计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.250200024

自动化软件缺陷定位技术研究

房金秋　员国荣　赵海勇　谢皓萌
聊城大学计算机学院　山东 聊城　252000
(2022400825@stu.lcu.edu.cn)

摘　要　软件缺陷定位已经成为软件调试领域重要的研究主题,自动化软件缺陷定位旨在提高缺陷定位的自动化程度,帮助开发人员更高效地定位大规模软件中可能存在缺陷的位置,以达到优化分配测试资源的目的。以自动化软件缺陷定位技术为核心,对相关研究成果进行系统梳理。首先,依据软件缺陷处理生命周期,给出了缺陷定位的一般性研究框架,总结了每一类代表性算法利用的缺陷信息,讨论了深度学习对相关算法的影响。其次,针对软件系统包含缺陷数量的不同,从单缺陷和多缺陷两个角度对比了相关算法。最后,给出了相关算法常用的评测数据集以及评价指标,指出了本领域研究面临的一些挑战,并展望了若干值得进一步研究的方向。

关键词:缺陷定位;缺陷检测;软件工程;软件质量;深度学习
中图分类号　TP311

Advances in Automatic Software Defect Location Techniques

FANG Jinqiu,YUN Guorong,ZHAO Haiyong and XIE Haomeng
College of Computer Science and Technology,Liaocheng University,Liaocheng,Shandong 252000,China

Abstract　Software defect localization has become an important research topic in the field of software debugging,and automated software defect localization aims to improve the degree of automation of defect location,help developers locate possible defects in large-scale software more efficiently,so as to achieve the purpose of optimal allocation of test resources. With the defect location technology of automatic software as the core,the relevant research results are systematically sorted out. Firstly,according to whether it is necessary to run test cases,the positioning technology is divided into static positioning technology and dynamic positioning technology,the representative algorithms of each type of technology are summarized,and the influence of deep learning on related algorithms is discussed. Secondly,according to the difference in the number of defects contained in the software system,the related algorithms are compared from the perspectives of single defect and multiple defects. Finally,the commonly used evaluation datasets and evaluation indicators of related algorithms are given,some challenges in this field are pointed out,and some directions worthy of further research are prospected.

Keywords　Fault localization,Defect detection,Software engineering,Software quality,Deep learning

随着软件规模的不断增加,软件的复杂性也越来越高,软件开发过程中引入各类软件缺陷将不可避免。软件缺陷的存在会破坏程序的正常执行,使其在某种程度上不能满足用户的期望。严重的软件缺陷可能会给用户造成重大经济损失,甚至危及生命安全。因此,当检测到软件系统中存在缺陷时,如何在海量源代码中准确定位引发缺陷的位置就显得尤为重要。软件缺陷定位是后续软件修复工作的重要前提,其精度的提升直接影响缺陷修复的成功率。

传统缺陷定位方法主要依靠人工方式进行代码审计或者软件调试,测试人员在代码可疑位置设置断点,重复执行失败测试用例,不断缩小可疑代码的搜索范围,直至定位到缺陷代码。有实证研究表明,大约84%～93%的缺陷被定位到单个或者少数几个源文件[1]。由于现代软件规模巨大,往往包含成百上千个源代码文件,人工方式进行软件缺陷定位成本较高,而且费时费力,存在定位精度不高、缺陷信息利用不足等问题。因此,相关研究人员开始针对自动化软件缺陷定位技术展开深入研究,提出了各种自动化缺陷定位模型,以提高缺陷定位的效率与精度。

自动化软件缺陷定位(Automatic Software Bug Localization,ASBL)技术旨在最小化人为介入的情况下,自动识别出引发软件功能异常或结果偏离预期的代码区域,并进一步探究缺陷的根本原因。ASBL可以有效帮助开发人员节约修复成本,优化分配调试资源,进而提高软件质量。

目前,已有的ASBL技术按照是否需要执行测试用例,可

基金项目:国家自然科学基金(62402202);山东省自然科学基金(ZR2024QF036);山东省本科教学改革研究项目(M2023218);2024年国家级大学生创新创业训练计划项目(202410447004)
This work was supported by the National Natural Science Foundation of China(62402202),Natural Science Foundation of Shandong Province,China(ZR2024QF036),Shandong Province Undergraduate Teaching Reform Research Project(M2023218) and 2024 National College Students' Innovation and Entrepreneurship Training Program Project(202410447004).
通信作者:赵海勇(zhaohaiyong@lcu-cs.com)

分为动态缺陷定位技术和静态缺陷定位技术。动态缺陷定位技术通过插桩技术、执行监控[2]等方式收集测试用例执行的动态信息(如执行路径等),对被测程序的内在结构进行分析,以确定缺陷语句在被测程序内的可能位置。静态缺陷定位技术通过分析缺陷报告、程序源文件文本信息以及其他静态信息,提取缺陷特征并基于特定模型确定可疑的程序模块列表,将其推荐给开发人员以辅助定位。

动态缺陷定位技术基于测试用例的覆盖信息定位缺陷,通常可获得比静态缺陷定位方法更好的定位精度,但需要花费较多时间成本和资源成本去设计测试用例以及执行测试用例。更为关键的是,在软件维护阶段,开发人员有时仅能搜集到缺陷报告的信息,并不能获得足够数量的测试用例,而且测试用例的质量对缺陷定位性能的影响较大。静态缺陷定位技术由于不需要搜集测试用例的执行信息,不会受到程序规模和编程语言等因素的影响,具有缺陷定位成本低和方法可扩展性强等优点,其不足之处在于定位粒度比较粗糙,通常在文件或者方法级别。

2011 年,Yu 等[3]梳理了 1999－2011 年自动化缺陷定位技术的发展历史,并介绍了动态定位技术的原理和建模方法。2015 年,Chen 等[4]以该综述的分析为基础,对基于程序频谱的动态缺陷定位问题进行了较为全面的分析和梳理。近几年,随着静态定位技术的不断发展,也有越来越多关于基于信息检索的软件缺陷定位方法的综述[5-8]。而本文则力图从静态定位技术和动态定位技术两个方面,全面阐述本领域的研究进展。

本文第 1 章介绍了国内外相关文献检索以及筛选统计过程;第 2 章描述了 ASBL 的一般性研究框架,给出了研究所涉及的重要概念的形式化表述;第 3 章对动态和静态两类定位技术所涉及的经典算法进行了详细的分析与对比,并对深度学习在软件缺陷定位中的应用进行了一定的分析;第 4 章针对缺陷间的相互影响问题,介绍软件多缺陷定位技术,并将单缺陷与多缺陷定位技术进行对比;第 5 章介绍常用的评测数据集及性能评价指标;最后总结全文并展望未来。

1 文献检索

1.1 文献检索与筛选

ASBL 一直是软件调试领域的一个重要研究问题。为了对该问题的研究进展进行系统归纳总结和分析比较,本文搜集了最近 10 年(2015－2024)发表的主要研究文献,检索和筛选相关文献的过程如下。

1)搜索引擎及检索数据库:谷歌学术搜索引擎(Google scholar),ACM Digital Library,IEEE Xplore Digital Library,Springer Link Online Library,Elsevier ScienceDirect,DBLP Computer Science Bibliography 以及 CNKI 等。

2)检索关键词:英文关键词包括“bug localization”“fault localization”“defect localization”等;中文检索关键词包括:“软件缺陷”“缺陷定位”以及“软件调试”等。

3)文献筛选:首先对检索出来的文献进行初步筛选,通过分析论文的标题、摘要以及关键字,过滤掉不符合研究主题的文献;然后逐一检查剩余论文中的参考文献列表,避免遗漏未检索到的文献;最后进行二次筛选,只保留软件工程领域重要国内外学术期刊及会议相关论文。国内期刊包括:计算机学报、软件学报、计算机研究与发展、电子学报、计算机科学等。国际期刊和会议包括:TSE,TOSEM,IST,STVR,SQJ,ASC,EMSE,JSEP,JCST,ASE,ICSE,ISSTA,ESEC/FSE,OOPSLA,SIGIR,SANER,ESEM,ICST,ICSM/ICSME,ICPC,COMPSAC,MSR,WCRE,APSEC,QRS 等。

1.2 文献汇总

初步文献检索经过主题筛选和发表年份筛选,得到缺陷定位相关的外文文献 858 篇,符合要求的中文学术文献 70 篇。在中文文献中,发表在计算机科学期刊上的论文有 5 篇,软件学报期刊上的论文有 18 篇,计算学报期刊上的论文有 3 篇,电子学报期刊上的论文有 6 篇,其他学术期刊上的论文有 38 篇。从 2015－2024 年每年发表的文献总数量统计如图 1 所示。图 1 中的数据显示,近十年间,与缺陷定位相关的外文研究论文数量基本呈增长趋势,这反映出该领域在软件工程界持续受到高度关注;相关中文文献发表数量呈现波动,但是涉及缺陷定位的其他主题论文,例如缺陷预测、缺陷报告以及缺陷修复等,发表数量总体呈上升趋势。这也在一定程度上反映出软件缺陷是软件工程领域一个重要的研究主题。

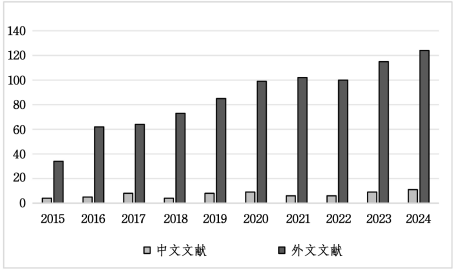

图 1 缺陷定位文献数量统计(电子版为彩图)

Fig. 1 Statistics on number of defect localization literatures

2 研究框架

根据 IEEE1044-2009 标准定义[9],软件缺陷指软件产品中存在的,使产品无法满足软件需求及其规格要求,需要修复的瑕疵、问题。软件缺陷主要指软件中存在的 bug 或者 defect,其产生的根源是开发人员在软件开发过程中无意犯下的技术错误,如果不修复,则将一直存在。软件缺陷在运行时可能触发软件故障,最终以软件失效的形式显现。因此,软件缺陷概念涵盖软件错误、软件失败以及软件故障。软件缺陷定位通过分析已经发生的软件缺陷现象,给出导致缺陷的代码位置。缺陷定位越精确,为后续软件修复提供的价值越高。

