软件特征对 bug 定位的影响分析

MF1933128 周慧聪

1 摘要

本课程报告主要围绕对于在信息检索的 bug 定位中软件特征的介绍，对比几种常用的信息检索 bug 定位方法，分析在这些定位方法中所用到的软件特征以及特征抽取的方法。最后根据研究所得的结果，制定研究论文计划。

关键字：bug 定位，信息检索，软件特征

目录

1 摘要 1
2 研究背景 2
3 Bug 定位的特征总结 2
4 基于信息检索 bug 定位的基本流程 3
5 部分信息检索定位方法总结 4
5.1 BLUiR 4
5.2 Amalgam 5
5.3 Lobster 6
5.4 BLIA 7
6 分类器的配置对于 bug 定位的影响 9
7 研究及论文计划 10
8 参考文献 12

2 研究背景

- 定位 bug 非常重要、困难且代价高，对于大型系统尤其如此。
- 为了解决这个问题，越来越多地使用自然语言信息检索技术来分析错误报告，定位潜在错误源文件。尽管这些技术具有很强的延展性，但在实践中，它们的将错误精确定位到少量文件方面的有效性仍然很低。
- 现在的定位方法大多通过分析软件的各种特征来进行 bug 定位，本报告将总结各种软件所用到的特征，并比较各个特征在定位 bug 中的影响。

3 Bug 定位的特征总结

Comparison of IR-based bug localization techniques.

Approach	BugLocator [19]	BLUiR [20]	AmaLgam [21]	BRTracer [23]	Lobster [24]	BLIA [1]
Published	ICSE 2012	ASE 2013	ICPC 2014	ICSME 2014	ICSME 2014	APSEC 2015
Full name	BugLocator	Bug Localization Using information Retrieval	Automated Localization of Bug using Various Information	BRTracer	LOcating Bugs using Stack Traces and tExt Retrieval	Bug Localization using Integration Analysis
IR methods	rVSM (revised VSM)	Indris built-in TF.IDF formulation	VSM	rVSM	Lucene	rVSM
Similarity between bug report and source files	O	O	O	O	O	O
Structured information of source file	X	O	O	X	X	O
Analyzing past similar bugs	O	X	O	O	X	O
Analyzing stack trace	X	X	X	O	O (Dependency graph analysis)	O
Version history	X	X	O	X	X	O
Evaluation metrics	Top N Rank MAP(Mean Average Precision) MRR(Mean Reciprocal Rank)					

- **错误报告和源代码文件的文本信息**

在文本上与 bug 报告文本相似的源代码文件往往与所报告的 bug 相关联。

- **结构化信息的源代码**

代码中的不同字段(例如，类名、包名、注释等)对于匹配错误报告词汇表的重要性各不相同。

- **错误报告中的结构化信息**

错误报告的不同部分，如标题和正文，可能包含特定的或详细的信息，以便进行匹配。此外，一些代码元素经常可以在 bug 报告中被识别，这对于 bug 本地化来说更加有效 。

- **堆栈跟踪**

错误位置可能在堆栈跟踪中列出的类或方法中。

- **分段**

匹配在代码块级别或分割源代码文件成同等大小的段可以提供更准确的 bug 定位。

- **提交日志（版本历史）**

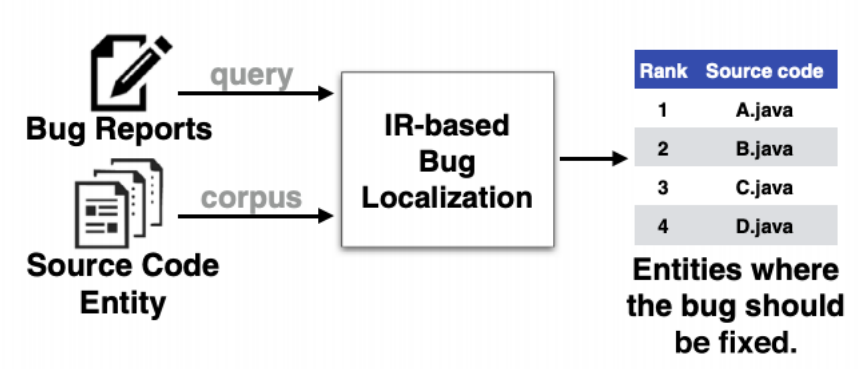
源代码版本管理系统中包含的消息可以提供与用户 bug 报告文本相匹配的功能描述，这比源代码标记更好。

