低耦合的特点, 抽象语法树结构的层次不会超过 10 层, 这样代码块对应节点之间的父子关系并不明显, 因此最后形成的代码块关系图为互不关联的点图, 但本文在已有 json 格式抽象语法树的基础上添加“brother”节点属性, 即将抽象语法树中同一层次的节点从左至右赋予兄弟关系, 进而重新遍历以构建新的树图结构, 如图 2 所示的代码对应的抽象语法树如图 3 所示。

算法3 代码块关系图构建算法

输入:代码文件 codefile,代码文件对应的抽象语法树 AST,代码文件中修改代码块的列表 chunkList

输出:代码块关系图

1. 依次遍历代码块列表 chunkList 中各个代码块 chunk(i) ($i < \text{len}(\text{chunkList})$)。
2. 将代码块 chunk(i) 以及代码文件 codefile 当作字符串处理, 获得字符串 chunk(i) 在母串代码文件 codefile 中的位置 pos (该 pos 值与抽象语法树各个节点的“pos”属性表达含义相同)。
3. 遍历抽象语法树 AST, 根据步骤 2 获得 pos 值找到在抽象语法树 AST 中的位置区域, 再结合代码块 chunk(i) 内容与区域内节点的“label”和“typeLabel”属性值获得 chunk(i) 在抽象语法树 AST 中的具体位置节点 node(i)。
4. 直到遍历完 chunkList, 获得所有 chunk(i) 在 AST 上的具体位置节点 node(i)。
5. 新建一个空的关系图。
6. 遍历抽象语法树 AST, 找到树中与 chunk(i) 相关的节点 node(i)。
7. 依次向下深度遍历 node(i) 的分支, 判断分支中是否存在其他代码块 chunk(j) 对应的 node(j), 若存在, 则在新的空的关系图上, 在 chunk(i) 与代码块 chunk(j) 之间构建父子关系, 无向相连并向上回溯, 继续深度遍历其他分支, 直到遍历完 node(i) 的所有分支。
8. 向右遍历 node(i) 的所有兄弟节点, 并判断兄弟节点中是否存在其

他代码块 chunk(k)对应的 node(k),若存在,则在新的空的关系图上,在其之间构建兄弟关系,无向相连,并退出 node(i)的向右遍历。

9. 重复步骤 6—步骤 8,直到所有 node 节点遍历完,生成一个新的代码块关系图,图中各个节点 node(i)对应相关代码块 chunk(i)。

(3)代码块重要性值的计算过程。在已有代码块关系图的基础上,根据图中各个节点的中心性值来表示节点对应代码块的重要性,本文选取紧密中心性(CC)和间接中心性(BC)作为节点中心性衡量指标,以归一化处理后 CC 和 BC 的和值来表示对应代码块的重要性。其中 CC 和 BC 的计算式如式(2)和式(3)所示。以图 4 所示的构建的代码块关系图为例,代码块 1、代码块 2、代码块 3 对应的归一化后的 CC 值为(0.43,0.29,0.29),归一化后的 BC 值为(1,0,0),根据本文的重要性计算方法,代码块对相关缺陷报告的重要性值为(0.48,0.26,0.26)。

$$CC(v)=\frac{1}{\sum_{s\neq v\in V}dis(v,s)}\tag{2}$$

其中,dis(v,s)表示节点 v 与节点 s 之间的最短路径长度。

$$BC(v)=\sum_{C_i\neq C_j\neq v\in V}\frac{\sigma_{C_iC_j}(v)}{\sigma_{C_iC_j}}\tag{3}$$

其中,BC(v)表示 v 节点的 BC 参数值;σ_{C_iC_j}表示端节点 i,j 之间的最短路径数目;σ_{C_iC_j}(v)表示端节点 i,j 之间经过节点 v 的最短路径数目之和。