在软件系统的生命周期中,潜在的缺陷往往在测试阶段或实际使用过程中才逐渐显现。缺陷定位过程本质上是对缺陷相关信息的深入挖掘,而挖掘的信息种类和深度不同,导致了多样化的缺陷定位策略的产生。如果缺陷发生时,用户或者开发人员通过缺陷追踪系统提交了缺陷报告,开发人员可以通过挖掘缺陷报告以及源代码相关信息实现缺陷定位,此类定位技术称为静态缺陷定位技术。如果缺陷发生时,开发人员可以执行事先精心设计的测试用例,事后就可以通过挖掘程序运行时动态信息以及测试结果实现缺陷定位,此类定位技术称为动态定位技术。每种方法都有其优势和局限性,因此在实际应用中,通常会结合多种方法来提高缺陷定位的准确性和效率。常见缺陷定位技术分类如表 1 所列。

表 1　软件缺陷定位方法分类

Table 1　Classification of software bug localization methods

分类	方法名	利用信息
动态定位技术	基于程序切片的方法	动态程序切片
	基于覆盖的方法	覆盖信息和测试用例
	基于统计的方法	谓词
	基于模型的方法	测试用例、程序执行结果、程序代码特征以及可能的程序切片信息
	基于变异的方法	测试用例和程序变异体执行结果
	基于频谱的方法	程序谱
静态定位技术	基于程序切片的方法	静态程序切片
	基于信息检索的方法	错误报告

ASBL 技术主要解决两个科学问题：1）如何找到缺陷发生的位置；2）如何理解缺陷产生的原因。静态定位技术的输入主要是包含缺陷的源代码文件以及描述缺陷相关信息的缺陷报告，输出是一个可疑程序实体列表，定位过程主要采用相关模型（例如信息检索模型、深度学习模型等）来提取源代码文件特征和缺陷报告特征并进行匹配，根据匹配结果判定可疑程序实体；动态定位技术的输入主要是测试用例，输出仍然是一个可疑程序实体列表，定位过程主要通过分析程序运行时信息以及运行结果判定可疑程序实体。不同的定位技术的定位精度不一样，反映定位精度常用的指标是定位粒度，包括文件（类）粒度、包粒度、函数粒度、语句粒度和代码变更粒度。缺陷定位精度通常会受到多种因素的影响，例如源代码中缺陷的数量、用于定位的测试用例的质量以及缺陷报告的质量等。

围绕 ASBL 技术需要解决两个科学问题，缺陷定位的研究框架如图 2 所示。

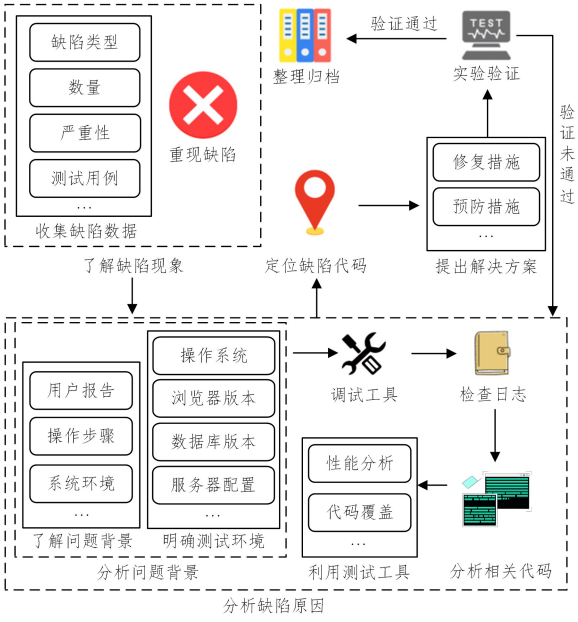

图 2　ASBL 研究框架

Fig. 2　Research framework of ASBL

定义 1（缺陷报告）　缺陷报告是一种半结构化文本文档，用于描述软件缺陷相关信息。缺陷报告包含的结构化信息有缺陷的状态、缺陷报告人、缺陷所属的产品和组件、缺陷的优先级和严重程度等；包含的非结构化信息有缺陷的问题概要描述、问题详细描述以及相关讨论等。

定义 2（测试用例集）　在程序测试过程中，为了尽可能揭示程序中存在的缺陷，测试人员需制定一系列精心设计的测试用例，把这些测试用例的集合叫作测试用例集。测试用例集可记为 $T=\{t_k=(i_k,o_k)\mid 1\leqslant k\leqslant m\}$，其中每一个 $t_k=(i_k,o_k)$ 表示一个成功测试用例或者失败测试用例，i_k 表示测试用例 t_k 的输入，o_k 表示测试用例 t_k 的输出。在特定输入条件下，每个测试用例都会产生一个预期输出。当测试用例的实际输出与预期输出一致时，该测试用例被认为是成功的，否则为失败测试用例。为了实现缺陷定位，动态定位技术需要运行成功测试用例和失败测试用例，以搜集程序运行时的相关信息，例如语句覆盖信息、分支覆盖信息、数据流信息以及执行路径等。

定义 3（程序频谱）　程序频谱指程序执行代码过程中生成的覆盖信息及其对应的测试结果，包含代码元素（如语句、分支）在测试用例运行时的覆盖状态。基于程序频谱的缺陷定位技术就是利用这些信息来实现缺陷定位的。程序频谱可表征为向量形式，其元素关联程序特定结构（如语句、分支、函数等），数值反映对应结构在测试中的覆盖频率。假设一个程序包含 n 个可测试的代码单元，定义程序频谱 $\boldsymbol{S}$ 为长度 n 的向量，其中每个元素 S_i 表示第 i 个代码单元被测试用例执行到的次数。该频谱可形式化定义为：$\boldsymbol{S}=[S_1,S_2,S_3,\cdots,S_n]$，其中 S_i 是一个非负整数，表示第 i 个代码部分被测试用例覆盖的次数。如果一个代码部分从未被覆盖，那么 $S_i=0$；如果被覆盖了 k 次，那么 $S_i=k$。

定义 4（程序切片）　程序切片包括静态切片和动态切片。静态切片依赖于程序的静态属性（如代码架构和变量间的依赖链）来构建，这种切片往往覆盖广泛的程序代码区域，因此它并不擅长精确地定位到具体的缺陷位置；动态切片是基于程序的实际输入和执行路径生成的切片，考虑了程序的动态信息和动态行为，使得动态切片的结果更具有针对性，准确性更高。用集合和函数的概念形式化地表示程序切片，程序切片 S_R 可表示为 $S_R=\{s\in S\mid \exists v\in V,t\in\{1,2,\cdots,n\},s\in D(v,t)\wedge(v,t)\in R\}$，其中将一个由一系列指令组成的程序设为 P，程序 P 中所有的变量集合为 V，所有的语句集合为 S，R 为切片准则，D 为数据依赖关系，C 为控制依赖关系。切片准则 R 是一个或者多个程序在执行过程中感兴趣的变量 v_i 和时间 t_i 的集合，$R=\{(v_1,t_1),(v_2,t_2),\cdots,(v_m,t_m)\}$。数据依赖关系 D 为一个描述程序中变量之间数据流的函数，可表示为 $V\times S\to 2^S$，其中 $D(v,s)$ 是在语句 s 之前影响变量 v 值的语句集合。另外，还有描述程序语句之间控制流的函数 C，可以表示为 $S\to 2^S$，其中 $C(s)$ 是所有必须执行语句的集合。

定义 5（定位粒度）　程序源代码 P 往往由若干程序实体 s 组成，可记为 $s=\{s_1,s_2,\cdots,s_k\}$，程序实体可以是一条程序语句或者一个程序块（例如分支、函数、类等）。定位结果往往是一个或者多个按照可疑度排序的程序实体。程序实体的大小决定了定位粒度。

3　常见软件缺陷定位算法

目前，常见软件缺陷定位算法可分为动态测试执行与静态程序分析两类方法。本章对两种算法类别进行了系统的归纳，介绍和讨论了近年来的经典算法，并在 3.3 节中结合深度学习在软件缺陷定位领域中的应用，对相关方法进行了分析。

3.1　静态软件缺陷定位

3.1.1 基于静态程序切片的缺陷定位

基于程序切片的缺陷定位（Fault Localization Based on Program Slicing）方法通过删除与特定变量和语句对相关性较低的部分，将程序抽象为一个简化的形式，从而减少了需要

分析的代码量，即减少程序员在试图定位其程序中的特定缺陷时需要经历的搜索域。程序切片技术包括静态切片技术和动态切片技术。前者可能会包含不应包含的语句的切片，因此利用静态切片处理故障定位的效率不高。而动态切片可以识别在特定位置观察到的特定值影响的语句，因此减少不必要的语句，因此和前者相比，切片规模更小，精度更高。

Weiser[10]最早提出程序切片的概念，文献[10]中的技术属于静态切片技术。静态切片技术将程序划为多个部分，依据源代码分析确定切片内容并定位缺陷。然而，由于所得程序集合较为宽泛，其精确性受到一定影响，因此静态程序切片缺陷定位方法具有一定的局限性。

3.1.2 基于信息检索的缺陷定位

基于信息检索的软件缺陷定位技术(Information Retrieval-based Bug Localization，IRBL)凭借其外部依赖少、计算成本低、普遍性和针对性更强等优点[5]，已成为该领域的研究焦点，并取得了显著的进展。这类方法基于历史已解决的缺陷报告以及其对应源代码构建数据集，通过挖掘缺陷报告和代码之间的相关特征来建立模型，用于分析新的待解决缺陷报告，最终生成一份按照程序实体与缺陷相关度降序排序的列表，其研究框架如图 3 所示。