4　基于信息检索 bug 定位的基本流程

基于信息检索的错误定位(IRBL)工具然后根据相关性的概率(通常作为相似度评分来衡量)对文档进行排序。高排名的文件预计是那些可能包含错误代码的文件。这个过程可以减少开发人员必须集中检查的文件数量。

步骤

- **Bug 报告中提取的特征**
- **源代码文件中提取特征**
- **计算特征的相似度**
- **根据相似度进行排名，相似度越高的排名排名越前**

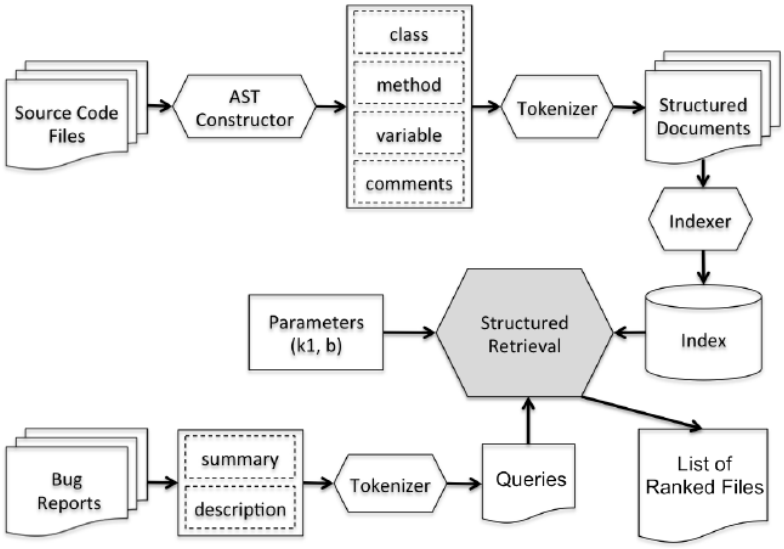


5　部分信息检索定位方法总结

5.1 BLUiR

所用特征：类似的报告和代码结构

方法：



Source Code Parsing & Term Indexing

构建抽象每个源文件的语法树(AST)，并提取所有标识符名称(类名、方法名、变量名等)。然后对完整标识符以及驼峰分割的令牌进行索引。

Retrieval Model

假设文档和查询分别由长度为 n(词汇总数或词汇量大小)的加权词汇频率向量 d 和 q 表示。

$$\vec{d} = (x_1, x_2, ..., x_n)$$
$$\vec{q} = (y_1, y_2, ..., y_n)$$

根据 tf-idf，获得向量：

$$\vec{d_w} = (tf_d(x_1)idf(t_1), tf_d(x_2)idf(t_2), ..., tf_d(x_n)idf(t_n))$$
$$\vec{q_w} = (tf_q(y_1)idf(t_1), tf_q(y_2)idf(t_2), ..., tf_q(y_n)idf(t_n))$$