3.2.3 缺陷-代码知识库构建

本文选取 neo4j 图形知识库^[20]构建缺陷-代码知识库,通过爬取的 5000 条 bug 报告,将 20000 多个代码文件作为知识库构建数据集。知识库实体为缺陷报告和代码块,并且缺陷报告以 bugID 命名;代码块以“pos in bugID_代码文件_函数模块”格式命名,其中“pos”表示代码块在源代码文件中的位置,“代码文件”表示代码块所属的代码文件,“函数模块”表示代码所属函数名。缺陷报告实体除标签“name”外还包含标签“content”,用于存储对应缺陷报告的描述信息;代码块实体包含标签“content”,用于存储代码块内容。本文为优化缺陷报告之间的联系,将在相似度高的缺陷报告之间建立相似关系,其中相似度阈值为 0.7;缺陷报告与代码块之间为重要性关系,且缺陷报告与代码块之间的重要性关系为一对多关系,即多个代码块对应着唯一的缺陷报告。

3.3 缺陷定位

代码搜索引擎实现的第二个阶段为缺陷定位,首先搜索引擎通过对输入的缺陷描述信息进行自然语言预处理,包括去停用词、词干化以及分词处理,统一描述信息与知识库中缺陷报告的文本格式;接着将预处理后的缺陷描述作为在缺陷-代码知识库中进行查询的依据,匹配知识库中的缺陷报告并计算与其之间的相似度,根据相似度大小排序推荐相似缺陷报告列表。此外,本文结合缺陷描述与缺陷报告之间的相似度以及对代码块的重要性值,计算了知识库中各个代码块对于用户输入的缺陷描述而含有缺陷的可能性 degree,并根据 degree 值排序返回缺陷代码块列表,再根据该代码块所属代码文件推荐缺陷代码文件列表,以辅助用户定位多个缺陷。代码块 degree 值的计算式如下:

$$degree(i,j)=\alpha * sim(i)+\beta * imp(i,j)$$

$$i\in sum,j\in sum(i)$$

其中,i 表示知识库中缺陷报告的位置;j 表示知识库中与第 i

条缺陷报告直接相关的缺陷代码块的位置;degree(i,j)表示针对用户输入的缺陷描述,第 i 条缺陷报告相关的代码块中第 j 条缺陷代码块含有缺陷的可能性;sim(i)值表示知识库中第 i 条缺陷报告与用户输入缺陷描述之间的余弦相似度;imp(i,j)表示知识库中第 i 条缺陷报告相关的第 j 条缺陷代码块重要性,即该代码块对于第 i 条缺陷报告的重要程度;α 和 β 分别为 sim 和 imp 的系数,本文通过实验来确定 α 和 β 的取值;sum 表示知识库中缺陷报告的数量;sum(i)表示知识库中与第 i 条缺陷报告相关的缺陷代码块数量。

4 实验评估

4.1 实验问题

为达到以上目的,本文设计了以下 3 个实验问题进行研究。

RQ1 改变缺陷信息推荐算法 degree 中 α 和 β 系数大小对引擎定位缺陷代码文件性能的影响?

RQ2 相比 VSM 和 bugLocator 缺陷定位工具,面向缺陷定位代码搜索引擎定位缺陷代码文件是否更准确?

RQ3 面向缺陷定位代码搜索引擎推荐的代码块信息是否有用?

4.2 数据集

实验数据如表 1 所列,实验从 Bugzilla 以及 Github 中爬取 Firefox for Android,AspectJ,SWT 这 3 个项目的缺陷报告、对应的项目变更信息,共有 5000 多条缺陷报告,20000 多个修改代码文件。其中,SWT 项目的缺陷报告数量和变更代码文件数量最多,分别为 4100 条缺陷报告和 15526 个变更代码文件。