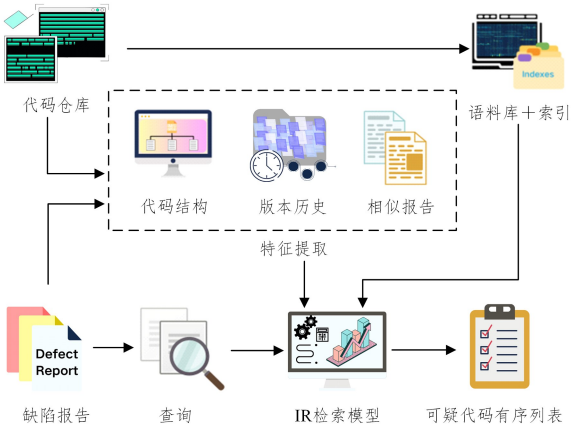

图 3 IRBL 研究框架

Fig. 3 Research framework of IRBL

近年来，为提高检索结果的准确性，研究人员通过进一步的特征分析来改良 IRBL 模型。表 2 列出了常用的提取特征以及使用对应特征的代表性文献。Wang 等[11]首次提出利用版本历史、相似报告、代码结构、堆栈踪迹和报告者信息 5 种特征进行缺陷定位的方法 AmaLgam+。其中，该方法引入了缺陷报告中的报告者信息来提升定位性能。他们提出，报告者倾向于报告与相同或相似的代码组件有关的缺陷。因此，当收到新的缺陷报告时，查看该报告者之前提交的缺陷报告和对应的包含缺陷文件有助于对新缺陷的定位。Shi 等[12]将利用文本相似度特征以外的其他特征(如版本历史、相似报告、代码结构等特征)来进行缺陷定位的方法定义为混合缺陷定位，在总结前人工作的基础上，他们比较了 8 种学习排序技术在混合缺陷定位中的性能，为后续研究提供了参考和指导。Yue 等[13]为提高定位粒度，提出了一种基于历史缺陷信息检索的语句级软件缺陷定位方法 STMTLocator。该方法通过对比历史缺陷报告与被测程序的相关信息，提取并分析相似语句来定位缺陷语句。Yang 等[14]提出了一种上下文感知程序简化技术 COPS，它通过分析数据依赖关系，有效地利用堆栈踪迹信息来跟踪潜在的缺陷代码实体，是第一个可用于 Python 项目的语句级缺陷定位方法。

由于代码库和缺陷报告的特征已经被广泛研究，提取新特征以改进 IRBL 方法变得困难。因此，研究人员提出了利用查询重构来提升 IRBL 的性能。这种技术不依赖于特定的 IRBL 方法，能够作为预处理步骤被整合到现有的方法中，具有良好的可移植性。Rahman 等[15]提出了一种名为 BLIZZARD 的上下文感知查询重构方法。他们从堆栈踪迹中提取代码实体，包括类名和方法名，依据执行顺序构建权重图，并应用 PageRank 算法确定每个代码实体的权重，以此来重构查询语句。Kim 等[16]提出了基于信息检索的缺陷定位自动查询重写的新方法，通过添加附件来扩展缺陷报告的信息，去除情感词汇以及在伪相关反馈过程中移除噪声文件来提高查询的质量。实验结果表明，该方法在所有评估指标上都取得了比 BLIZZARD 方法更好的表现，不仅提高了查询质量，还通过这些改进的查询提高了 IRBL 的准确性，具有显著的实际应用价值。

表 2 IRBL 常用提取特征

Table 2 Common features of IRBL

特征	描述	参考文献
版本历史	源代码实体的修订历史记录	[11,13,17,18,19,20,21]
相似报告	以前修复的缺陷报告，它们的文本形式与当前的缺陷报告相似	[11,17,18,20,21,22]
代码结构	源代码包括各种不同的代码实体，例如类名、方法名等	[11,23,24,25,26]
堆栈踪迹	缺陷产生时代码执行的异常信息	[11,14,15,16,17,27,28]
执行信息	测试用例的覆盖信息和执行结果	[22,29,30]
依赖关系	源代码方法和类的依赖关系图	[14,21,31]

3.2 动态软件缺陷定位

3.2.1 基于程序频谱的缺陷定位

基于程序频谱的缺陷定位(Spectrum-Based Fault Localization，SBFL)是一种典型的动态定位技术，具有计算复杂度低、成本低且适用范围广的特点[32-33]。如图 4 所示，它通过记录程序实体的调用信息和测试用例的执行结果，利用这些信息，使用度量公式来计算程序实体的可疑度，可疑度越高说明程序实体越有可能存在缺陷。

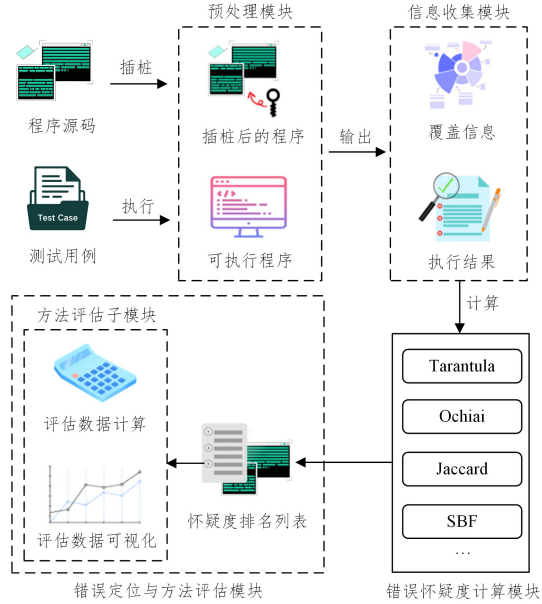

图 4 SBFL 研究框架

Fig. 4 Research framework of SBFL

在 SBFL 中，任意一个程序实体 e 在测试用例下的覆盖特征都可以用一个四元组表示[4]，如式(1)所示。

$$N(e)=\langle n_{cf}(e),n_{uf}(e),n_{cp}(e),n_{up}(e)\rangle \tag{1}$$

其中，这 4 个元素分别表示执行失败且覆盖 e 的测试用例个数、执行失败且未覆盖 e 的测试用例个数、执行成功且覆盖 e 的测试用例个数和执行成功且未覆盖 e 的测试用例个数。则测试套件 T 中，失败测试用例总数为 $n_f=n_{cf}(e)+n_{uf}(e)$，成功测试用例总数为 $n_p=n_{cp}(e)+n_{up}(e)$，测试用例总数为 $n=n_f+n_p$，每一个程序实体 e 的四元组之和等于测试用例总数 n。

近年来，在该四元组的基础上，研究人员陆续提出一些怀疑度计算式。表 3 列出了几个常用的 SBFL 怀疑度度量式。其中，Jones 等[34-35]首次提出 Tarantula 怀疑率计算式，如式(2)所示。文中指出，被更多失败测试用例覆盖的程序实体比被更多成功测试用例覆盖的更可能存在缺陷。对此，他们提出了一种利用覆盖信息和测试数据来识别缺陷的方法，并开发了可视化工具 Tarantula。

$$Tarantula(e)=\frac{\frac{n_{cf}(e)}{n_f}}{\frac{n_{cf}(e)}{n_f}+\frac{n_{cp}(e)}{n_p}} \tag{2}$$

Abreu 等[36]受聚类分析研究领域的启发，提出利用 Jaccard 相似度系数来计算程序实体的怀疑度，Jaccard 计算式如式(3)所示。

$$Jaccard(e)=\frac{n_{cf}(e)}{n_f+n_{cf}(e)} \tag{3}$$

后来，Abreu 等[37]又受分子生物学研究领域的启发，提出利用 Ochiai 相似度系数来计算程序实体的怀疑度，Ochiai 计算式如式(4)所示。

$$Ochiai=\frac{n_{cf}(e)}{\sqrt{n_f\times(n_{cf}(e)+n_{cp}(e))}} \tag{4}$$

在实证研究中，他们发现，采用 Ochiai 公式时的缺陷定位效果优于 Jaccard 公式和 Tarantula 公式。

早期研究仅依赖失败的测试用例进行基于频谱的缺陷定位，后续研究者开始同时考虑成功和失败的测试用例，并比较两者的差异，从而获得了更优的结果[38]。然而，现有的基于频谱的缺陷定位技术仅关注语句的执行信息，未能充分考虑语句的执行顺序和特征。因为机器学习能够从各种特征的角度分析数据对象，所以许多研究人员受到启发，将机器学习技术用于自动缺陷定位。2015 年，Naish 等[39]指出，评估度量的巨大搜索空间使现有的遗传规划方法变得效率低下且具有不稳定性，为此引入“hyperbolic”度量的限定类，此类包含少量数值参数。这使得遗传规划算法能够在广泛的程序频谱数据上可靠地识别出有效度量，进而提高了程序组件排序的准确性。Wong 等[40]将 BP 神经网络技术与程序谱相结合。它们以覆盖矩阵的形式表示程序语句的覆盖信息，这与测试用例的结果向量相对应。矩阵和向量分别作为训练数据和标签输入到神经网络中。此外，2022 年，Li 等[41]提出一种基于频谱的深度神经网络缺陷定位方法 Deep-SBFL。该方法首先通过收集深度神经网络的神经元输出信息和预测结果作为频谱信息；然后将频谱信息进行处理作为贡献信息，以用于量化神经元对预测结果所做的贡献；最后提出了针对深度神经网络缺陷定位的怀疑度公式 DeepDStar，基于贡献信息计算深度神经网络中神经元的怀疑度并进行排序，以找出最有可能存在缺陷的神经元。

表 3　SBFL 常用怀疑度计算公式

Table 3　Common formula for calculating suspicion in SBFL

公式名称	公式表达式
Tarantula	$\frac{n_{cf}(e)/n_f}{n_{cf}(e)/n_f+n_{cp}(e)/n_p}$
Jaccard	$\frac{n_{cf}(e)}{n_f+n_{cf}(e)}$
Ochiai	$\frac{n_{cf}(e)}{\sqrt{n_f\times(n_{cf}(e)+n_{cp}(e))}}$
Dstar3	$\frac{n_{cf}(e)^3}{n_{uf}(e)+n_{cp}(e)}$
Wong2	$n_{cf}(e)-n_{cp}(e)$
Barinel	$\frac{n_{cf}(e)}{n_{cf}(e)+n_{cp}(e)}$
Dice	$\frac{2n_{cf}(e)}{n_{cf}(e)+n_{uf}(e)+n_{cp}(e)}$
Ample2	$\frac{n_{cf}(e)}{n_{cf}(e)+n_{uf}(e)}-\frac{n_{cp}(e)}{n_{cp}(e)+n_{uf}(e)}$
Kulczynski2	$\frac{1}{2}\left(\frac{n_{cf}(e)}{n_{cf}(e)+n_{uf}(e)}+\frac{n_{cf}(e)}{n_{cf}(e)+n_{cp}(e)}\right)$