文档 d 对查询 q 的相似度评分为：

$$s(\vec{d}, \vec{q}) = \sum_{i=1}^{n} tf_d(x_i)tf_q(y_i)idf(t_i)^2$$

Incorporating Structural Information

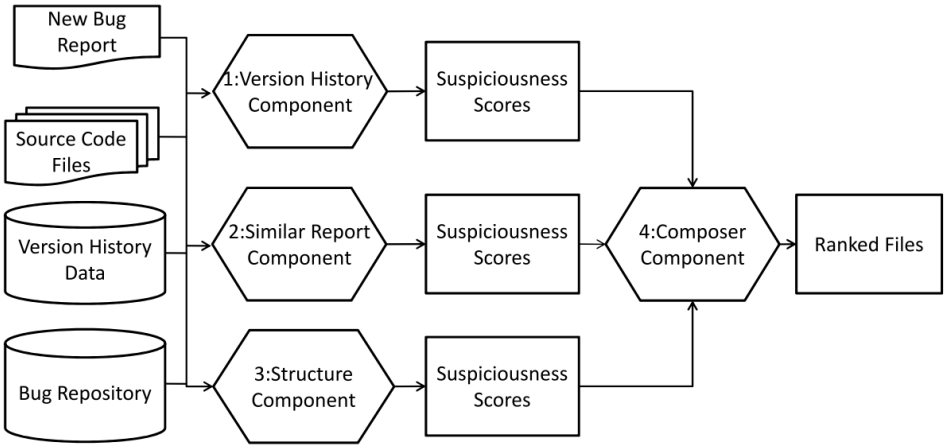
区分来自错误报告的不同字段的两种不同的查询表示(摘要和更详细的描述)。解析源代码结构还允许我们区分四个不同的文档字段:类、方法、变量和注释。

对八个(查询表示、文档字段)组合中的每个组合执行单独的搜索，然后在所有八个搜索中对文档得分求和。

$$s'(\vec{d}, \vec{q}) = \sum_{r \in Q} \sum_{f \in D} s(d_f, q_r)$$

5.2 Amalgam

所用特征：版本历史、类似的报告和代码结构
方法



Version History Component
利用对 bug 预测的研究，其目标是预测将来哪些文件可能有 bug。
该算法将输入提交日志作为输入，并输出一组文件及其可疑性评分。它首先确定相关的 bug 修复提交。相关的 bug 修复提交是通过以下两条规则确定的:
Rule1: 提交日志必须匹配以下正则表达式 regex: (. * fix.*)|(. * bug.*)。这个正则表达式指定将匹配所:有包含单词“fix”或“bug”的提交日志。
Rule2: 必须是过去 k 天之内的提交日志（**k 实验中设置为 15**）

$$score^H(f,k,R) = \sum_{c \in R \wedge f \in c} \frac{1}{1+e^{12(1-((k-t_c)/k))}}$$

R 是**相关提交的集合**，tc 是提交 c 和输入错误报告之间经过的天数。该算法的输出是一组可疑性评分，每个文件一个分数。

Similar Report Component
该算法接受一个输入错误报告和已在错误存储库中修复的旧的错误报告。

$$\overrightarrow{b} = tf_b(t_1)idf(t_1), tf_b(t_2)idf(t_2), \ldots, tf_d(t_n)idf(t_n)$$

$$score_R(f,b,B) = \sum_{b' \in \{b' | b' \in B \wedge f \in b'.Fix\}} \frac{sim(b,b')}{|b'.Fix|}$$

在上式中，b 为输入 bug 报告，B 为旧的修复的错误报告集合。使用 tf-idf 获得错误报告的向量，根据计算余弦相似度获得得分。取每一个输入报告与旧的报告的平均相似度。

Structure Component
使用了 BLUiR，它对 bug 定位执行结构化检索。BLUiR 将 bug 报告分为两部分:摘要和描述。它将源代码文件分成 4 部分:类名、方法名、变量名和注释。所有的部分都用 tf-idf 转化成向量。

给定一个输入错误报告 b，源代码文件 f 的可疑性分数可以计算为:

$$score_S(f,b) = \sum_{fp \in f} \sum_{bp \in b} sim(fp, bp)$$

fp 是文件 f 的部分，bp 是错误报告的部分。这个就是计算文件与错误报告两两相对的相似值之和，作为结构相似度。因为把错误报告分为 2 部分，源代码分成 4 部分，所以是 8 个相似值相加。

Composer Component

将前面 3 个相似度得分按一定的比例相加起来，获得最后的相似度分数。

首先将结构组件输出的分数与文件 f 的类似报告组件相结合: (a 的值设置为 0.2)

$$Susp^{S,R}(f) = (1-a) \times Susp^{S}(f) + a \times Susp^{R}(f)$$

$$Susp^{S,R,H}(f) =$$

$$\begin{cases} (1-b) \times Susp^{S,R}(f) + b \times Susp^{H}(f), & if Susp^{S,R}(f) > 0 \\ 0, & otherwise \end{cases}$$

在上面的等式中，如果 SuspS,R(f)为 0，最终的怀疑度分数设为 0。(b 的值设置为 0.3)

