**摘要**

软件缺陷定位通过定位程序中的缺陷所在位置为调试提供关键信息，从而提高软件调试和修复的效率，是一种重要的软件质量保障技术。在此前的研究中，基于频谱、变异和切片的软件缺陷定位方法已取得了较好的效果。然而，上述方法都存在大量语句可疑度相同的问题，需要测试人员人工确认，影响缺陷定位的效率和精确度。例如：使用Ochiai和DStar方法对Defects4J 中6个项目的395个存在缺陷的版本进行缺陷定位，都有近70%的版本存在多条语句可疑度并列排名第1。针对上述问题，本文提出了一种通过语句复杂度分析提升缺陷定位精确度的方法CBFL（Complexity-based fault localization）。首先，使用SBFL、MBFL等传统方法进行缺陷定位，得到程序中语句的可疑度排名；然后，针对排名靠前的语句进行复杂度分析，提取特征以及类别标签，构建数据集；最后，构造基于语句复杂度特征的缺陷倾向性分类模型，预测排名靠前的语句中更有可能存在缺陷的位置。基于上述方法，本文实现了基于语句复杂度分析的缺陷定位工具，并在真实的缺陷数据集 Defects4J 项目上进行了仿真实验，分别与基于频谱的缺陷定位方法DStar和基于变异的缺陷定位方法MUSE进行了对比，仿真结果表明本文所提出的方法具有更好的缺陷定位效果。

1. **引言**

近年来，随着互联网与信息技术的飞速发展，各种软件产品的数量和规模与日俱增。各行各业对软件产品的需求越来越迫切，也越来越复杂。当软件产品的数量和规模日趋庞大，发现和修复软件中缺陷的难度也就逐步增加。而软件中包含的缺陷数目与软件产品的质量息息相关，包含的缺陷数量越多，软件可靠性面临的挑战就越大。为了保证软件的可靠性，在软件项目正式发布上线之前，通常要经过科学且系统的软件测试。软件开发及测试人员需要精准找到缺陷所在位置并进行修复，直至达到上线标准才可正式发布，这个过程往往会消耗掉大量的人力、物力与时间，而且人工修复错误率较高。

In recent years, with the rapid development of the Internet and information technology, the number and scale of various software products have increased day by day. The demand for software products in all walks of life is becoming more and more urgent and more complex. As the number and scale of software products become larger, the difficulty of finding and repairing software faults gradually increases. The number of faults contained in software is closely related to the quality of software products. The more faults contained, the greater the challenge to software reliability.(The more faults a software contains, the more challenges it faces to reliability.)To ensure the reliability of the software, before the software project is officially released, it usually undergoes scientific and systematic software testing.(Before the software project is officially released and launched, in order to ensure the reliability of the software, it is necessary to go through scientific and systematic software testing.) Software developers and testers need to accurately find the location of the fault and fix it until it reaches the online standard before it can be officially released.(,only when it reaches the online standard can it be officially released.) This process often consumes a lot of manpower, material resources and time, and (on the other hand) the error rate of manual repair is high. (the error rate of manual repair of faults is high.)

软件缺陷定位通过定位程序中的缺陷所在位置为调试提供关键信息，从而提高软件调试和修复的效率，是一种重要的软件质量保障技术。在此前的研究中，基于频谱、变异和切片的软件缺陷定位方法已取得了较好的效果。然而，这些方法都面临一个十分严峻的挑战，即缺陷定位的效率和精度不高。例如，使用Ochiai和DStar方法对Defects4j中6个项目的395个存在缺陷的版本进行缺陷定位，都有近70%的版本存在多条语句可疑度并列排名第1的问题~~，并且~~~~DStar生成的并列第1的语句平均数量是22~~。这时候的定位结果仍然需要耗费大量人力进行人工确认，而这无疑加重了软件开发过程中的成本开销。因此，研究出有效提升软件缺陷定位精确度的方法，用于提升软件缺陷定位工作的效率是十分必要的。

Software fault location provides key information for debugging by locating the location of the defect in the program, thereby improving the efficiency of software debugging and repairing. It (Software fault location)is an important software quality assurance technology. In previous research, software fault location methods based on frequency spectrum, mutation, and slice have achieved good results.(better positioning effect.)However, these methods all face a very serious challenge, that is that the efficiency and accuracy of fault location are not high. (inefficient and inaccurate.)For example, using Ochiai or DStar to locate 395 faulty versions of 6 projects in Defects4J, nearly 70% of the faulty versions have more than one suspicious statement are ranked as tied 1.~~The average number of statements for 1 is 22.~~The positioning result at this time still requires a lot of manpower for manual confirmation, which undoubtedly increases the cost of the software development process. Therefore, it is very necessary to develop a method to effectively improve the accuracy of software fault location, which is used to improve the efficiency of software fault location work.

在本文中，提出了一种通过语句复杂度分析提升缺陷定位精确度的方法CBFL （Complexity-based fault localization）。通常复杂度被应用于数学和计算机导论中，用于计算可执行程序运行时所需要的时间资源和内存资源。而在本论文中，我们要提出一种新的语句复杂度，以程序中每一行代码为单位，根据所提度量元，度量该语句的复杂度作为判断缺陷语句的条件。据我们所知，这样的度量元在之前的研究中还未被提出过（尤其在缺陷定位的背景下）。

In this article, a method CBFL (Complexity-based fault localization) is proposed to improve the accuracy of fault location through statement complexity analysis. Complexity is usually used in mathematics and computer introduction (Introduction to Computers)to calculate the time resources and memory resources required by the executable program to run. In this thesis, we will propose a new statement complexity, which takes each line of code in the program as a unit, and measures the complexity of the statement as a condition for judging faulty statements according to the proposed metric. As far as we know, such a metric has not been proposed in previous researches (especially in the context of fault location).