对于基于程序频谱的缺陷定位方法，由于程序谱仅考虑了每条语句的执行情况，并没有将上下文语句之间包含的依赖关系进行分析，因此有时会出现定位到的结果并不是真正的缺陷语句，而是因缺陷传递而产生的非预期运行结果的相关语句。另外，程序谱的构造需要大量测试用例，对测试用例的要求也相对较高。

3.2.2　基于程序变异的缺陷定位

基于程序变异的缺陷定位(Mutation-Based Fault Localization，MBFL)是一种使用变异算子[42]对原始程序进行突变，生成一系列变异体，然后利用变异体模拟含缺陷程序元素对程序运行的影响，进而通过可疑度公式计算得到含缺陷程序元素的可疑度，最终确定最有可能包含软件缺陷的代码位置。2014 年，Moon 等[43]提出一种利用突变体特征差异实现缺陷定位的新型缺陷定位技术 MUSE。2015 年，Papadakis 等[44]提出利用各个变异体间的相似性来检测缺陷的方法 Metallaxis，并证明了基于变异分析的缺陷定位方法的有效性。与传统的基于代码覆盖率的方法相比，该方法更加准确地定位了程序中的缺陷，并且只需要很少的变异就可以实现。同年，Hong 等[45]提出了一种适用于多语言程序的变异算子，旨在使基于变异的缺陷定位技术对多语言程序缺陷定位更加准确。

基于程序变异的定位技术能够通过生成变异体而精确地识别出含有缺陷的语句，准确率比其他技术更为优越。然而，大量的程序变异体验证测试用例集，导致计算成本相对较高。为了解决此问题，Liu 等于 2017 年提出了两项策略：面向语句的变异体减少策略(SOME)[46]和动态突变执行策略(DMES)[47]，旨在降低 MBFL 的计算负担。次年，Oliveira 等[48]则提出了一种面向失败测试的变异执行策略(FTMES)，旨在提升 MBFL 的效率并减小计算开销。FTMES 策略聚焦于用失败的测试用例集执行变异，并利用覆盖率数据优化执行过程，从而避免已通过测试用例的重复执行。2021 年，Kim 等[49]提出了 SIMFL(Statistical Inference for Mutation-based Fault Localization)，允许用户在观察到故障之前提

前进行突变分析，从而可以摊销分析成本。2023 年，Xia 等[50]提出了 FastMBFL 方法，该方法通过考虑来自失败测试的冲击信息的重要性，同时保持相似的准确性来提高 MFBL 的效率，降低 MFBL 的成本。

近年来，不少研究人员将深度学习与基于程序变异的缺陷定位相结合，取得了较好成效。2021 年，Jeon 等[51]提出一种新的突变算子，利用神经网络生成合理的代码元素，以提高基于突变的故障定位对遗漏故障的性能。2023 年，Ghanbari 等[52]在 DNN 模型的背景下重新审视了基于突变的故障定位，并提出了一种名为 deepmufl 的新技术，适用于广泛的 DNN 模型。

3.2.3 基于动态程序切片的缺陷定位

2007 年，Zhang 等[53]通过多切片定位代码缺陷，基于后向动态切片(BWS)又提出了两种动态切片，即前向动态切片(FWS)和双向动态切片(BIS)。虽然动态切片规模比静态切片小得多，但是在实际软件调试中，代码规模仍然可能很大。因此，研究者提出了许多动态切片的改进算法来减小动态切片的代码规模。Zhang 等[54]提出了 3 种精确的动态切片算法，即完全预处理(FP)、没有预处理(NP)和限制预处理(LP)，通过实证分析得出 LP 能大大降低切片规模和时间消耗。后期，Zhang 等[55]又提出了一种有效的关键谓词自动定位技术，通过在程序运行时切换谓词的结果来修改程序的状态，从而实现定位效果。他们发现，双向动态切片方法的关键谓词可以缩小切片规模，从而更好地定位缺陷。因此，他们开发了基于动态插装的实验方法，并通过多项测试证明了该技术的实用性。他们首先求得能够让失败测试用例执行转变为成功测试用例执行的关键谓词；其次，利用 Delta Debugging 方法求得导致程序执行失效的最小输入；然后，计算关键谓词的后向动态切片和最小输入的前向动态切片的交集，以及关键谓词的前向动态切片和失效输出的后向动态切片的交集；最后，将这两个交集的并集作为关键谓词的双向切片。他们通过实证分析得出，双向动态切片在缺陷定位中具有更高的效率。2021 年，Ezekiel 等[56]将基于频谱的缺陷定位技术与动态切片技术进行了有效性评估，发现动态切片对于单缺陷的程序更有效，但基于频谱的缺陷定位技术在多缺陷定位上的表现更好。然而，最好的结果是通过混合方法获得的。2023 年，Peter 等[57]将基于程序频谱的缺陷定位与程序切片这两种技术相结合，对故障定位的潜力进行全面检查，以试图更有效地识别软件缺陷。

3.2.4 基于覆盖的缺陷定位

基于覆盖的缺陷定位技术(Coverage-Based Defect Localization)是一种动态缺陷定位方法[58-61]。首先，搜集代码覆盖信息和测试用例的执行结果；然后，设计度量公式对每个程序实体的可疑度进行可疑度评估；最后，按照可疑度值从高到低对程序实体进行排序，以确定最可能存在缺陷的代码区域。这种定位技术能够有效缩小定位的范围，从而提高缺陷定位的效率。

基于覆盖分析的缺陷定位技术的计算复杂度低、可自动化程度高，受到广泛的研究和关注。Nashi 等[62]通过建立模型和进行了一系列的实验，对 36 种覆盖分析方法进行了深入分析，这些方法各自采用不同的度量公式。排名结果表明 Optimal 的指标总体优于其他方法，更能有效地实现缺陷定位。2013 年，Xie 等[63]提出了一种创新的理论研究框架，旨在提高软件缺陷定位的准确性，然后进一步通过蜕变技术将软件测试分析技术和缺陷定位过程相结合，使得缺陷定位技术更能贴合实际的开发环境和场景。Wong 等[64]对代码语句和被执行的测试用例之间做了如下假设：一条语句存在缺陷的可疑度和它被成功测试用例执行的频率呈负相关关系，和它被失败测试用例执行的频率呈正相关关系。简单来说，若某语句在成功测试用例中很少出现，而在失败测试用例中频繁出现，则其存在缺陷的概率增大，可疑度就会随之提升。2021 年，Lou 等[65]提出一种新的基于覆盖率的缺陷定位技术——GRACE。此方法充分利用了覆盖信息和图表征学习，用图形结构来表示覆盖率，然后又通过门控图神经网络进行特征学习，生成程序实体的有序列表。利用门控图神经网络从基于图的覆盖率表示中学习有价值的特征，并以列表方式对程序实体进行排名。通过实验测试发现，GRACE 有非常好的定位效果。在跨项目预测场景评估中，GRACE 的一致性优于最先进的基于覆盖的缺陷定位技术。

由于基于覆盖的缺陷定位技术是通过搜集程序实体的覆盖信息和执行结果，然后计算相关性和可疑度来定位缺陷，因此该方法对测试用例的数量和质量十分严格，如果测试集中的测试用例数目过少则会导致语句覆盖不全面的情况；若测试用例数量过多且质量不佳，将引发大量冗余覆盖，致使程序语句被反复执行，进而削弱定位的准确性。此外，由于缺少对软件执行状态及其转移的分析，覆盖分析方法无法有效解决缺陷在程序体之间的传播问题，故基于覆盖的缺陷定位技术应考虑程序实体间的数据流或者控制流依赖关系，以提高定位的精度。

3.2.5 基于模型的缺陷定位

基于模型的缺陷定位(Model-Based Fault Localization，MBFL)方法是基于覆盖信息定位的一种扩展方法，在利用覆盖信息进行缺陷定位时，该方法通过预先定义好的模型来辅助预测和定位软件缺陷。

Reiter 等[66]首次提出基于模型定位缺陷的概念。他们首先定义了一个独立域的概念系统，目的是将组件的概念以及相互作用的组件集合的概念尽可能抽象地形式化；然后对于具有观测值 OBS 的系统(SD，COMPONENTS)，提出一种观测方法，并使用了包含(SD，COMPONENTS，OBS)的三元组结构来形式化地定义。其中，SD 是系统描述，通常用一组一阶逻辑语句描述程序内各部分之间的关系及行为；COMPONENTS 是系统组件，通常为一组常数；OBS 则定位为根据输入得到的观测值，是有限一阶语句的子集合。DeMillo 等[67]认为故障和分析故障产生的结果有利于软件调试的过程，因此提出了一种故障分析模型，用于软件故障分析和调试。该方法在模型中定义了故障模式和故障类型，模型的流程可概括为：将某一故障识别为属于故障类型中的其中一种，根据故障模式与故障类型的关系，找出可能导致故障的故障类型，然后根据不同情况采用启发式算法进行故障定位。2005 年，Cleve 等[68]提出了 Delt Debugging 自动策略，分别从空间和时间上进行搜索，通过对比识别成功测试用例和失败测试用例之间的差异，来实现对缺陷的定位。2020 年，Ju 等[69]提出了一个名为 Mulr4FL 的框架，用于多变量回归模型进行缺陷定位，该模型结合了从调试中的程序中收集的静