5.3 Lobster

所用特征：代码结构和堆栈跟踪

方法：

Textual Similarity

bug 报告和代码元素 e 之间的文本相似性定义为:

$$\text{sim}_{\text{textual}}(bugReport, e) = \text{score}_{\text{TR}}(bugReport, e)$$

评分函数由任何 TR 技术提供文档之间的相关性度量，例如。VSM ， LSI 或 LDA。Lobster 的相似性计算是由 Lucene 给出的，该 TR 模型通过结合 VSM 和布尔模型被证明在错误定位中比 LSI 等技术表现更好。

Structural Similarity

给定一个堆栈跟踪和一个代码元素 e，根据堆栈跟踪中的代码元素与 e 之间的最小距离来定义它们之间的结构相似性。该距离的计算基于**软件系统的程序依赖图**，即，其中每个节点表示系统的一个不同的代码元素，从一个节点到另一个节点的边表示控制或数据流。

$$\text{dist}(stackTrace, e) = \min \begin{pmatrix} \forall_{d \in stackTrace} \text{shortestPath}(d, e) \\ \cup \forall_{d \in stackTrace} \text{shortestPath}(e, d) \end{pmatrix}$$

如果两个代码元素之间不存在路径，则它们之间的最短路径为无穷大。注意，如果在堆栈跟踪中列出了代码元素 e，它们之间的距离是 0。

将堆栈跟踪与代码元素 e 之间的结构相似性定义为它们之间的归一化距离的补充，具体如下:

$$\text{sim}_{\text{struct}}(stackTrace, e) = 1 - \frac{\min(\text{dist}(stackTrace, e), \lambda)}{\lambda}$$

λ参数是定义最大考虑距离的阈值。结构相似度值在[0,1]范围内，其中 1 表示最大相似度，0 表示无相似度。

Total Similarity

将软件中的 bug 报告和代码元素 e 之间的总相似性定义为它们的文本和结构相似性之间的线性组合：

$$\begin{aligned}\mathrm{sim}(bugReport, e) = (1-\alpha) * \mathrm{sim}_{\mathrm{textual}}(bugReport, e) \\ + \alpha * \mathrm{sim}_{\mathrm{struct}}(\mathrm{getStackTrace}(bugReport), e)\end{aligned}$$

5.4 BLIA

使用特征：版本历史，相似报告，代码结构和堆栈踪迹

方法：

分为 file 级别和 method 级别

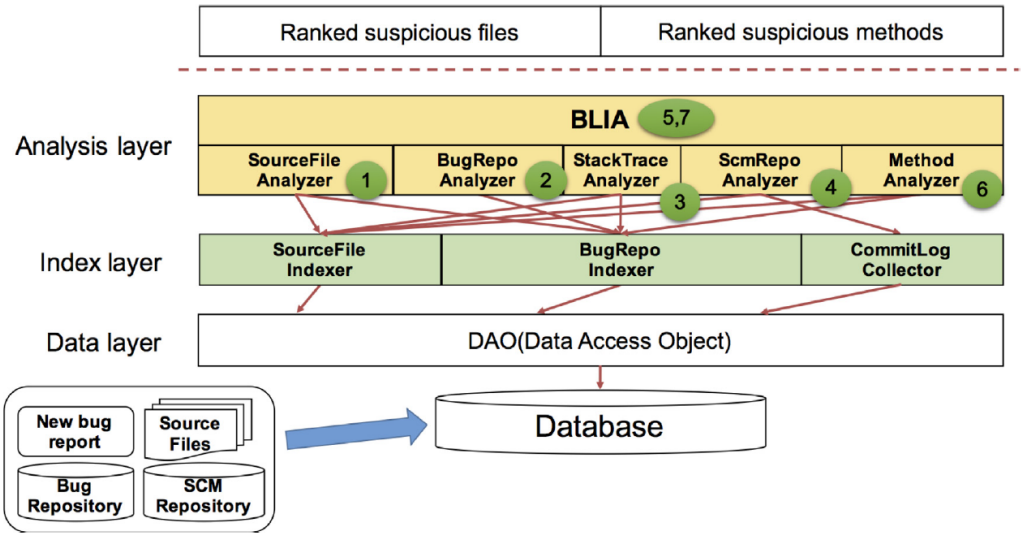
输入

• Bug report (report date, scenario, stack traces, comments, etc.)