A.解决方案（PROPOSED SOLUTION）

本文提出了一种通过语句复杂度分析提升缺陷定位精确度的方法CBFL。首先，使用传统方法进行缺陷定位，得到程序中语句的可疑度排名；然后，针对排名的语句进行复杂度分析，提取特征以及类别标签，构建数据集；最后，构造基于语句复杂度特征的缺陷倾向性分类模型，预测排名靠前的语句中更有可能存在缺陷的位置。本文实现了基于语句复杂度分析的缺陷定位方法，并在真实的缺陷数据集 Defects4J上进行了仿真实验，对每个经过缺陷定位的程序结果进行筛选，并通过使用Einspect@n和EXAM作为评价标准与原方法对比，印证了本文所提出方法的有效性。

*B.* CONTRIBUTIONS

本文的主要贡献可以总结如下。

**·** 提出一种新的方法CBFL，通过语句复杂度分析提升缺陷定位精确度。

**·** 提出一种新的语句复杂度计算方法，以程序中每一行代码为单位，根据本文所提出度量元，提取语句特征并构造缺陷倾向性分类模型

**·** 在Defects4j基准测试上对CBFL与SBFL和MBFL进行的实验评估。

本文的其余部分组织如下。第2节介绍了与本文中提出主题相关的工作。第3节概述了基于语句复杂度分析的缺陷定位方法CBFL的设计及步骤。第4节对所提出的研究问题设计实验，介绍评估标准，并根据评估标准与对照实验进行比较，分析实验结果。展示了本文提出的方法如何提高了缺陷定位结果的有效性。第5节论述了方法的有效威胁和未来发展。最后，我们在第6节做出对本文工作的总结。

1. **相关工作（RELATED WORK）**

软件缺陷定位通过定位程序中的缺陷所在位置为调试提供关键信息，从而提高软件调试和修复的效率，是一种重要的软件质量保障技术。在这一节，我们将介绍与本文提出缺陷定位方法相关的传统软件缺陷定位工作。

**2.1基于频谱的软件缺陷定位（SBFL）**

SBFL运用程序谱和测试结果分析软件中的语句，从而定位到缺陷语句所在位置。程序谱包含软件测试时程序实体的覆盖信息，测试结果是指测试用例是否执行通过。当一条语句没有被失败的测试用例覆盖时，则认为该语句为缺陷的可疑度极低，相反，如果一条语句频繁地被失败的测试用例覆盖，则认为这条语句的可疑度非常高。通常SBFL具有计算方法易操作以及消耗资源少的特点，并且该技术的定位效果较好，所以是目前软件缺陷定位问题的研究热点。

研究人员已经提出了很多基于频谱的缺陷定位方法，其中较为经典的有：由 Jone 等人最先提出的Tarantula法，以及Abreu等人随后提出的Jaccard方法和 Ochiai方法。这三种基于频谱的缺方法在缺陷定位工作中都取得了不错的定位效果。而随着研究的不断深入，Wong 等人通过进一步分析测试用例在程序实体上，成功执行的数量对缺陷定位结果的影响，得出了被成功测试用例覆盖次数越多的程序实体，其覆盖次数对可疑值的贡献度越小这一结论，从而提出了Wong1(s)、Wong2(s)和Wong3(s)三个公式。随后为了突出在成功用例数量较多时，失败用例对定位结果的影响，又提出了DStar方法，并且通过实验对比发现 DStar 方法的缺陷定位效果优于其他同类缺陷定位方法。

**2.2基于变异的软件缺陷定位（**MBFL**）**

MBFL是基于变异的缺陷定位技术。这里的变异是一种基于错误注入的软件缺陷定位方法，将一个程序采用某种规则对程序中的一处或多处进行改变，得到一个修改后的可执行的程序副本。其中使用的规则称为变异算子，得到的程序称为变异体。具体来讲，MBFL可以通过变异程序中的缺陷来修复程序，其关注的是当变异一个有缺陷的语句会对程序带来什么影响。首先，MBFL会产生多个突变算子，通过用变异算子去替换或修改程序中的语句，突变算子可以是语句，操作，变量，常量等。然后，将突变后的变异体（程序）使用测试用例执行，收集变异体覆盖信息，最后计算变异体的可疑度，一般选择可疑度最大的变异体作为对应语句的可疑度值。通常来说，如果一个正确的语句被变异，那么程序执行通过的测试用例会减少。相比之下，变异有缺陷的语句，会比变异正确的语句通过更多的测试用例。

MBFL是近几年出现的新兴缺陷定位方法，其中被提出的经典方法有：由文章[]提出的一种MBFL技术MUSE和文章[]提出的Metallaxis，MUSE和Metallaxis中被kill的定义是不一样的。在MUSE中，失败的测试用例必须变为通过才能算作一个突变体。在Metallaxis中，一个失败的测试用例只需要生成一个不同的输出(可能仍然失败)就可以算作为一个突变体。而经过研究表明，MUSE和Metallaxis这两个方法是性能最好的两种MBFL技术[]，能够有效的提高软件缺陷的定位效率，从而达到减小缺陷定位的成本的目标。