态和动态特征，以提高缺陷定位的效率。

由于现有基于模型的缺陷定位技术中，未考虑非相邻节点间传递依赖和测试用例对可疑度的影响，导致缺陷定位精度和效率低。2021 年，Ren 等[70]提出了基于概率模型检测的软件缺陷定位方法(SFL)。该方法首先提出一种程序概率模型用于提高模型的推理能力；然后设计了基于执行路径构建程序概率模型的学习算法；最后设计了基于概率模型检测的软件缺陷定位算法，用于缺陷定位分析。

3.2.6 基于数据挖掘的缺陷定位

基于数据挖掘的缺陷定位技术通过从大量程序运行信息数据中提取相关信息来生成模型，数据挖掘可以发现数据样本中隐藏的模式，同时可以在合理的时间内处理大规模的数据，提高了缺陷定位的效率。其定位方法研究框架如图 5 所示。

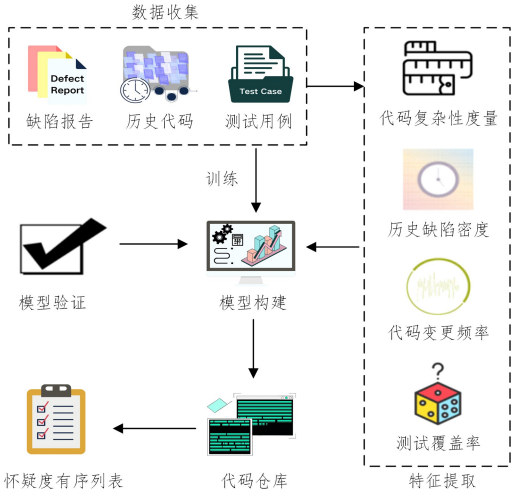

图 5 基于数据挖掘的缺陷定位方法研究框架

Fig. 5 Research framework of data mining-based defect localization

在数据挖掘领域，Apriori 算法被用于挖掘缺陷间的潜在关联性，旨在提升缺陷定位的精确度和效率。该算法通过评估支持度(Support)和置信度(Confidence)来量化规则的强度，是关联规则重要性的衡量准则，计算式如下：

$$Support(X,Y)=P(XY)=\frac{number(XY)}{num(AllSamples)} \tag{5}$$

其中，$Support$ 表示规则在所有交易中出现的频率；X 和 Y 是变量集合。式(5)表示为包含项集 X,Y 的事务数量与事务集总数 $AllSamples$ 的比值，反映了 X,Y 项集同时出现的频率。

置信度(Confidence)指在一种规则 X 出现的情况下，规则 Y 出现的条件概率。其计算式如下：

$$Confidence(X\rightarrow Y)=\frac{Support(X\cup Y)}{Support(X)} \tag{6}$$

其中，$Support(X\cup Y)$表示规则 X 和规则 Y 同时出现的支持度，$Support(X)$表示规则 X 出现的支持度。

Denmat 等[71]通过分析执行轨迹，开发了一种基于关联规则的缺陷定位技术，该技术利用支持度和置信度指标来筛选与错误相关联的代码语句。

Wong 等[72]采用优化后的径向基神经网络，分析了语句覆盖信息与相应的测试结果(成功或失败)间的联系，进而提出了一种基于神经网络的软件缺陷定位方法。Cellier 等[73]运用数据挖掘技术中的关联规则和概念分析，旨在提高缺陷定位效率。他们通过分析语句覆盖率与相关执行失败的测试用例之间的关联性，统计各规则出现的频次，构建规则矩阵。最后，生成一个可疑度排名来定位缺陷。Zhang 等[74]提出一种基于马尔可夫逻辑的数据挖掘缺陷定位技术，融合了一阶逻辑和加权马尔可夫随机域，用于感知器算法的学习。

3.3 基于深度学习的缺陷定位

近年来，深度学习技术不断进步和突破，作为人工智能的一个重要分支，已经在图像识别、自然语言处理、语音识别等多个领域取得了显著的成就，众多研究者开始探索将其应用于软件缺陷定位领域。与主要依赖于缺陷报告和源代码文本特征的信息检索式缺陷定位方法不同，基于深度学习的方法利用深度学习模型强大的特征学习和模式识别能力，从软件的各种信息(如代码结构、测试用例执行结果、运行日志等)中自动挖掘与软件缺陷相关的特征和模式，进而确定缺陷在软件系统中的位置(如代码中的具体行、函数、模块等)。这类方法不仅降低了特征设计的人工劳动强度，还提升了缺陷定位的精确度。Polisetty 等[75]通过分析研究证明，基于深度学习方法的缺陷定位性能优于基于传统检索模型的方法，同时可以有效缓解词不匹配问题，但也需要更长的训练时间和更多的硬件资源。

2017 年，Lam 等[76]提出了一种结合深度学习和信息检索技术进行缺陷定位的新方法——DNNLOC。该方法融合了深度神经网络(DNN)和信息检索(IR)技术 rVSM。该方法通过 rVSM 收集缺陷报告与源代码文件之间的文本相似性特征。DNN 被训练以识别缺陷报告中的术语与源代码文件中的代码令牌之间的关联。DNN 与 IR 技术的结合，使得 DNNLOC 在缺陷定位的准确性上超越了单一模型的性能。同年，Xiao 等[77]充分利用缺陷报告和程序模块中的语义信息，将卷积神经网络(CNN)和缺陷修复历史信息相结合，提出了基于增强 CNN 的 DeepLocator。但是该方法使用相同的单词嵌入技术和 CNN 来处理缺陷报告和源文件，忽略了它们之间的差异，从而限制了 DeepLocator 的性能。2019 年，Xiao 等[78]改进了文献[34]中的模型，对缺陷报告和程序模块采用两种不同的词嵌入技术(Sent2Vec 和 word2vec)，提出了利用词嵌入和增强卷积神经网络改进缺陷定位方法 DeepLoc。相较于之前的方法 DeepLocator，该方法在性能和效率上均展现出更优的表现。2022 年，Qi 等[79]提出利用相关性匹配模型来进行缺陷定位的方法 DreamLoc。该方法使用注意力机制和门控机制，通过从局部匹配和全局匹配相结合的角度考虑缺陷报告和代码，可以有效地对缺陷报告和源文件的特征进行建模，提高缺陷报告和代码文件关联匹配的准确性。实验结果与 DeepLoc 方法相比，在评估标准 Accuracy@10、MAP 和 MRR 上分别提高了 6.4%，7.4%和 7.2%。Li 等[41]提出了一种基于频谱的深度神经网络缺陷定位方法 Deep-SBFL，用于定位最有可能存在缺陷的神经元。2024 年，Song 等[80]提出一种基于深度神经网络(DNN)技术挖掘代码抽象语法树(Abstract Syntax Tree，AST)细粒度特征信息的缺陷定位方法 Deep-FGDL。该方法利用正确代码片段构建模板库，突破了缺乏标注数据的限制。常规流程下，先运用代码语义相似度分析搜取正确模板完成初步锚定，接着提出怀疑度计算公式，联动代码缺陷检测模型，对初步结果予以加权精修，以此达成细粒度的精准缺陷定位，全方位保障模型的有效性与精准度。在代码缺陷定位过程中，当无法直接依据代码语义找到适配模板时，引入缺陷模式定位法作为补充。该

方法的总体模型如图 6 所示。同年，Shen 等[81]提出了一种两阶段的融合信息检索和深度模型特征的缺陷定位方法 TosLoc。在第一阶段，该方法针对特定版本的缺陷报告及其对应的所有项目代码，运用基于信息检索的定位技术执行粗粒度搜索，以识别出与缺陷报告相关的候选代码段；进入第二阶段后，将借助深度学习模型进一步分析源代码和缺陷报告的特征，从而实现更精准的缺陷定位。该方法与 DreamLoc 相比，在花费 DreamLoc 方法 35% 的检索时间下，平均 MRR 值比 DreamLoc 方法提高了 2.5%，平均 MAP 值提高了 6.0%。

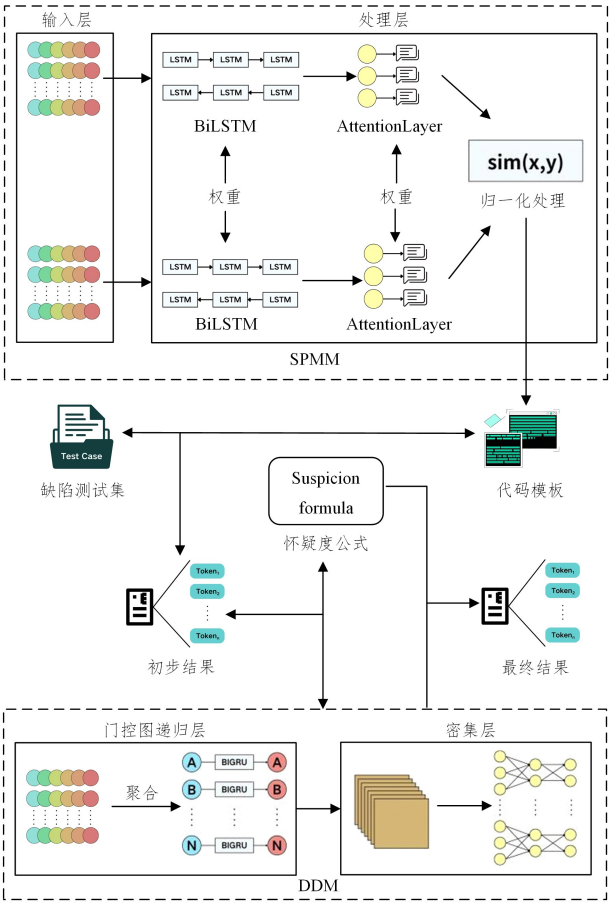

图 6 基于深度学习的缺陷定位方法的总体模型

Fig. 6 Research framework of defect location methods based on deep learning

4 多缺陷定位技术

多缺陷定位(Multiple Fault Localization，MFL)是在缺陷软件程序中识别多个缺陷位置的技术。与假定软件只包含一个缺陷的传统方法相比，该方法更加困难、耗时、昂贵，但准确率更高，更符合实际应用场景。近年来，研究人员针对 MFL 问题发表了大量的研究成果。2022 年，Li 等[82]以 MFL 研究问题为核心，系统地回顾了该领域的相关研究成果，将现有的 MFL 技术分为 3 种类型：1)基于缺陷干扰假设的多缺陷定位方法；2)基于缺陷独立假设的多缺陷定位方法；3)不基于任何假设的多缺陷定位方法。此外，他们还总结了每种分类方法的主要设计思想和相关研究成果。

