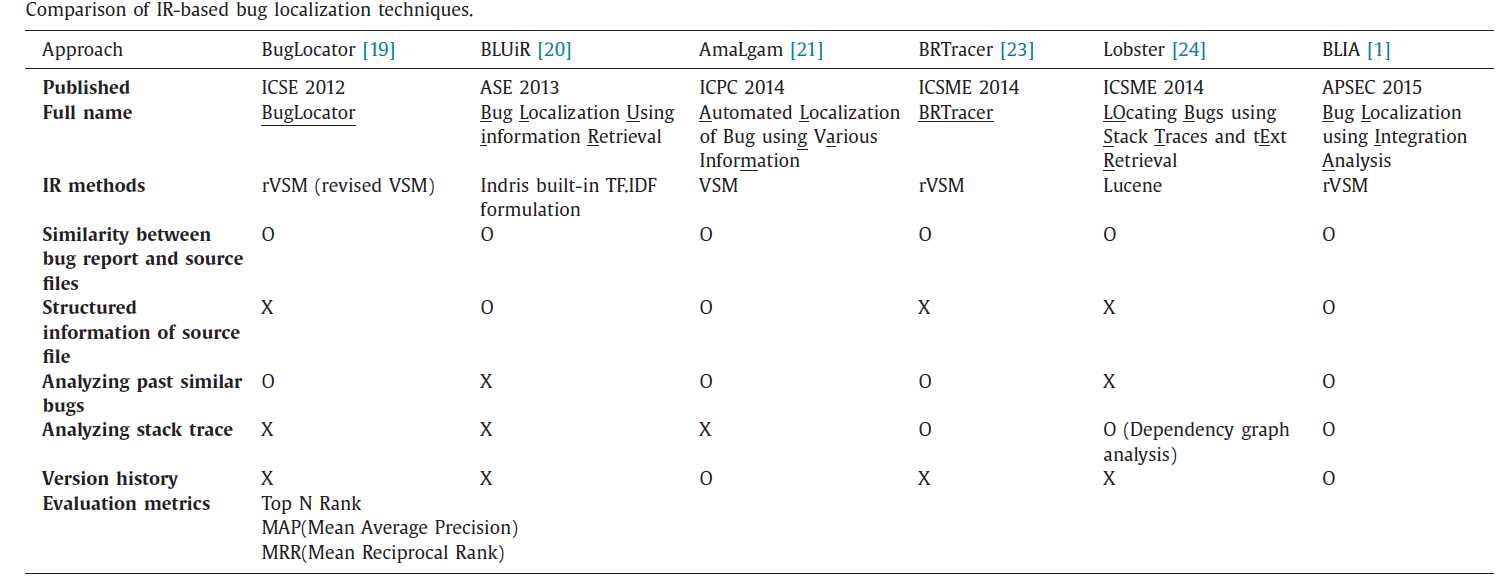
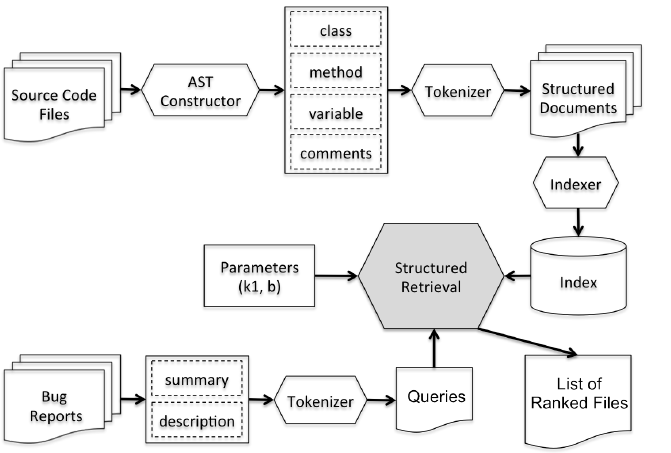
**Bug定位的特征总结**