**2.3其他缺陷定位方法**

目前还有其他的缺陷定位方法，例如基于程序切片的缺陷定位方法[]，在定位工作过程中，分析程序的输出变量以及执行路径，将输出语句或者与变量有所关联的语句做程序切片，从而达到缩小程序代码规模的效果。早在1984年，程序切片用于软件缺陷定位首次引起了研究人员的注意[]。Lin等[]将动态切片与正向跟踪技术相结合，提出了一种新的缺陷定位方法，该方法在前向跟踪过程中利用了来自设计规范的信息，增强上下文以减小所构造动态切片的大小[]。谢晓媛等[]使用变质切片的概念开发了一种可扩展SBFL的框架，该框架在不需要测试预言机的情况下，可提供与现有SBFL技术相近的性能。此外，董俊华[]将程序频谱与程序切片相结合，提出了一种程序切片缺陷定位方法，并与Tarantula、Jaccard、Ochiai三种频谱方法进行对比，实验结果再次证明了程序切片缺陷定位方法能够有效提高软件缺陷的定位效率，从而达到减小缺陷定位的成本的目标。

1. **方法设计**

根据本文第一节内容，现有缺陷定位方法中最突出的问题是缺陷定位效率不高，定位结果往往存在大量语句可疑度相同，难以区分的问题。因此本文提出基于语句复杂度分析的方法，旨在区别可疑度相同的程序语句，提高现有缺陷定位方法的定位效率，减少人工审查的代码量，从而降低软件项目的成本。其步骤如图1所示。

1. 数据提取与度量元度量：根据一个已知真实的缺陷数据集进行数据提取，对程序中每条语句进行是否是真实缺陷的标记。并根据提出的语句复杂度度量元对定位到的每条语句进行度量，提取每条语句的度量元信息。每条语句的度量元都形成一个能够输入机器学习模型的特征向量。
2. 构建模型：将标记好的特征向量输入到有监督学习的软件缺陷倾向性预测模型。软件缺陷倾向性预测对应机器学习中的二分类问题，此步骤即是利用机器学习中的分类算法构建一个能够用于二分类的分类器。所构建的分类器用于判断待测语句是否有缺陷。
3. 缺陷预测：利用构建好的分类器对待测语句进行缺陷预测。（这里要不要对待测语句进行度量元度量）将待测语句输入构建好的软件缺陷倾向性预测模型，输出得到测试语句的预测结果，查看其是否有缺陷。