随着计算机性能的不断提升，研究人员尝试将新的领域引入到多缺陷定位问题的求解中，其中机器学习领域和启发式技术领域受到广泛关注。Wang 等[83]提出一种基于遗传算法的多缺陷定位方法 GAMFal。该方法首先通过构建候选缺陷分布的染色体编码来表示潜在的缺陷位置，并基于扩展的 Ochiai 系数计算个体的适应度值；随后使用遗传算法在解空间中搜索具有最高适应度值的候选缺陷分布，在终止条件被满足后返回最优解种群；最后根据这个种群对程序实体进行排序。实证研究以 Siemens 套件中的 7 个程序和 Linux 的 3 个程序(gzip，grep 和 sed)作为评测对象，GAMFal 方法在整体定位效率方面优于传统方法，且需要更少的人工交互。除此之外，GAMFal 的执行时间也在可接受的范围之内。Zheng 等[84]提出一种基于遗传算法的 FSMFL 方法。该方法使用随机策略生成初始候选解的种群，然后通过定义的适应度函数对种群中的每个候选解进行评估，最后通过选择、交叉、变异、接受和替换操作生成新的种群，同时设置终止条件来决定是否继续进化当前种群。实验结果表明，FSMFL 在多缺陷定位的效果和精度上，比较传统的基于程序频谱的缺陷定位方法具有更好的定位效果。Cao 等[85]提出了一种使用遗传操作的蚁群算法进行多故障定位的自动方法(ACOMFL)。该方法首先通过覆盖率信息构建搜索图，然后通过设计的适应度函数以选择最佳路径并动态调整信息素更新，最后通过最优路径集中的蚂蚁路径信息计算每个语句的可疑度排名。实验结果表明，与 6 种基于频谱的故障定位方法和 1 种基于遗传算法的多故障定位方法相比，ACOMFL 在处理多故障程序时取得了更好的结果。

相较于单缺陷定位，MFL 在处理实际软件缺陷时更为复杂和全面，因为其能够在程序中定位多个缺陷，同时考虑解决不同缺陷之间的干扰情况，从而更加贴合实际应用场景。单缺陷定位技术与多缺陷定位技术的对比差异如表 4 所列。

表 4 单缺陷定位的与多缺陷定位的对比

Table 4 Comparison of single defect localization and multi-defect localization

	单缺陷定位	多缺陷定位
假设	被测试的程序或软件内仅含有一个缺陷	被测试程序或软件内可能存在多个缺陷的情况
测试用例与缺陷对应关系	每一个失败的测试用例都只与同一个缺陷有关	一个失败的测试用例可能由多个缺陷导致
技术挑战性	面临的挑战较小	更具挑战性，需要更复杂的算法和策略来处理多个缺陷的相互作用
定位效果	仅需要定位一个缺陷	需要考虑多处定位，数目未知
研究方法	研究方法相对单一	研究方法可以分为 3 类：INF-MFL，IDP-MFL，NOH-MFL
应用场景	仅适用于简单的单缺陷场景	往往更适用于复杂的场景，更加贴合实际应用
评估方法	传统的评估方法	研究人员对评估方法进行了拓展，以适应多缺陷场景

5 常用评测数据集及评价指标

5.1 常用软件缺陷数据集

表 5 列出了软件缺陷定位研究中常用的评测数据集，涵盖了不同语言和规模的程序测试。数据集大多可从 SIR(Software-artifact Infrastructure Repository)[86]或 GitHub 网站下载。

Siemens 是软件缺陷定位领域最常用的数据集之一[87-90]，最初用于 Hutchins 等[90]的一项实证研究，用来比较

各种结构测试标准。后来，它被其他研究人员适当扩展和改编，以支持他们的实验[91]。Siemens 是由西门子公司开发的，包含了 7 个 C 语言程序（tcas，print_tokens，print_tokens2，replace，schedule，schedule2 and tot_info），每个程序都有一个正确版本以及多个含缺陷的版本，后续还新增了 Java 和 C++ 等常用语言的程序定位。

表 5　ASBL 常用数据集

Table 5　Common datasets of ASBL

Program	Lines of Code	Description	Language	Fault types
Siemens：tcas	173	Altitudeseparation	C	Seeded
Siemens：schedule	412	Priority scheduler	C	Seeded
Siemens：schedule2	307	Priority scheduler	C	Seeded
Siemens：print_tokens	565	Lexical analyzer	C	Seeded
Siemens：print_tokens2	510	Lexical analyzer	C	Seeded
Siemens：replace	563	Patternrecognition	C	Seeded
Siemens：tot_info	406	Information measure	C	Seeded
Unix：cal	202	Print a calendar for a specified year or month	C	Real
Unix：checkeq	102	Report missing or unbalanced delimiters and EQ/. EN pairs	C	Real
Unix：col	308	Filter reverse line	C	Real
Unix：comm	167	Select or reject lines common to two sorted files	C	Real
Unix：crypt	134	Encrypt and decrypt a file using a user supplied password	C	Real
Unix：look	170	Find words in the system dictionary or lines in a sorted list	C	Real
Unix：sort	913	Sort and merge files	C	Real
Unix：spline	338	Interpolate smooth curves based on given data	C	Real
Unix：tr	137	Translate characters	C	Real
Unix：uniq	143	Report or remove adjacent duplicate lines	C	Real
Space	9 564	ADL Interpreter	C	Real
NanoXML	7 646	XML parser	Java	Seeded
Ant	75 333	Java applications builder	Java	Seeded
Make	20 014	Manage building of executable and other products from code	C	Seeded
XML-sec	21 613	Library for XML encryption	C	Seeded
Defects4j：Commons Math	103 900	Apache commons mathlibrary	Java	Real
Defects4j：Commons Lang	49 900	Apache commonslang library	Java	Real
Defects4j，Joda-ime	105 200	Joda-time library	Java	Real
Defects4j：JFreeChart	391 200	An open-source Java chart library	Java	Real

近年来，Defects4j 在 ASBL 领域被广泛使用[92-95]，最初由 Just 等[96]提出，它包含多个大型 Java 开源项目产生的真实缺陷。每个开源程序都具有多个历史版本，且每个版本都至少包含一个可在修复提交记录中查询的错误。在使用 Defects4J 进行缺陷定位相关实验时，开发者可利用提供的命令查看各程序的缺陷报告和历史版本等信息，同时它还提供了所有程序的编译源文件，便于研究者进行修改和重新编译。

5.2　评价指标

评价指标是衡量方法有效性的关键依据。针对不同模型的输出结果，研究者会采用不同类型的指标来进行评估。在缺陷定位领域，这些评价指标主要可以分为基于精度和基于代价两大类。

5.2.1　基于精度的评价指标

基于精度的评价指标是计算经过缺陷定位技术处理之后达到某一定位精度要求的程序版本数量，从而评估缺陷定位技术有效性。常用的基于精度的评价指标有 TOP-N[11]，MRR[13]，MAP[15]等。

1）TOP-N

TOP-N 衡量了定位缺陷位置的绝对排名，即在一个语句怀疑度降序列表 S 中，至少有一个导致程序故障的元素排名位于前 N 的个数。N 越小，TOP-N 越大，则缺陷定位技术效果越好。大多数实验采用 TOP-1，TOP-5 和 TOP-10 这 3 个指标来进行评估[97]。

2）MRR

MRR（Mean Reciprocal Rank）是衡量第一个相关文档排名的综合指标，指所有可疑语句列表中缺陷语句排名倒数的平均值。其计算式如式（7）所示。

$$MRR=\frac{1}{|Y|}\sum_{y\in Y}\frac{1}{effectiveness(y)} \tag{7}$$

其中，Q 表示缺陷版本的数量，$effectiveness(i)$ 为第 i 个缺陷版本的可疑语句列表中含有缺陷语句的排名。MRR 值越大，缺陷定位技术的性能越好。

3）MAP

MAP（Mean Average Precision）是基于集合 Y 中每个查询 y 的平均精度来衡量缺陷定位方法准确性的指标。如式（8）所示，给定与查询 y 相关的文档集 T_y，平均精度计算为每个文档最终排名的精度值的平均值，MAP 则是这组查询集 Y 的平均精度的平均值。

$$MAP=\frac{1}{Y}\sum_{y\in Y}\frac{1}{|T_y|}\sum_{t\in T_y}precision(rank(t)) \tag{8}$$

5.2.2　基于代价的评价指标

基于代价的评价指标是计算定位程序中真实缺陷位置所需的时间成本，来评估缺陷定位技术的有效性。目前，常用的基于代价的评价指标有 T-Score[98]，EXAM[99]，CNSE[42]等。

1）T-Score

T-Score 是基于程序依赖图的评价指标。通过构建一个程序依赖图，对于语句 n，首先定义 k 依赖集 DS_k，表示存在长度不大于 k 的有向路径连接 n 和它们的节点集合；然后，设 $DS_*(R)$ 为包含缺陷语句的最小依赖域。则该评价指标的计算式为：

$$T\text{-}Score = 1 - \frac{|DS_*(R)|}{|PDG|} \tag{9}$$

2)EXAM

EXAM 指标[100]是软件缺陷定位中最常用的评价指标之一,是一种基于语句怀疑度排序的评价指标,它指在检测到缺陷语句前需要检查的程序语句占所有程序语句的百分比。EXAM 值越小,表示缺陷定位性能越好。EXAM 如式(10)所示。

$$EXAM = \frac{n_{ds}}{n_{es}} \times 100\% \tag{10}$$

其中,n_{ds} 表示当前执行错误代码数,n_{es} 表示当前总执行代码数。研究者为使 EXAM 评价指标适应于更多软件缺陷方法,将 EXAM 进行了拓展。例如,Wang 等[83]提出针对单缺陷的 EXAM 指标并不完全适用于多缺陷定位,因此他们拓展了 EXAM 的定义,提出了 EXAMF,EXAML 指标。EXAMF 指返回检测到第一个缺陷前需要检查的程序语句占总程序语句的百分比,而 EXAML 表示返回检测到最后一个缺陷前需要检查的程序语句占总程序语句的百分比,保证了所有的缺陷程序实体的可疑度值排序较为靠前。