表 1 实验数据

Table 1 Experimental data

项目	缺陷报告数量	代码更改文件数量
Firefox for Android	600	2006
AspectJ	500	2960
SWT	4100	15526

4.3 实验评估标准

为了衡量本文提出的代码搜索引擎的有效性,本文选取 topN、sumN、平均精度值 MAP(Mean Average Precision)作为衡量标准。

topN 指在返回结果数为 N 时,被正确定位缺陷的缺陷报告数量占总报告数的比例,一般来说,当缺陷定位返回的 N 个结果中有一个结果正确,则规定该定位是成功的。

sumN 在本文实验中指在返回结果数为 N 时,所有缺陷报告中被正确定位的缺陷代码文件总数,如分别对 reportA 和 reportB 进行缺陷定位,N 条结果中正确定位的缺陷文件数量为 2 和 1,则该 sumN 为 3。

MAP 是用来衡量排序结果优劣的度量标准。本文 MAP 的计算方法如式(5)所示,表示所有缺陷报告的平均正确率 AP(Average Precision)的平均值。

$$MAP=\frac{sum(AP)}{N}$$

(5)

其中,N 为缺陷报告的数量,sum(AP)为所有缺陷报告的 AP 值的和,而单个缺陷报告 AP 的计算式如下:

$$AP=\frac{\sum_{k=1}^N P(k)\times pos(k)}{\sum_{k=1}^N pos(k)}\tag{6}$$

其中, k 表示结果列表中结果的位置; N 指结果列表中结果的总数; $P(k)$ 指 k 个结果中的正确定位率,其计算方法如式(7)所示; $pos(k)$ 表示第 k 个结果是否为正确定位的结果。

$$p(k)=\frac{sum(k)}{k}\tag{7}$$

其中, $sum(k)$ 表示在结果列表中, k 位置之前正确定位的结果数量。

4.4 实验设置

为了回答 RQ1,实验采取控制变量法,控制 3 个项目的对应引擎规模以及测试集数量一致,探究 α 和 β 的改变对引擎性能的影响。从而按照本文爬取数据,分别将 AspectJ, Firefox for Android, SWT 项目中较旧的 400 条缺陷报告以及对代码变更数据作为代码搜索引擎的构建数据,按照 10% 的比例,选取 40 条较新的缺陷报告为测试集,通过改变 α 和 β 在 $(0,1)$ 区间的取值(其中 $\alpha+\beta=1$)来比较 40 条测试缺陷报告结果中 topN 和 sumN 的数值。进而分析 α 和 β 取值的变化对本文设计的代码搜索引擎性能的影响。

RQ2 是为了分析本文提出的面向缺陷定位代码搜索引擎对缺陷代码文件的定位效果,因此在 RQ1 的实验情况下,本次实验中引擎选取 (α,β) 的最优解,将 VSM^[21] 以及 RVSM (BugLocator)^[22] 作为对比工具,分析代码搜索引擎与其他工具对于定位缺陷代码文件的效率。实验控制项目的规模一致,选取 AspectJ, Firefox for Android, SWT 项目中较旧的 400 条缺陷报告作为训练数据,按照 10% 的比例,选取各个项目中 40 条较新的缺陷报告作为测试数据,根据 40 条测试报告的结果中的 topN 和 MAP 数值来比较各个工具缺陷代码文件定位的准确性以及缺陷代码文件列表的排序效果。

为了回答 RQ3,实验从缺陷代码块定位的准确性以及推荐缺陷代码块与实际修改代码块之间的相关性来分析代码搜索引擎中缺陷代码块推荐功能的实用性。在实验中,根据引擎推荐缺陷代码块名称“pos in bugID_代码文件_函数模块”来获取缺陷代码文件以及缺陷函数名,通过判断推荐缺陷代码块所属函数模块是否与实际缺陷函数模块一致来分析缺陷代码块的定位准确性;通过比较推荐缺陷代码块与实际修改代码块之间的相似度以及通过人工分析来分析推荐缺陷代码块的相关性。