接下来将概述基于语句复杂度分析的缺陷定位方法CBFL的设计及步骤，并且详细介绍本文提出的度量元，在此之前的缺陷定位研究中，语句复杂度的度量元还没有被提出过。

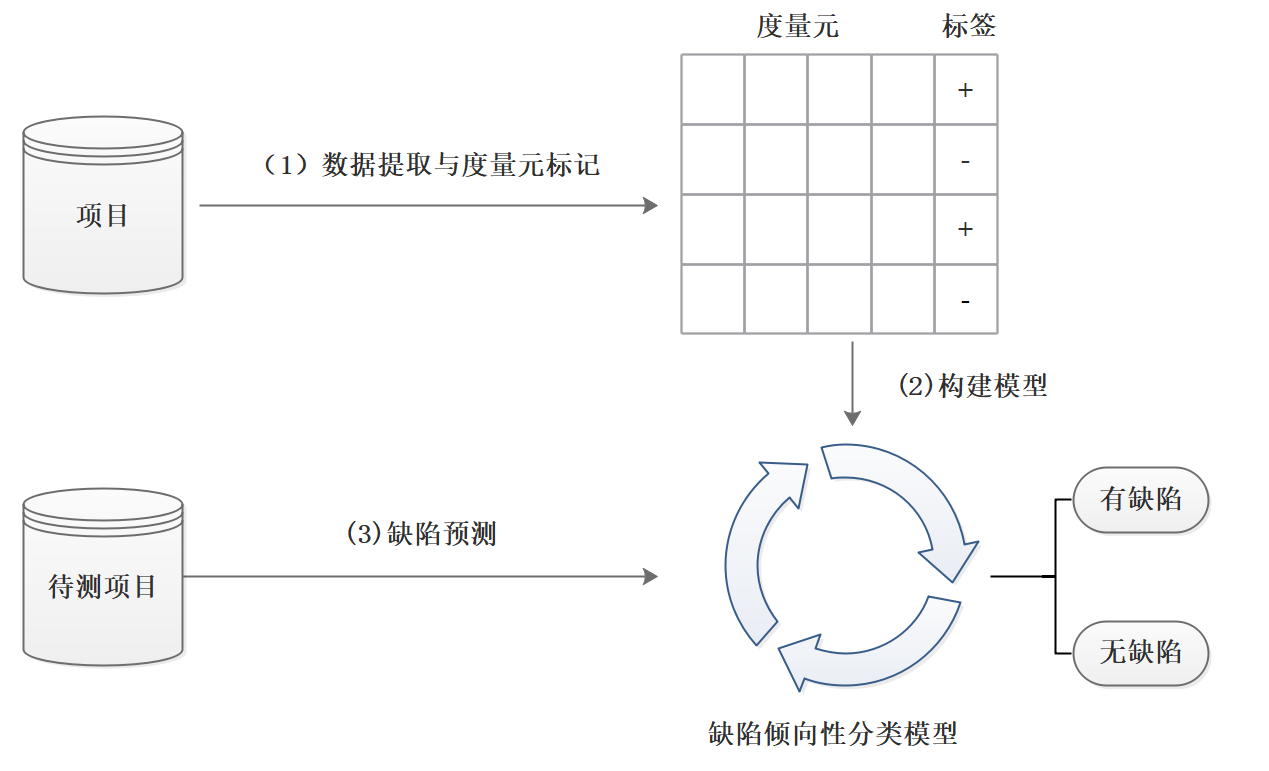


图 1软件缺陷倾向性预测技术流程图

3.1软件缺陷倾向性预测模型的构建工作

本文是在基于语句复杂度分析的基础上进行一次缺陷倾向性预测。首先要对一个已知的缺陷数据集进行数据提取，提取到程序中的每条语句，并对每条语句是否是真实缺陷进行标记。然后，根据提出的语句复杂度度量元对定位到的每条语句进行度量，提取每条语句的度量元信息。并将每条语句的度量元都形成一个能够输入机器学习模型的特征向量。如图1中（2）步骤上的表格，每一行代表一条语句，在每一行的最后标注着此语句的标签，“+”为真实缺陷，“-”为错误缺陷。每一列表示一个度量元，表示当前语句中是否含有此度量元，或含有此度量元的数量。最后，将标记好的特征向量输入到有监督学习的软件缺陷倾向性预测模型。软件缺陷倾向性预测对应机器学习中的二分类问题，此步骤即是利用机器学习中的分类算法构建一个能够用于二分类的分类器。目标是所构建的分类器能够用于判断待测语句是否有缺陷。

由于模型构建的过程对数字类型的训练数据相对友好，故本文在确定度量元属性后，根据已知数据集提供的项目版本，获取到相应的源程序语句集。利用AST树和正则表达式等方法对各个代码文件进行处理，并且得到每条语句信息，信息包括语句的所在项目文件以及行号等信息。考虑到并不是每条语句都能取到所有属性值，故而本文在度量元数据统计过程中将不存在的特征值记为 0。

3.2软件缺陷倾向性预测模型的度量元设计

在软件工程的算法中有复杂度的概念，比如用来定性描述算法运行时间的复杂度，被称为时间复杂度（Time complexity）。本文的研究对象为程序语句，并且将复杂度的概念引入到程序语句中。根据本文提出的一种新的语句复杂度度量元确定语句的复杂程度，而后根据程序语句的复杂度来判断其缺陷情况。

本文提出一种新的语句复杂度以程序中每一行代码为单位，根据度量元度量该语句的复杂度作为判断缺陷语句的条件。据我们所知，在缺陷定位的背景下这样的度量元在此前研究中还未被提出过。度量元的质量是决定缺陷预测性能最重要的因素之一。缺陷预测领域经常使用的度量元有CK度量元、McCabe度量元等，主要关注代码复杂度以及面向对象特征两方面。本文针对语句的缺陷预测设计了度量元集SC度量元(statement complexity)。

我们在设计语句复杂度的度量元时，首先考虑了程序语句中的变量、变量类型、常量、常量类型、关键字、运算符、语句长度等常见代码语句特征，又考虑到语句在整体代码中的嵌套层数也会影响语句的复杂度，所以加入了代码深度这一特征。

本文使用Eclipse的jdt库将java文件转换为AST树，通过遍历AST树的方式查找指定代码语句中包含的变量、类、函数和代码深度等特征；通过编写语法分析器，解析代码语句中的括号数量、括号深度和运算符等特征。

本文将java文件生成AST，并遍历整个语法树，将当前语句在代码中的括号层数计算出来即为语句的代码深度。（这里昊宸需要用再细致的步骤重新描述一下）。

最终我们设计出了24种语句复杂度的度量元，并在多次调试后，选择了其中的12种，将其作为模型的度量元评估语句的复杂度，如表3所示。

表3 语句复杂度度量元（Metric of statement complexity）

|  |  |  |  |
| --- | --- | --- | --- |
| Metric | implication | | employ |
| 变量（variable） |  |  |  |
| 变量数量（Number of variables） |  |  | √ |
| 变量类型（Variable type） |  |  |  |
| 变量种类（Variable types） |  |  |  |
| 常量（Constant） |  |  |  |
| 常量数量（） |  |  |  |
| 常量类型 |  |  |  |
| 运算符 |  |  |  |
| 运算符数量 |  |  | √ |
| 数组/列表 |  |  | √ |
| 关键字 |  |  |  |
| 关键字数量 |  |  | √ |
| 函数 |  |  |  |
| 函数数量 |  |  | √ |
| 类 |  |  |  |
| 类数量 |  |  | √ |
| 逻辑运算 |  |  | √ |
| 逻辑运算数量 |  |  |  |
| 代码长度（字符串） |  |  | √ |
| 代码元素数量 |  |  | √ |
| 注释（//,/\*\*/） |  |  |  |
| 括号深度 |  |  | √ |
| 括号总数 |  |  | √ |
| 代码深度 |  |  | √ |

这些度量元包括变量，运算符，数组，括号深度，括号总数，关键字，函数，类，逻辑，元素数，字数（字符串的长度），深度。其中变量varTotal表示当前语句中的变量总数；optTotal表示当前语句中运算符的数量；array表示当前语句中是否包含数组；bracketDepth表示当前语句中括号的层数；bracketTotal表示当前语句中括号的总数；keywordTotal表示当前语句中关键字的数量；methodTotal表示当前语句中包含的函数数量；typeTotal表示当前语句中包含类的数量；Logic表示当前语句中是否含有逻辑运算；lengthEle表示当前语句中包含元素的数量；lengthWorld表示当前语句的长度，包括所有空格，字符；depth表示当前语句在代码中的层数。