• Source files

• Similar fixed bug reports

• Source code change history (commit messages and differences among changes)



file 级别

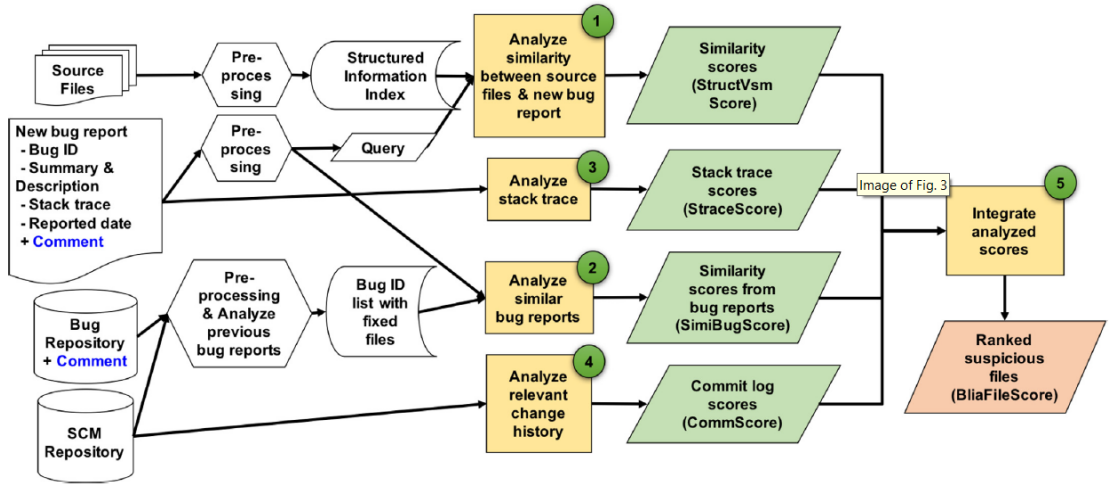
步骤 1：分析源文件和新的错误报告之间的相似性。源文件的结构化信息是通过预处理生成的。(**StructVsmScore**)。

步骤 2：我们将分析类似的修复错误报告。如果发现类似的 bug 报告，则计算每个修复文件的相似度评分(**SimiBugScore**)。

步骤 3：如果堆栈跟踪在错误报告中，从错误报告中的堆栈跟踪提取源文件信息。(**StraceScore**)

步骤 4：为提交的文件生成一个计算分数(**CommScore**)。相关方法的提交日志分数将在方法级别分析期间使用。？

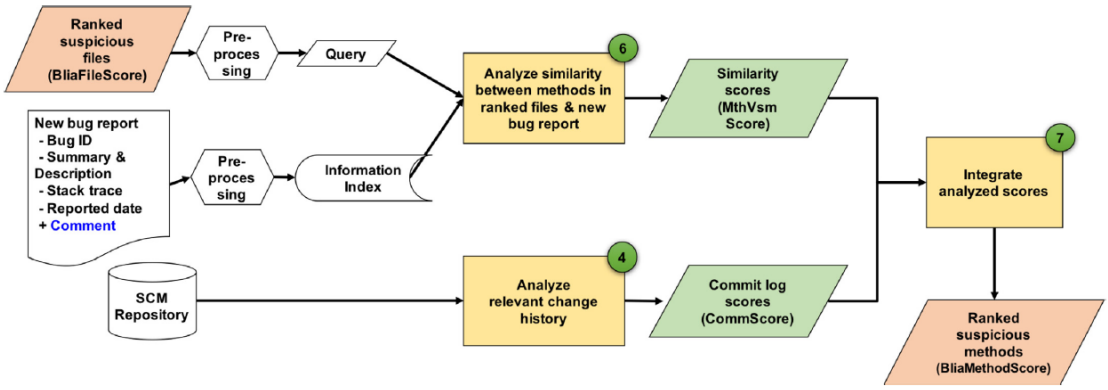
步骤 5：上述四个分析的分数与三个控制参数集成在一起，以计算每个源文件的最终等级可疑分数(**BliaFileScore**)。



method 级别

第 6 步：方法和新的错误报告之间的相似之处是使用第 5 步中排列的可疑源文件作为输入进行分析。对排序后的可疑文件进行预处理，然后使用方法信息进行查询。(file 级别的基础上) bug 报告也经过预处理。将生成这些对象的信息索引，而不是步骤 1 中的查询。