3)CNSE

累积检查语句和(Cumulative Number of Statements Examined,CNSE)指对于给定的被测程序,设 n 为缺陷版本数量,A 和 B 为两种不同的缺陷定位技术,则每个程序版本 j,$A(j)$和 $B(j)$分别表示使用 A 和 B 技术所有缺陷所需要检查的代码行数之和。若总代码行数之和 $\sum_{j=1}^{p} A(j)$ 小于 $\sum_{j=1}^{p} B(j)$,则说明 A 方法比 B 方法更高效。CNSE 值越小,表示缺陷定位技术越好。

6 挑战与展望

通过上述的分析研究可以看出,随着软件规模的不断增大,为提高软件维护的效率和质量,自动化软件缺陷定位技术已成为目前的研究热点。本文通过对文献的深入调研,基于动静态的分类方法,较为全面地总结了自动化软件缺陷定位技术、评测数据集以及评测指标,并对单缺陷与多缺陷技术进行了比较。虽然越来越多的技术被不断提出,但从上文中可以看出,现有的一些缺陷定位技术主要停留在研究实验阶段,缺少实际实践环境。同时,这些技术方法主要使用 C/C++和 Java 等主流编程语言,涉及其他语言的研究较少。因此,尽管现在存在多种不同的定位技术,但从缺陷定位技术所必须具备的属性(如定位精度、效率等)角度出发,目前软件缺陷定位技术仍然面临着不少的挑战并需要进一步地得到解决和优化。

1)增强技术实用性

目前,很多缺陷定位方法仅实现在研究实验阶段,并且大部分实验使用的均为开源程序,其中还包括人工植入缺陷,很少有方法用真实工业环境进行实验并应用于实际实践。Lee 等[101]指出,论文提出的研究方法与实际开发中应用的技术存在显著差异。尽管不同的缺陷定位方法在性能上表现出相近的结果,但并非所有技术都能完全适应研究对象。未来,软件缺陷定位技术还需要结合实际的数据环境,进一步提高在工程实践中的准确性。

2)提高语言拓展性

以 TIOBE 编程语言排行榜每年的编程语言排行指标为参考标准,2009 年至 2020 年这十多年间,C/C++和 Java 这 3 种编程语言几乎稳居榜单前三,这也是导致现有的大多数缺陷定位研究主要集中在 C/C++和 Java 等主流编程语言,而对其他语言的程序的研究较少的主要原因,因此研究人员针对某种特定语言开发出有效缺陷定位方法后,这些方法可能只局限于该语言而无法实现拓展。

自 2016 年开始,Python 等解释型语言的市场逐渐被打通,在编程语言排行榜上呈稳步上升趋势。2021 年 10 月,编程语言达到排行榜第一的位置,自此 Python 逐渐成为了更加流行的编程语言。考虑到 Python 目前的流行程度,了解自动化软件缺陷定位技术如何在 Python 项目上执行变得越来越重要。然而,这仍然是一个未充分研究的主题。2022 年,Widyasari 等[102]分析了典型的 SBFL 技术在 Python 项目中的有效性,并将在 Python 上观察到的性能与之前在 Java 上报告的性能进行比较。2024 年,Rezaalipour 等[103]探究了经典缺陷定位方法在 Python 中的有效性、效率和其他特征,并比较了其与 Java 等其他语言的效果是否相同。他们以 Zou 等[93]最近对 Java 进行的大规模实证研究为基础,对 4 种类型的 7 个常见的缺陷定位技术进行了测试,共涉及 135 个来自 13 个开源 Python 项目的故障。结果表明,Python 和 Java 之间存在许多相似之处,但也有一些不同点需要进一步研究。因此,为提高缺陷定位技术的适用性和普遍性,未来研究需考虑更多的编程语言,探索不同语言解决缺陷定位问题的实践效果。

3)考虑时间成本

使用不同的工具进行软件缺陷定位,耗时情况会有很大的差异,这取决于定位技术的类型、软件项目的大小、代码复杂度、缺陷的隐蔽性以及所使用的工具和算法的效率。

静态分析工具通过分析代码的静态特性和缺陷报告来识别潜在缺陷,根据具体的项目规模和复杂性来完成定位,通常用时在几分钟到几小时;动态分析工具主要通过测试用例的执行结果来定位缺陷,因此耗时与测试用例的数量和复杂度有关。其中,在基于变异的方法中,变异测试是引入变异到代码中,通过运行测试用例来查看这些变异是否被检测到,从而评估测试用例的质量,研究人员因其需要多次编译和执行变异的测试用例,进行成百上千次测试和计算,而认为过程可能非常耗时。同时还有基于机器学习的方法,机器学习需要较长的时间来训练模型,尤其是当碰到大型数据集时,耗时较长,如深度学习。但是当机器学习模型训练完成,用这种技术进行缺陷定位的速度是相对较快的。上述内容说明,尽管许多缺陷定位方法能够较好地识别出缺陷的位置,但它们可能会因具体的工具实现、硬件性能、并行处理能力等因素而伴随着较高的时间成本,这在快速迭代和持续集成的现代软件开发流程中构成了显著挑战。在敏捷开发和持续交付成为行业标准的背景下,开发团队需要频繁地进行代码更新和测试,以确保软件产品的质量和稳定性。因此,为了更好地适应当前高效、快速的开发模式,未来的研究需要致力于优化现有的缺陷定位技术,减小时间成本,同时保持甚至提高其准确性。

4)应用深度学习技术

深度学习的特点是使用多个层次的神经网络模型来处理和分析数据,通过自动学习数据中的特征来实现任务,如分类、聚类、语音识别、图像识别等。近年来,深度学习技术被广

泛应用于软件缺陷定位领域[104-108],通过利用深度学习模型提取程序结构信息,并整合软件缺陷数据的各种特征,能够更准确地刻画和解决软件缺陷定位问题。未来的研究需要更加深入地探索如何进一步优化深度学习模型,使其能够更加高效地处理大规模软件系统中的多缺陷情况,同时提升算法的泛化能力以适应不同类型的编程语言和软件项目。

结束语 软件缺陷定位作为软件调试过程中的关键环节,近年来受到了广泛关注。本文通过深入调研,系统地回顾了该领域的研究背景、已有研究成果、评测数据集以及评价指标,并进行了全面的总结与分析。目前,此研究领域正处于快速发展阶段,新方法和新技术不断涌现。本文将现有的缺陷定位技术划分为静态和动态两大类,并对每类中的几种典型方法进行了详细介绍。此外,还探讨了软件缺陷定位未来研究的趋势,提出了几个潜在的研究议题,旨在为后续研究提供有价值的参考和启发。

参 考 文 献

[1] LUCIA L,THUNG F,LO D,et al. Are faults localizable? [C]// Proceeding of the 2012 9th IEEE Working Conference on Mining Software Repositories(MSR). 2012:74-77.

[2] ZHANG W,LI Z Q,DU Y H,et al. Fine-grained software bug location approach at method level[J]. Journal of Software,2019,30(2):195-210.

[3] YU K,LIN M X. Advances in automatic fault localization techniques[J]. Chinese Journal of Computers,2011,34(8):1411-1422.

[4] CHEN X,JU X L,WEN W Z,et al. Review of dynamic fault localization approaches based on program spectrum[J]. Journal of Software,2015,26(2):390-412.

[5] 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.

[6] ZHANG Y,LIU J K,XIA X,et al. Research progress on software bug localization technology based on information retrieval [J]. Ruan Jian Xue Bao:Journal of Software,2020,31(8):2432-2452.

[7] LI Z L,CHEN X,JIANG Z W,et al. Survey on information retrieval-based software bug localization methods[J]. Ruan Jian Xue Bao:Journal of Software,2021,32(2):247-276.

[8] NI Z,LI B,SUN X B,et al. Research and Progress on Bug Report-oriented Bug Localization Techniques[J]. Computer Science,2022,49(11):8-23.

[9] IEEE Std 1044-2009,IEEE Standard Classification for Software Anomalies[S]. IEEE Standard Indus,2010:1-23.

[10] WEISER M D. Program slices:formal,psychological,and practical investigations of an automatic program abstraction method [M]. University of Michigan,1979.

[11] WANG S,LO D. Amalgam+:Composing rich information sources for accurate bug localization[J]. Journal of Software:Evolution and Process,2016,28(10):921-942.

[12] SHI Z,KEUNG J,BENNIN K E,et al. Comparing learning to rank techniques in hybrid bug localization[J]. Applied Soft Computing,2018,62:636-648.

[13] YUE L,CUI Z Q,CHEN X,et al. Statement level software bug localization based on historical bug information retrieval[J]. Journal of Software,2024,35(10):4642-4661.

[14] YANG Y,WANG Z,CHEN Z,et al. COPS:An improved information retrieval-based bug localization technique using context-aware program simplification[J]. Journal of Systems and Software,2024,207:111868.

[15] RAHMAN M M,ROY C K. Improving ir-based bug localization with context-aware query reformulation[C]// Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2018:621-632.

[16] KIM M,LEE E. A novel approach to automatic query reformulation for ir-based bug localization[C]// Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing. 2019:1752-1759.

[17] GHARIBI R,RASEKH A H,SADREDDINI M H,et al. Leveraging textual properties of bug reports to localize relevant source files[J]. Information Processing & Management,2018,54(6):1058-1076.

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

[19] WEN M,WU R,CHEUNG S C. Locus:Locating bugs from software changes[C]// Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. 2016:262-273.

[20] ALMHANA R,MKAOUER W,KESSENTINI M,et al. Recommending relevant classes for bug reports using multi-objective search[C]// Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. 2016:286-295.

[21] 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,2015,42(4):379-402.