1. **实验与结果**

**A.**

DStar和MUSE是两种最流行的缺陷定位方法，通过检查前10个可疑元素，DStar可以定位大约43%的故障。MUSE可以通过检查前10个可疑元素定位约19%的故障。因此，我们选择DStar作为频谱方法的代表技术，MUSE作为变异方法的代表技术，使用两种方法产生的缺陷定位结果作为下一步数据集的组成。

我们提出CBFL方法实现了CDStar和CMUSE两种方法，方法分别结合了DStar和Muse。在我们的实证研究中，我们选择0.3%、0.5%、0.7%、1%、1.3%、1.7%、2%的语句作为重新排名的候选，以评估我们的方法。（为了证明以这个比值是有效的，画个图，解释以前百分之几作为数据集，能够有效的降低类不平衡问题）

为了评估该方法的有效性，本研究旨在回答以下研究问题:

RQ1: CBFL能否单独使用？怎么使用会更好？

RQ2: CBFL是否提高了缺陷定位结果的精确度？效果如何？

RQ3:与SBFL和MBFL相比，CBFL提升了多少？

**B.实验步骤**

**1）数据集（Datasets）**

为了回答研究问题，我们使用DStar和MUSE在Defects4j数据集上进行了缺陷预测。Defects4j是一个真实的故障数据集，由6个开源Java项目组成，是一个可重现错误的集合。对于每个缺陷，Defects4j提供了一个buggy版本源码和一个fix版本的源码，以及开发人员编写的测试用例集合。Defects4j数据集已经被应用到多个缺陷定位研究中，作为一个被众所周知且广泛使用的数据集，能够确保我们的结果能够具有普遍性。

表1总结了Defects4j基准的细节数据，它包含来自6个开源项目的395个缺陷版本:Chart(26)、Closure (133)、Lang(65)、Math(106)、Mockito (38)和Time(27)。在表2中，第一列是介绍项目名称，第二列是介绍该项目下有多少个版本的缺陷，第三列是每个项目的平均代码行数。第四列是每个项目的平均测试用例数量，每个缺陷版本包含一个或多个缺陷。

表1 **Defects4J(V1.4.0)基准的细节数据**

|  |  |  |  |
| --- | --- | --- | --- |
| Projects | Faults | LOC | Number of test cases |
| Chart | 26 | 7057 | 428 |
| Closure | 133 | 30660 | 351 |
| Lang | 65 | 1731 | 88 |
| Math | 106 | 7036 | 305 |
| Mockito | 38 | 4252 | 964 |
| Time | 27 | 3959 | 738 |
| Total | 395 | 48748 | 2874 |

**2）**类不平衡问题处理

在现实的分类学习任务中，经常会遇到类不平衡问题。类不平衡问题是指在分类问题中，各个类别的样本数据量差距较悬殊。在不平衡数据集中，少数类的数据量占比仅为数据总量的10-20%，甚至还会更少。用类别不平衡的数据集训练所得的模型，往往会对少数类别的预测精度(例如precision和recall等)较低，在数据严重不平衡状态下训练出来的模型对于少数类而言有效性极低。在数据类别不平衡的问题上，常用的解决办法大致分为两种，即对原始数据进行过采样或者欠采样。其中，过采法（Oversampling）一般是指增加原始数据中少数类的样本数量，反之，欠采样（Undersampling）一般指通过外部干预，减少原始数据中多数类的样本数量。

在构建数据集的过程中，我们发现缺陷预测数据集存在非常严重的类不平衡现象，为缓解类不平衡问题，本方法参考文献[]中使用软件缺陷预测领域中常见的采样方法对每个缺陷版本的预测结果进行处理。具体来讲，本方法中使用SMOTE方法对缺陷预测数据集进行过采样处理。