**BLUiR**

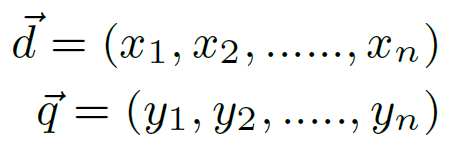
**所用特征：类似的报告和代码结构**

**方法：**

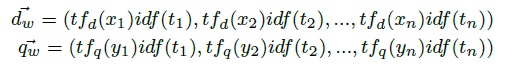
**Source Code Parsing & Term Indexing**

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。

根据tf-idf，获得向量：



文档d对查询q的相似度评分为：



**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。

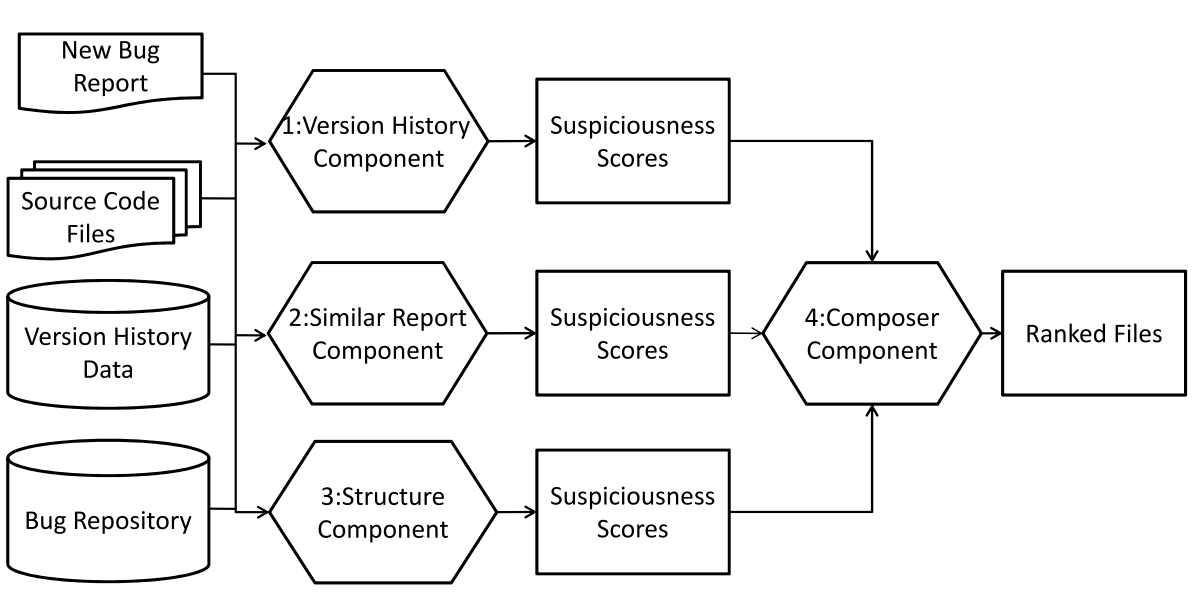
对八个(查询表示、文档字段)组合中的每个组合执行单独的搜索，然后在所有八个搜索中对文档得分求和。（**Amalgam的一部分**）



**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。

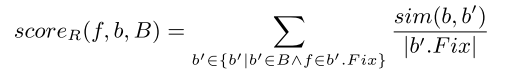
Rule2: 必须是过去k天之内的提交日志（**k实验中设置为15**）

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。

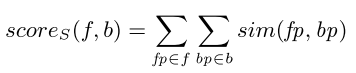




在上式中，b为输入bug报告，B为旧的修复的错误报告集合。使用tf-idf获得错误报告的向量，根据计算余弦相似度获得得分。取每一个输入报告与旧的报告的平均相似度。

**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:

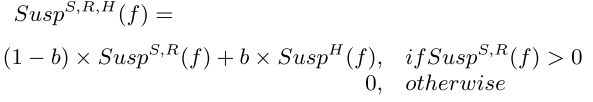
fp是文件f的部分，bp是错误报告的部分。这个就是计算文件与错误报告两两相对的相似值之和，作为结构相似度。因为把错误报告分为2部分，源代码分成4部分，所以是8个相似值相加。

**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)





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)

**Lobster**

**所用特征：代码结构和堆栈跟踪**

**方法：**

**Textual Similarity**

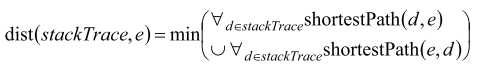
bug报告和代码元素e之间的文本相似性定义为:



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。



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。

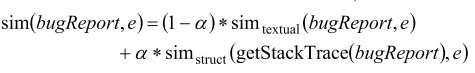
将堆栈跟踪与代码元素e之间的结构相似性定义为它们之间的归一化距离的补充，具体如下:



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：



**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**

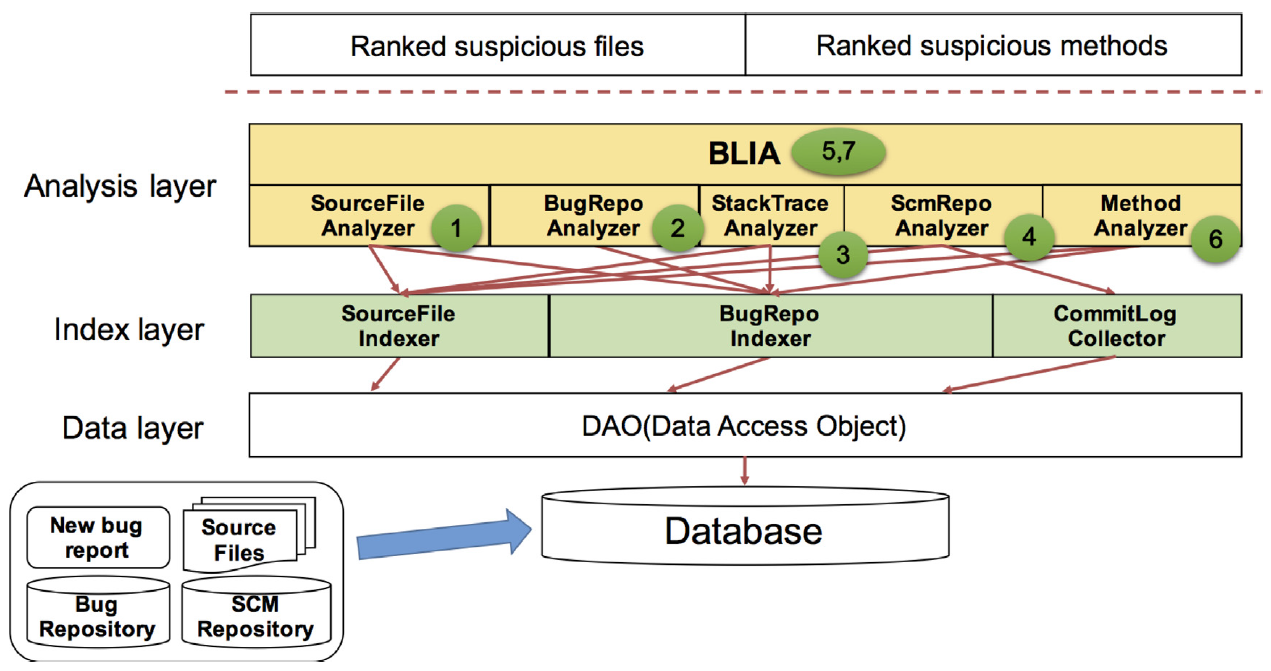
**Input**

• 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**

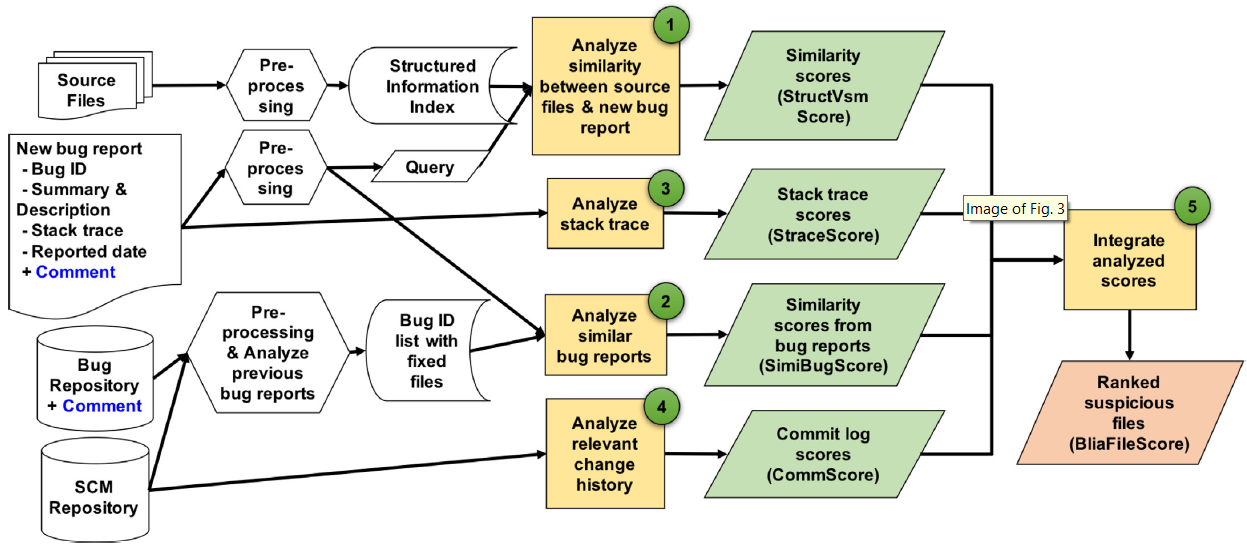
步骤1：分析源文件和新的错误报告之间的相似性。源文件的结构化信息是通过预处理生成的。 (**StructVsmScore**)。

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。