4.5 实验结果分析

在回答 RQ1 问题的实验中,改变 α 和 β 的取值时,3 个项目对应代码引擎 topN 和 sumN 指标的变化如图 5 所示。图 5 中,绿色、黄色、红色折线分别为依据 Firefox for Andriod, AspectJ, SWT 项目构建的代码搜索引擎。其中,图 5(a)、图 5(c)给出了 α 和 β 变换下,40 条测试缺陷报告中代码引擎 topN 数量的变化,图中横坐标为 α 在 $(0,1)$ 区间内的取值,纵坐标分别为代码搜索引擎 top5 和 top10 的数值,根据图中的实验结果可知,随着 α 的增大,代码搜索引擎对应的 top5 和 top10 变化趋势为“不变→增加→下降”,并在 $(0.6,0.9)$ 区间内的达到峰值;图 5(b)、图 5(d)给出了 α 和 β 变换下,引擎返

回的 $40 * N$ 个缺陷代码文件中,被成功定位的缺陷代码文件数量的变化,图中横坐标表示 α 在 $(0,1)$ 区间内取的取值,纵坐标分别表示代码搜索引擎 sum5 和 sum10 的数值,根据图中的实验结果可知,随着 α 增大,代码搜索引擎对应 sum5 和 sum10 的变化趋势为“不变→增加→下降”,并在 $(0.7,1)$ 区间内达到峰值。

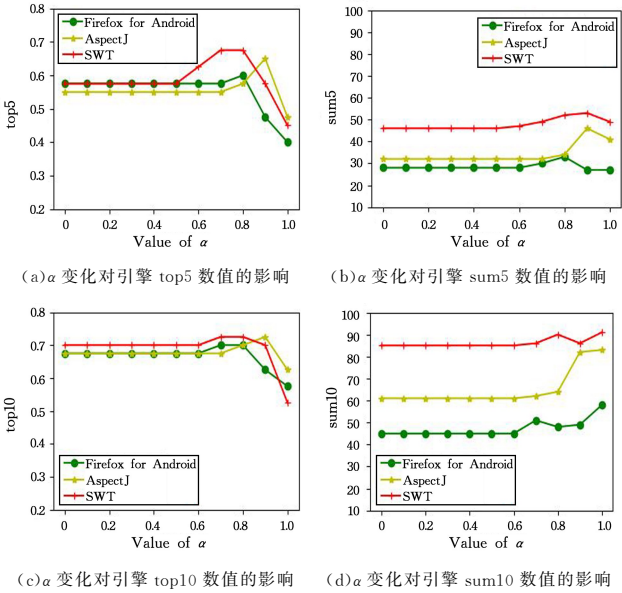


图 5 代码搜索引擎性能变化图(电子版为彩色)

Fig. 5 Change of code search engine performance

分析实验结果可得:由于缺陷代码块之间的重要性差值很小,且重要性高的代码块数量很多,当 α 取值小于 0.5、重要性值比重较大时,引擎推荐结果不会被改变且性能较低;当 α 取值大于 0.5 时,代码块重要性值的比重降低,即主要依赖于缺陷描述与缺陷报告间的相似度时,代码搜索引擎的性能较好。当 (α,β) 取 $(0.8,0.2)$ 时,引擎对应的实验结果如表 2 所列。

表 2 (α,β) 为 $(0.8,0.2)$ 时代码搜索引擎的实验结果
Table 2 Experiment results of code search engine when (α,β) is $(0.8,0.2)$

项目	top5	sum5	top10	sum10
Firefox for Android	24(60%)	33	28(70%)	48
AspectJ	23(57.5%)	34	28(70%)	64
SWT	27(67.5%)	52	29(72.5%)	90

结合表 2 及图 5 数据可知,根据 Firefox for Android 构建的代码搜索引擎 top5 为峰值 60%, top10 为峰值 70%, sum5 为峰值 33;根据 SWT 项目构建的代码搜索引擎 top5 为峰值 67.5%, top10 为峰值 72.5%, sum10 为峰值 90;针对根据 AspectJ 项目构建的引擎,虽然实验测试指标皆未取得最高值,但是对应 top5、top10 值的大小排序为第 2,由 Firefox for Android 以及 SWT 项目构建的代码搜索引擎性能相对最优。根据以上实验数据,当 (α,β) 取 $(0.8,0.2)$ 时,引擎性能相对最优,此外,根据图中折线数据的显示,相比 α 取 0.8,当 (α,β) 取 $(0.7,0.3)$ 时,3 个项目对应代码搜索引擎的性能都较优且稳定。实验结果表明,当 α 在 $(0.7,0.8)$ 区间内时 3 个项目对应的引擎性能相对较好。