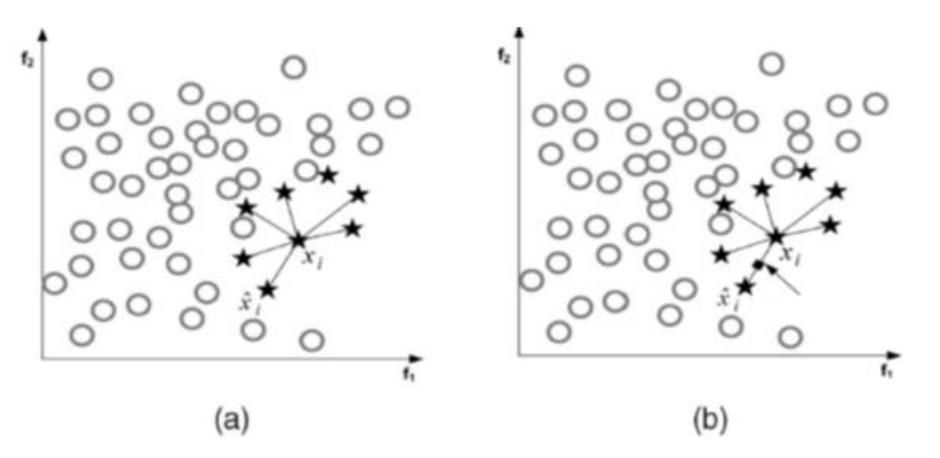
**（改成放我们的真实缺陷的比例）**

图 4 SMOTE 算法样本模拟

SMOTE(Synthetic minority oversampling technique)算法是实现过采样的一种方式，又叫做合成少数过采样算法。其工作原理不同于随机过采样，并不只是对原始样本中的少数类进行复制，而是通过分析该类样本数据的特点，而后根据现有样本特性模拟形成新样本，并将模拟的新样本添加到数据集中。因此SMOTE平衡数据也更具有代表性，被广泛用于解决机器学习中训练数据类别不平衡的问题。SMOTE算法新样本的模拟过程采用了KNN 技术[]。如图4中(a)所示，五角星为少数类别样本，其中一个样本点用xi表示，计算其K近邻，根据所需训练数据的平衡需求设置采样倍率N，从K近邻中随机挑选若干样本点x*j*，对于任意近邻xj可根据公式（3）进行新样本的创建，其中rand(0,1)指在0到1范围内的随机数。如图4中(b)所示，x*i*和x*j*之间的方块即通过SMOTE算法模拟出的新样本。

3）预测模型构建及度量

本文参考文献[ ]，使用软件缺陷预测领域中常用的5种二分类模型分别作为CBFL的缺陷倾向性预测模型。使用的缺陷倾向性预测模型有：高斯贝叶斯分类器（GNB）、K邻近分类器（KNNC）、决策树分类器（DTC）、随机森林分类器（RFC）和支持向量机分类器（SVC）。

实验的测试集选取，本文选择在生成的缺陷预测排名中分别取前0.3%、0.5%、0.7%、1%、1.3%、1.7%、2%作为测试数据集。对实验结果使用F1-score[]、EXAM、Einspect@n作为评估指标。

为了计算recall，需要使用如表2示的混淆矩阵对样本的预测结果进行评价，其中正样本被正确预测称为TP，负样本被错误预测称为FP，负样本被正确预测称为TN，正样本被错误预测称为FN。

表2 二分类问题的混淆矩阵

|  |  |  |
| --- | --- | --- |
| 预测样本  真实样本 | 正 | 负 |
| 正 | TP | FN |
| 负 | FP | TN |

根据表2中的混淆矩阵，召回率Recall的计算公式如公式（）所示。

精确率Precision的计算公式如公式（）所示。

本文采用了EXAM，Einspect@n来衡量SBFL，MBFL及CBFL的性能。在实际中，开发人员往往更关注排名靠前的可疑语句，当他们刚开始检查时，逐条尝试可疑度更高的语句，而当连续检查多个不是真实缺陷的语句时，程序员可能就会失去耐心并且开始跳跃着检查，甚至放弃检查。因此使用EXAM，Einspect@n作为评估指标更贴合实际。

EXAM是缺陷定位领域中常用的度量标准，它被定义为在第一个错误语句出现之前需要检查的语句的百分比。这里我们的主要目的是能够尽早的为开发人员提供真实的缺陷，所以当需要检查的语句越少，EXAM值越低，缺陷预测的效果就越好。

文章[]使用Einspect@n来评估缺陷预测方法的有效性，Einspect@n是通过检查排名前n个语句，并计算成功定位到的缺陷总数。在n越小的情况下，Einspect@n的值越大，则缺陷预测更有效。

上述的实验均采用8折交叉验证。具体而言，将原始的据集等分为8份，使用其中7份作为该轮模型的训练集，剩余的1份作为测试集，记录本轮模型的预测结果。如此重复8次，每次选择不同的测试，以此保证每份数据被作为过测试集1次。

一张展示用10折交叉验证和8折交叉验证后真实数据量过少的情况的图。

通常使用10折交叉验证，但根据本实验数据集实际情况，（如图\*\*\*\*），使用10折交叉验证会导致数据集中（真实数据）量过少，在单独计算recall和precision时，数据过于单一，无法在实验中确定模型优异。（大概这个意思，还得换点说话和润色一下）