第 7 步： 提交日志分数为每个方法从步骤 4 和相似性得分方法和错误报告之间的集成来计算最终的排名可疑得分为每个方法(**BliaMethodScore**)为了定位缺陷的错误报告在方法级别。



6 分类器的配置对于 bug 定位的影响

The configuration parameters and the values of the IR (e.g., VSM, LSI, and LDA) and EM family of classifiers, as proposed by Thomas et al. [2].

Parameter	Value
Parameters common to all IR classifiers	
(A) Bug report representation	A1 (Title only)
	A2 (Description only)
	A3 (Title + description)
(B) Entity representation	B1 (Identifiers only)
	B2 (Comments only)
	B3 (Idents + comments)
	B4 (PBR-All)
	B5 (PBR-10 only)
	B6 (Idents + comments + PBR-All)
(C) Preprocessing steps	C0 (None)
	C1 (Split only)
	C2 (Stop only)
	C3 (Stem only)
	C4 (Split + stop)
	C5 (Split + stem)
	C6 (Stop + stem)
	C7 (Split + stop + stem)
Parameters for VSM only	
(D) Term weight	D1 (tf-idf)
	D2 (Sublinear tf-idf)
	D3 (Boolean)
(E) Similarity metric	E1 (Cosine)
	E2 (Overlap)
Parameters for LSI only	
(F) Term weight	F1 (tf-idf)
	F2 (Sublinear tf-idf)
	F3 (Boolean)
(G) Number of topics	G32 (32 topics)
	G64 (64 topics)
	G128 (128 topics)
	G256 (256 topics)
(H) Similarity metric	H1 (Cosine)
Parameters for LDA only	
(I) Number of iterations	I1 (Until model convergence)
(J) Number of topics	J32 (32 topics)
	J64 (64 topics)
	J128 (128 topics)
	J256 (256 topics)
(K) α	K1 (Optimized based on K)
(L) β	L1 (Optimized based on K)
(N) Similarity metric	N1 (Conditional probability)
Parameters for EM only	
(M) Metric	M1 (Lines of code)
	M2 (Churn)
	M3 (New bug count)
	M4 (Cumulative bug count)

实验结果

- 1.不同配置的性能差异很大
- 2.性能相同的配置在工作量方面差异很大
- 3.方法级别的定位中，VSM 模型在性能和工作量两方面均表现最好
- 4.方法级别的最有效配置同样适用于文件级别的定位，反之亦然

7 研究及论文计划

实证研究型论文

使用不同的配置的 bug 定位方法，比较定位结果的准确性指标和工作量感知的指标，找出最优配置。已有研究表明 VSM 在 IRBL 领域中表现最好[1]，并且被用来构建多种定位方法(BugLocator, BRTracer etc.)[3,4,5,6]，近年，[2]提出使用信息论(PMI,NGD)建模效果好于传统的 VSM，但是目前仍没有工作使用基于信息论新模型结合软件特征来构建定位方法,本文主要研究 VSM 和信息论模型(PMI,NGD)在结合特征之后的表现效果

（1） 模型

基于 IR 的分类器模型配置（VSM，PMI，NGD）

参数		数值
所有 IR 分类器模型的通用参数		
A bug 报告的表示		A1 Title
		A2 Description
		A3 Title+Description
B 源代码表示	源代码本身文本	B1 标识符名称
		B2 注释
		B3 注释+标识符
	基于历史错误报告的特征值	B4 基于 10 个历史错误报告
		B5 标识符+注释+基于所有的历史错误报告
C 预处理阶段		C0 不做任何处理
		C2 拆分标识符，删除停用词
		C3 拆分标识符，删除停用词，使用 Porter 词干算法进行词干分析
VSM 的参数		
D 术语权重		D1 Tf-idf
		D2 线性 TF-idf
		D3 布尔型
E 相似度评分		E1 余弦相似度
		E2 重叠相似度
PMI 的参数		
F IDF 的计算		F1 Tf-idf 中的 idf
		F2 线性 TF-idf 中的 idf
NGD 的计算		
G IDF 的计算		G1 Tf-idf 中的 idf
		G2 线性 TF-idf 中的 idf