代码搜索引擎 VSM 以及 BugLocator 对于 3 个项目数据的缺陷定位结果如表 3 所列。

表 3 本文方法与 VSM, BugLocator 方法定位效果的比较结果
Table 3 Comparison results with VSM and BugLocator methods

项目	测量标准	VSM	BugLocator	本文方法
Firefox for Android	top5	7(17.5%)	10(25%)	24(60%)
	MAP5	0.02	0.05	0.05
	top10	10(25%)	13(32.5%)	28(70%)
	MAP10	0.04	0.07	0.06
AspectJ	top5	8(20%)	12(30%)	23(57.5%)
	MAP5	0.02	0.06	0.06
	top10	11(27.5%)	18(45%)	28(70%)
	MAP10	0.06	0.12	0.09
SWT	top5	12(30%)	13(32.5%)	27(67.5%)
	MAP5	0.03	0.06	0.10
	top10	15(37.5%)	20(50%)	29(72.5%)
	MAP10	0.06	0.12	0.17

本文构建的代码搜索引擎在 Firefox for Android 项目的 40 个测试数据中, top5 和 top10 的数值最高, 分别比 BugLocator 高出 0.35 和 0.375, 且 MAPN 数值相对较高; 在 AspectJ 项目的 40 个测试数据中, top5 和 top10 的数值最高, 分别比 BugLocator 高出 0.275 和 0.25, 且 MAPN 数值相对较高; 在 SWT 项目的 40 个测试数据中, top5 和 top10 数值最高, 分别比 BugLocator 高出 0.35 和 0.225, 同样 MAP5 和 MAP10 的数值也最高, 分别比 BugLocator 高出 0.04 和 0.05。分析实验结果可知, 尽管我们在构建缺陷-代码知识库的过程中, 未考虑无效、重复、错误的缺陷报告等情况, 但相比 VSM 和 BugLocator, 本文提出的面向缺陷代码的搜索引擎在定位缺陷代码文件时更准确, 引擎返回的结果列表排序也相对合理。

在回答 RQ3 问题实验中, 本文向代码搜索引擎输入 10

条最新的缺陷报告描述信息, 将其中正确定位到函数模块的缺陷定位结果整理成表 4, 其中缺陷报告列为 10 条缺陷报告中被成功定位缺陷函数的报告 ID; 缺陷代码文件列为引擎成功定位出的缺陷代码文件名; 函数列为引擎成功定位出缺陷代码文件中含有缺陷的函数名称; 相似度列表示引擎返回的缺陷代码与实际对应函数内修改代码之间的余弦相似度。

表 4 引擎搜索结果
Table 4 Engine search results

缺陷报告	缺陷代码文件	函数	相似度
407479	Tracker	open	0.41
407555	CTabFolder	updateBkImages; computeControlBounds	0.77; 0.06
413200	Scrollable	gtk_scroll_event	0.11

本文的代码搜索引擎返回报告相关缺陷代码块与实际修改代码块的对照如表 5 所列, 通过相似度计算、缺陷代码块所属函数路径判断以及人工判断来分析成功定位的缺陷代码块数量。例如, 分析缺陷报告 407479 返回结果, 缺陷代码块位置 "Tracker.java\open()" 被成功定位出来, 并且人工分析两个代码块, 根据其有相同的函数调用、参数数量来判断两者之间是相关的; 分析缺陷报告 413200 返回结果, 虽然缺陷代码块位置定位正确, 由于 if 判断条件相差较大, 代码块本身与实际修改代码之间并无相关性。