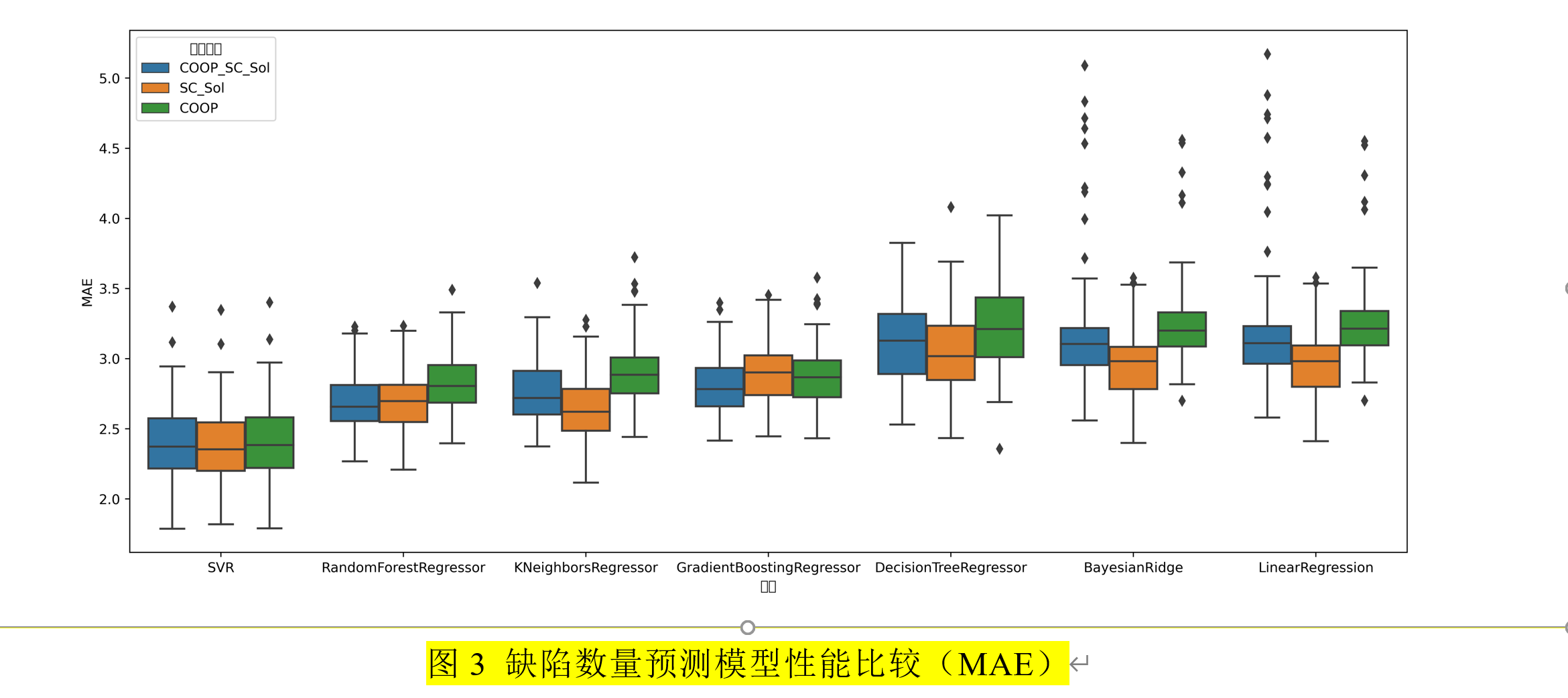
C．EXPERIMENTAL RESULTS AND ANALYSIS（实验结果与分析）

RQ1: CBFL的缺陷数量预测模型性能比较？（比较分类器的效果）

RQ2: CBFL能否单独使用？怎么使用会更好？

RQ3:与SBFL和MBFL相比效果如何？CBFL提升了多少？

介绍数据集对5种分类器进行了训练，并得到了结果，进行比较，使用recall进行比较，



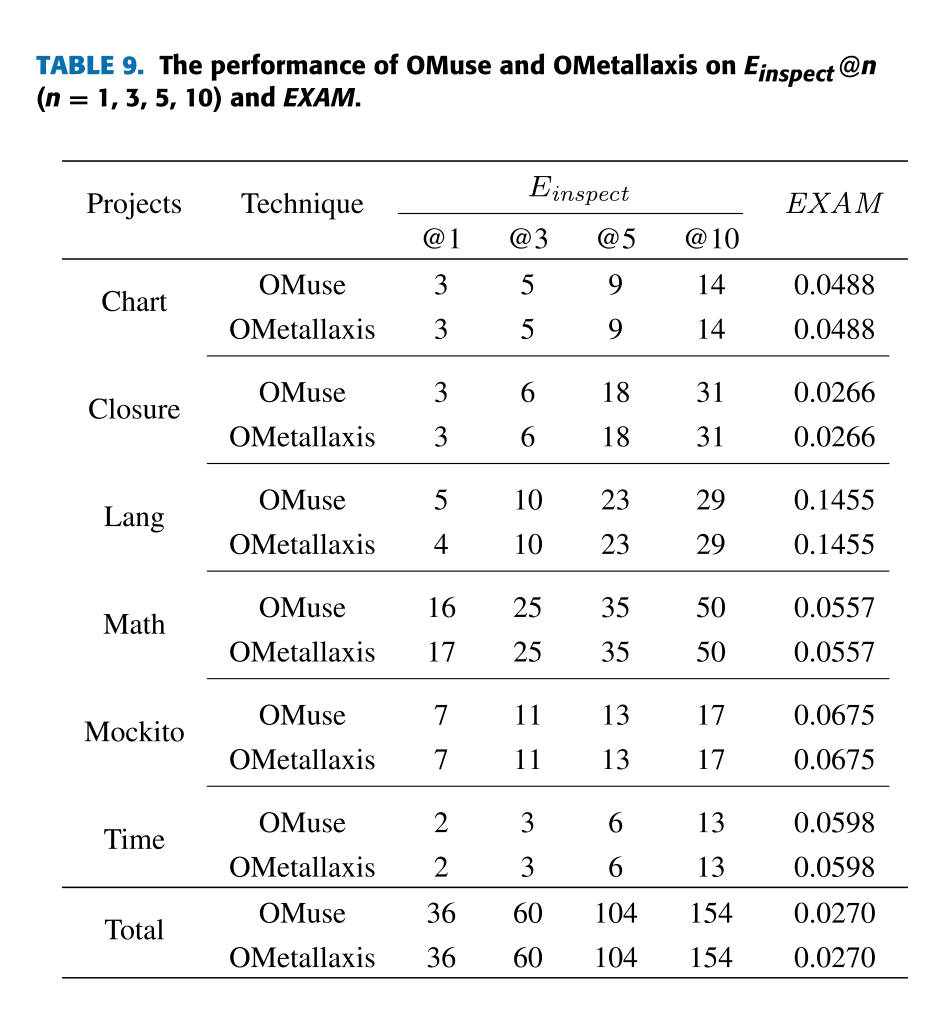
根据文章提出的结论，在缺陷结果中有多条语句具有相同的可疑度是很常见的，对于defects4j的6个项目，可疑度并列第1的语句平均数量是22，具体来说，平均有22条语句被DStar认为可疑度相同。