定位方法所抽取的软件特征组合配置

特征	抽取方法
H 代码结构	H0 不做任何处理
	H1 将源代码分为不同字段：类、方法、变量和注释 分别进行相似度计算
I 类似的报告	I0 不做任何处理
	I1 使用 VSM 计算相似度
	I2 使用 PMI 计算相似度
	I3 使用 NGD 计算相似度
J 版本历史	J0 不做任何处理
所提交日志的天数 k	J1 k=5
	J2 k=10
	J3 k=15
K 堆栈跟踪	K0 不做任何处理
	K1 考虑堆栈跟踪

（2）评价指标

传统的评价指标：TopK,MAP,MRR

基于工程量感知的评价指标

（3）研究问题

1、基于 IR 的定位方法所使用的信息检索模型配置是否会显著影响文件级错误定位的性能？

该研究问题主要经比较不同检索模型(VSM，PMI，NGD)的定位结果是否差异以及是否显著。

2、基于 IR 的定位方法所抽取的软件特征组合配置是否会显著影响文件级错误定位的性能？

该研究问题主要比较特征组合方式对定位性能的影响。

3、基于 IR 的定位方法在文件级上错误定位的最优配置是否适用在方法级上的错误定位？

（4）实验方案

根据配置做对照组实验

1、 是寻找 IR 分类器配置上的最优。

首先将配置 ABC 进行整合，在 VSM 模型进行实验，找出最优的 ABC 配置。

其次，使 PMI 和 NGD 来代替 VSM 模型，验证这两种模型是否优于 VSM。

2、 在 1 的基础上获取特征值组合的最优。

根据不同的组合，来观测每种特征值对于最后定位效果的影响。

3、 以相同的配置在方法级上进行错误定位。

验证方法级与文件及的错误定位最优配置是否一致。

（5）结果表达方法

使用箱线图可以看出在不同参数情况下，在各个数据集上的定位效果。

8 参考文献

[1] Tantithamthavorn, Chakkrit, Abebe, et al. The impact of IR-based classifier configuration on the performance and the effort of method-level bug localization[J]. IST 2018:160-174.

[2] Wang SW, Lo D. Version history, similar report, and structure: putting them together for improved bug localization. ICPC 2014: 53-63.

[3] Zhang W, Li ZQ, Qing Wang, Juan Li. FineLocator: A novel approach to method-level fine-grained bug localization by query expansion. Information & Software Technology, 2019,110:121-135.

[4] Koyuncu A, Bissyande TF, Kim DS, Liu K, Klein J, Monperrus M, Traon YL. D&C: A divide-and-conquer approach to IR-based bug localization. IEEE Trans. on Software Engineering, 2019.

[5] Youm KC, Ahn J, Lee E. Improved bug localization based on code change histories and bug reports. Information & Software Technology, 2017,82:177-192. IST

[6] Dilshener T, Wermelinger M, Yu YJ. Locating bugs without looking back. Automated Software Engineering, 2018,25(3):383-434.

[7] Rahman MM, Roy CK. Improving IR-based bug localization with context-aware query reformulation. In: Proc. of the 26th ACM SIGSOFT Int' l Symp. on Foundations of Software Engineering. ACM Press, 2018:621-632.

[8] Rath M, Lo D, Mäder P. Analyzing requirements and traceability information to improve bug localization. MSR 2018:442-453.

[9] Wang YJ, Y Yuan, Tong HH, Huo X, Li M, Xu F, Lu J. Bug localization via supervised topic modeling. ICDM 2018: 607-616.

[10] Loyola P, Gajananan K, Satoh F. Bug localization by learning to rank and represent bug inducing changes. CIKM 2018:657-665.