分析引擎返回结果可知, 在 10 条测试缺陷报告中: 引擎成功定位缺陷代码块所属函数位置数量为 4; 成功定位出缺陷代码块的数量为 2。从以上实验结果得出, 本文构建的面向缺陷定位代码搜索引擎能够定位出相关的缺陷代码块, 向用户提供的缺陷代码块信息能够帮助用户定位缺陷, 减少缺陷维护成本。

表 5 缺陷代码块的结果分析对照表
Table 5 Analysis of buggy code

缺陷报告	引擎返回缺陷代码块	实际修改代码块
407479	OS. SetLayeredWindowAttributes (hwndOpaque, 0xFFFFFFFF, (byte)0xFF, OS. LWA_COLORKEY OS. LWA_ALPHA);	OS. SetLayeredWindowAttributes(hwndTransparent, 0, (byte) 0x01, OS. LWA_ALPHA);
407555	bounds, width=10; bounds, y=-bounds, y; bounds, height-=2* bounds, y-1	bounds, width=10; bounds, y=-bounds, y; bounds, height-=2* bounds, y-1; bounds, x=0;if(controlBkImages[i] != null)
407555	ctrlSize= controls[i]. computeSize (bodyWidth, SWT. DEFAULT);	int y=-bodyTrim, y; rects[i]. y=y;
413200	if(scrollBar !=null &&. ! OS. GTK_WIDGET_VISIBLE(scrollBar, handle) &&. scrollbar, getEnabled())	if(OS. GTK_VERSION < OS. VERSION(2,8,0)) { hBarHandle= OS. GTK_SCROLLED_WINDOW_HSCROLLBAR (scrolledHandle);} else { hBarHandle= OS. gtk_scrolled_window_get_hscrollbar (scrolledHandle);}

5 相关工作

5.1 知识库构建

在软件项目开发和维护过程中, 研究人员常设计一些知识库来管理、修复缺陷。例如, Wang 等^[23]整合缺陷报告、commit 信息及其之间的联系, 构建基于终身主题模型的缺陷报告、commit 信息的知识图谱; Ye 等^[24]整合项目开发先验知识, 从 Bugzilla 缺陷报告平台和 Github 项目仓库中获取源代

码函数模块、组件 API、缺陷修复历史、代码变更信息以及源文件之间的依赖关系, 构建细粒度基准知识库; Youm 等^[25]从 Bugzilla 平台和 Github 项目仓库中获取缺陷报告文本信息、代码文件结构信息以及代码变更信息, 进而构建缺陷知识库。相比以代码文件和函数模块为处理单位, 本文以更精确的代码块为实体, 设计出缺陷代码块对相关缺陷报告重要性的计算方法, 以构建缺陷代码块与缺陷报告之间的重要性联系, 进而构建更为全面实用的缺陷-代码知识库。

5.2 缺陷定位技术

目前缺陷定位技术主要分为基于信息检索的静态缺陷定位技术^[5-8]和动态缺陷定位技术^[9-11]。

静态定位技术根据缺陷报告与代码文件的文本相似性定位到相关代码文件。例如,Zhang等^[12]构建一种定位精度为方法级的静态定位模型 MethodLocator,该模型基于文档词向量^[13]的表示方法,将缺陷报告以及源代码中的函数内容以向量表示,并根据夹角余弦计算方法计算缺陷报告内容和函数代码之间的相似度,进而定位到有缺陷的函数模块;Wang等^[26]根据监督模型,针对修复历史的实用性、bug报告与源代码文件内关键词的相关性、规模大的代码文件含有缺陷的可疑性这3个特征,提出了一种基于监督主题模型的定位方法,该方法利用缺陷的修复历史以及缺陷报告与代码文件的文本相似性以及语义相似性来有效定位到相关代码文件。