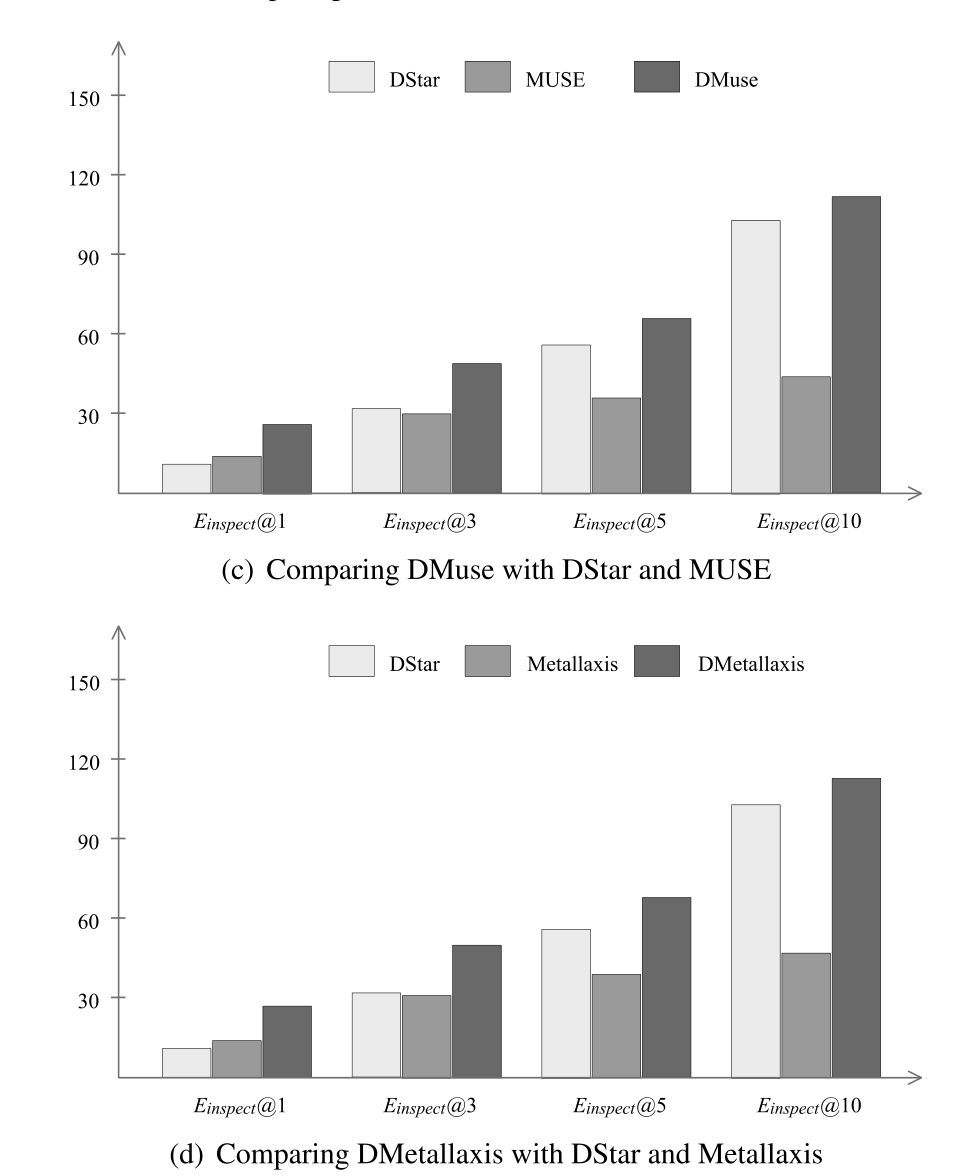
如表所示，（RQ2展示单独使用cbfl的实验结果，放出EXAM和Einspect@n）

（解释表格的信息）

RQ2未定

RQ3，比较DStar，MUSE和CDStar，CMUSE使用柱状图和表格的形式，比较，得到





CDStar，CMUSE，

Answer for RQ1:回答RQ1.

随后

1. **讨论（DISCUSSION）**

**根据研究结果表明，CBFL对。。。有效，讨论结果的有效性和方法的局限性**

把CBFL作为补充测试技术，与多种软件缺陷定位方法结合，提高缺陷定位的结果，并且因为训练模型的数据还不够多，尤其是目前使用了SMOTE来过采样，如果我们能够得到一个真实的类平衡的数据集去训练，是不是结果会更好。

举例说明，数据集的提高对实验结果的影响时很大的。得出我们的模型可以更好。

威胁有效性**Threats to Validity**

**局限和未来工作**

**数据集的局限性**

**未来是否可以把这个思想投入到APR工作中去**

1. **结尾（CONCLUSION）**

**根据本文概述，**现有缺陷定位方法中最突出的问题是缺陷定位效率不高，定位结果往往存在大量语句可疑度相同，难以区分的问题。因此本文提出基于语句复杂度分析的方法，旨在区别可疑度相同的程序语句，提高现有缺陷定位方法的定位效率，减少人工审查的代码量，从而降低软件项目的成本。