动态定位依赖于程序插桩^[27]和程序运行跟踪^[28]等来实时定位缺陷。例如,Ding等^[29]借助程序信息和测试信息,提出了一种基于事件信息量的缺陷定位方法 SIQ,该方法结合语句执行信息分析测试信息中事件的类型及发生概率,进而动态计算调整缺陷的定位结果,实现缺陷定位;Shu等^[30]通过统计学中的条件概率思想,构建量化程序频谱与执行结果之间的关系强弱模型,进而依赖程序频谱信息提出一种新的基于条件概率模型的缺陷定位方法;Hoang等^[14]结合基于信息检索模型的缺陷定位方法和基于程序频谱的缺陷定位方法^[15],提出了一种网络集群的多模式缺陷定位方法 NetML。相比提供代码文件级别和函数级别的缺陷定位信息,本文提出缺陷可能性计算方法 degree,并计算知识库中代码块对用户输入的缺陷描述而含有缺陷的可能性值,进而返回相关的代码块级别的缺陷定位信息。

结束语 本文构建了一个新的缺陷-代码知识库,并在该知识库的基础上实现了面向缺陷定位的代码搜索引擎。实验结果表明,当缺陷信息推荐依据 degree 中的系数(α, β)为(0.8, 0.2)时,本文设计的代码搜索引擎的性能最优,相比已有的缺陷定位工具 BugLocator 和 VSM,该引擎的搜索结果更准确,并且推荐的缺陷定位信息能够帮助修复人员有效定位到代码块级别的缺陷。

但是,本文构建的面向缺陷定位的代码搜索引擎依然存在一些问题需要改进。在构建缺陷-代码知识库的过程中,未进行缺陷报告中无效、重复、错误情况判断以及专有名词的规定,这会影响引擎结果的准确性;在下一步的工作中,我们将不断扩充和完善引擎构建的缺陷-代码知识库。另外,我们将从多方面优化缺陷报告的质量,从而进一步提高缺陷代码搜索引擎的准确性。

参 考 文 献

- [1] ZHAO Y, LEUNG H, YANG Y, et al. Towards an understanding of change types in bug fixing code[J]. Information & Software Technology, 2017, 86: 37-53.
- [2] LATOZA T D, MYERS B A. Hard-to-answer questions about code[C]// Evaluation and Usability of Programming Languages and Tools on — PLATEAU' 10. Reno, Nevada: ACM Press, 2010: 1-6.
- [3] PRESSMAN R S. Software Engineering: A Practitioner's Approach [M]. New York: McGraw-Hill, 2010: 46-48.
- [4] LI B, HE Y P, MA H T. Automatic program repair: Key problems and technologies[J]. Ruan Jian Xue Bao/Journal of Software, 2019, 30(2): 244-265.
- [5] YOUM K C, AHN J, KIM J, et al. Bug localization based on code change histories and bug reports[C]// Asia-Pacific Software Engineering Conference. 2015: 190-197.
- [6] LE T D B, OENTARYO R J, LO D. Information retrieval and spectrum based bug localization: Better together[C]// FSE. ACM, 2015: 579-590.
- [7] GUO Z Q, ZHOU H C, LIU S R, et al. Information retrieval based bug localization: research problem, progress, and challenges[J]. Journal of Software, 2020, 31(9): 2826-2854.
- [8] WANG S, LO D. Version history, similar report, and structure: Putting them together for improved bug localization[C]// IEEE International Conference on Program Comprehension. ACM, 2014: 53-63.
- [9] NAISH L, HUA J L, RAMAMOCHANARAO K. A model for spectra-based software diagnosis [M]// ACM Trans. Software Engineering & Methodology. 2011, 20(3): 1-32.
- [10] WANG X, ZHANG W, WANG Q. Two-phase bug localization method based on defect repair history[J]. Computer Systems & Applications, 2014, 23(11): 99-104.
- [11] LOYOLA P, GAJANANAN K, SATOH F. Bug Localization by Learning to Rank and Represent Bug Inducing Changes[C]// ACM Press the 27th ACM International Conference on Information and Knowledge Management. 2018: 657-665.
- [12] ZHANG W, LI Z Q, DU Y H, et al. Fine-grained software bug location approach at method level[J]. Ruan Jian Xue Bao/Journal of Software, 2019, 30(2): 195-210.
- [13] TANG M, ZHU L, ZOU X C. Document vector representation based on Word2Vec[J]. Computer Science, 2016, 43(6): 214-217.
- [14] HOANG T, OENTARYO R J, LE T D B, et al. Network-clustered multi-modal bug localization[J]. IEEE Transactions on Software Engineering, 2019, 45(10): 1002-1023.
- [15] CHEN X, JU X L, WEN W Z, et al. Review of dynamic fault localization approaches based on program spectrum[J]. Ruan Jian Xue Bao/Journal of Software, 2015, 26(2): 390-412.
- [16] JAYDEN Z, GIULIA B, ADELE G, et al. How abstract is syntax? Evidence from structural priming[J]. Cognition, 2019, 193(12): 1-13.
- [17] JEAN R F, MORANDAT F, BLANC X, et al. Fine-grained and Accurate Source Code Differencing[C]// ACM/IEEE International Conference on Automated Software Engineering. 2014: 313-324.
- [18] SOUMYA S, SANDIPAN S, SANJUKTA B, et al. Using core-periphery structure to predict high centrality nodes in time-varying networks [J]. Data Mining and Knowledge Discovery, 2018, 32(7): 1368-1396.
- [19] BIRD S. NLTK: the natural language toolkit[C]// Proceedings

of the COLING/ACL on Interactive presentation sessions. Association for Computational Linguistics, 2006:69-72.

[20] ZHU Z W, ZHOU X H, SHAO K. A novel approach based on Neo4j for multi-constrained flexible job shop scheduling problem [J]. Computers & Industrial Engineering, 2019, 130 (4): 671-686.

[21] SALTON G, WONG A, YANG C S. A vector space model for automatic indexing[J]. Communications of the ACM, 1975, 11: 613-620.

[22] ZHOU J, ZHANG H, LO D. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports[C]//IEEE International Conference on Software Engineering. 2012:14-24.

[23] SUN X B, WANG L, WANG J W, et al. Construct Knowledge Graph for Exploratory Bug Issue Searching[J]. Acta Electronica Sinica, 2018, 46(7):1578-1583.

[24] YE X, BUNESCU R, LIU C. Mapping Bug Reports to Relevant Files: A Ranking Model, a Fine-Grained Benchmark, and Feature Evaluation[J]. IEEE Transactions on Software Engineering, 2016, 42(4):379-402.

[25] YOUM K C, AHN J, LEE E. Improved bug localization based on code change histories and bug reports[J]. Information and Software Technology, 2017, 82(2):177-192.

[26] WANG Y, YAO Y, TONG H, et al. Bug Localization via Supervised Topic Modeling [C] // International Conference on Data Mining(ICDM). IEEE, 2018:607-616.

[27] HORVAT H, FEREN C, GERGEL Y, et al. Code coverage differences of Java bytecode and source code instrumentation tools[J]. Software Quality Journal, 2017, 27(12):79-123.

[28] MEHRDAD G, SEYED M B. Runtime deadlock tracking and prevention of concurrent multithreaded programs: A learning-based approach[J]. Concurrency and Computation: Practice and Experience, 2020, 32(10):1-21.

[29] DING H, CHEN L, QIAN J, et al. Fault localization method using information quantity[J]. Ruan Jian Xue Bao/Journal of Software, 2013, 24(7):1484-1494.

[30] SHU T, HUANG M X, DING Z H, et al. Fault localization method based on conditional probability model[J]. Ruan Jian Xue Bao/Journal of Software, 2018, 29(6):1756-1769.



CHANG Jian-ming, born in 1998, post-graduate. His main research interests include bug localization and so on.



BO Li-li, born in 1989, Ph.D, lecturer. Her main research interests include software testing, software security, etc.