[22] HOANG T,OENTARYO R J,LE T D B,et al. Network-clustered multi-modal bug localization[J]. IEEE Transactions on Software Engineering,2018,45(10):1002-1023.

[23] RAHMAN S,GANGULY K K,SAKIB K. An improved bug localization using structured information retrieval and version history[C]// 18th International Conference on Computer and Information Technology(ICCIT 2015). IEEE,2015:190-195.

[24] KOYUNCU A,BISSYANDÉ T F,KIM D,et al. D&c:A divide-and-conquer approach to ir-based bug localization[J]. arXiv:1902.02703,2019.

[25] ZHANG W,LI Z,WANG Q,et al. FineLocator:A novel approach to method-level fine-grained bug localization by query expansion[J]. Information and Software Technology,2019,110:121-135.

[26] DILSHENER T,WERMELINGER M,YU Y. Locating bugs without looking back[C]// Proceedings of the 13th International Conference on Mining Software Repositories. 2016:286-290.

[27] RATH M,LO D,MÄDER P. Analyzing requirements and traceability information to improve bug localization[C]// Proceedings of the 15th International Conference on Mining Software Repositories. 2018:442-453.

[28] SISMAN B,AKBAR S A,KAK A C. Exploiting spatial code

proximity and order for improved source code retrieval for bug localization[J]. Journal of Software: Evolution and Process, 2017, 29(1): e1805.

[29] LE T D B, OENTARYO R J, LO D. Information retrieval and spectrum based bug localization: Better together[C]// Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. 2015: 579-590.

[30] DAO T, ZHANG L, MENG N. How does execution information help with information-retrieval based bug localization? [C]// IEEE/ACM 25th International Conference on Program Comprehension(ICPC 2017). IEEE, 2017: 241-250.

[31] KUMAR D, SHARMA R. Bug Localization using LDACG Approach[J]. International Journal of Engineering Research and Technology, 2015(4).

[32] DE SOUZA H A, CHAIM M L, KON F. Spectrum-based software fault localization: A survey of techniques, advances, and challenges[J]. arXiv: 1607. 04347, 2016.

[33] SARHAN Q I, BESZÉDES Á. A survey of challenges in spectrum-based software fault localization[J]. IEEE Access, 2022, 10: 10618-10639.

[34] JONES J A, HARROLD M J, STASKO J. Visualization of test information to assist fault localization[C]// Proceedings of the 24th International Conference on Software Engineering. 2002: 467-477.

[35] JONES J A, HARROLD M J. Empirical evaluation of the tarantula automatic fault-localization technique[C]// Proceedings of the 20th IEEE/ACM international Conference on Automated Software Engineering. 2005: 273-282.

[36] ABREU R, ZOETEWEIJ P, VAN GEMUNDA J C. An evaluation of similarity coefficients for software fault localization[C]// 2006 12th Pacific Rim International Symposium on Dependable Computing(PRDC'06). IEEE, 2006: 39-46.

[37] ABREU R, ZOETEWEIJ P, VAN GEMUND A J C. On the accuracy of spectrum-based fault localization[C]// Testing: Academic and Industrial Conference Practice and Research Techniques-MUTATION(TAICPART-MUTATION 2007). IEEE, 2007: 89-98.

[38] RENIERES M, REISS S P. Fault localization with nearest neighbor queries[C]// 18th IEEE International Conference on Automated Software Engineering, IEEE, 2003: 30-39.

[39] NAISH L, RAMAMOHANARAO K. Multiple bug spectral fault localization using genetic programming[C]// 2015 24th Australasian Software Engineering Conference. IEEE, 2015: 11-17.

[40] WONG W E, QI Y. BP neural network-based effective fault localization[J]. International Journal of Software Engineering and Knowledge Engineering, 2009, 19(4): 573-597.

[41] LI Z, CUI Z Q, CHEN X, et al. Deep-SBFL: Spectrum-based fault localization approach for deep neural networks[J]. Journal of Software, 2023, 34(5): 2231-2250.

[42] JIA Y, HARMAN M. An analysis and survey of the development of mutation testing[J]. IEEE Transactions on Software Engineering, 2010, 37(5): 649-678.

[43] MOON S, KIM Y, KIM M, et al. Ask the mutants: Mutating faulty programs for fault localization[C]// 2014 IEEE Seventh International Conference on Software Testing, Verification and Validation. IEEE, 2014: 153-162.

[44] PAPADAKIS M, LE TRAON Y. Metallaxis-FL: mutation-based fault localization[J]. Software Testing, Verification and Reliability, 2015, 25(5/6/7): 605-628.

[45] HONG S, LEE B, KWAK T, et al. Mutation-based fault localization for real-world multilingual programs(T)[C]// 30th IEEE/ACM International Conference on Automated Software Engineering(ASE 2015). IEEE, 2015: 464-475.

[46] LIU Y, LI Z, WANG L, et al. Statement-oriented mutant reduction strategy for mutation based fault localization[C]// IEEE International Conference on Software Quality, Reliability and Security(QRS 2017). IEEE, 2017: 126-137.

[47] LIU Y, LI Z, ZHAO R, et al. An optimal mutation execution strategy for cost reduction of mutation-based fault localization [J]. Information Sciences, 2018, 422: 572-596.

[48] DE OLIVEIRA AA L, CAMILO-JUNIOR C G, DE ANDRADE FREITAS E N, et al. Ftmes: A failed-test-oriented mutant execution strategy for mutation-based fault localization[C]// IEEE 29th International Symposium on Software Reliability Engineering(ISSRE 2018). IEEE, 2018: 155-165.

[49] KIM J, AN G, FELDT R, et al. Ahead of time mutation based fault localisation using statistical inference[C]// IEEE 32nd International Symposium on Software Reliability Engineering(ISSRE 2021). IEEE, 2021: 253-263.

[50] LI X, JUZA R. Accelerating Mutation-Based Fault Localization via Optimized Mutant Execution[C]// 13th International Conference on Software Technology and Engineering (ICSTE 2023). IEEE, 2023: 82-86.

[51] JEON J, HONG S. Improving mutation-based fault localization with plausible-code generating mutation operators[C]// 36th IEEE/ACM International Conference on Automated Software Engineering(ASE 2021). IEEE, 2021: 1205-1207.

[52] GHANBARI A, THOMAS D G, ARSHAD M A, et al. Mutation-based Fault Localization of Deep Neural Networks[C]// 38th IEEE/ACM International Conference on Automated Software Engineering(ASE 2023). IEEE, 2023: 1301-1313.

[53] ZHANG X, GUPTA N, GUPTA R. Locating faulty code by multiple points slicing[J]. Software: Practice and Experience, 2007, 37(9): 935-961.

[54] ZHANG X, GUPTA R, ZHANG Y. Precise dynamic slicing algorithms[C]// 25th International Conference on Software Engineering, 2003. Proceedings. IEEE, 2003: 319-329.

[55] ZHANG X, GUPTA N, GUPTA R. Locating faults through automated predicate switching[C]// Proceedings of the 28th International Conference on Software Engineering. 2006: 272-281.

[56] SOREMEKUN E, KIRSCHNER L, BÖHME M, et al. Locating faults with program slicing: an empirical analysis[J]. Empirical Software Engineering, 2021, 26: 1-45.

[57] SOHA P A. On The Efficiency of Combination of Program Slicing and Spectrum-Based Fault Localization[C]// IEEE Conference on Software Testing, Verification and Validation (ICST 2023). IEEE, 2023: 499-501.

[58] WONG W E, GAO R, LI Y, et al. A survey on software fault localization [J]. IEEE Transactions on Software Engineering, 2016, 42(8): 707-740.

[59] ZHANG Z, JIANG B, CHAN W K, et al. Fault localization

through evaluation sequences[J]. Journal of Systems and Software,2010,83(2):174-187.

[60] YU Y,JONES J A,HARROLD M J. An empirical study of the effects of test-suite reduction on fault localization[C]// Proceedings of the 30th International Conference on Software Engineering. 2008:201-210.

[61] HARROLD M J,ROTHERMEL G,SAYRE K,et al. An empirical investigation of the relationship between spectra differences and regression faults[J]. Software Testing,Verification and Reliability,2000,10(3):171-194.

[62] NAISH L,LEE H J,RAMAMOHANARAO K. A model for spectra-based software diagnosis[J]. ACM Transactions on Software Engineering and Methodology(TOSEM),2011,20(3):1-32.

[63] XIE X,CHEN T Y,KUOF C,et al. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization[J]. ACM Transactions on Software Engineering and Methodology(TOSEM),2013,22(4):1-40.

[64] WONGW E,DEBROY V,GAO R,et al. The DStar method for effective software fault localization[J]. IEEE Transactions on Reliability,2013,63(1):290-308.

[65] LOU Y,ZHU Q,DONG J,et al. Boosting coverage-based fault localization via graph-based representation learning[C]// Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2021:664-676.

[66] REITER R. A theory of diagnosis from first principles[J]. Artificial intelligence,1987,32(1):57-95.

[67] DEMILLO R A,PAN H,SPAFFORDE H. Failure and fault analysis for software debugging[C]// Proceedings Twenty-First Annual International Computer Software and Applications Conference(COMPSAC'97). IEEE,1997:515-521.

[68] CLEVE H,ZELLER A. Locating causes of program failures[C]// Proceedings of the 27th International Conference on Software Engineering. 2005:342-351.

[69] JU X,QIAN J,CHEN Z,et al. Mulr4FL:effective fault localization of evolution software based on multivariate logistic regression model[J]. IEEE Access,2020,8:207858-207870.

[70] REN S B,CHEN J,TAN W Z,et al. Probabilistic model checking method for software fault location[J]. Application Research of Computers,2021,38(11):3387-3392,3397.

[71] DENMAT T,DUCASSÉ M,RIDOU O. Data mining and cross-checking of execution traces:a re-interpretation of jones,harrold and stasko test information[C]// Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. 2005:396-399.

[72] WONG W E,DEBROY V,GOLDEN R,et al. Effective software fault localization using an RBF neural network[J]. IEEE Transactions on Reliability,2011,61(1):149-169.

[73] CELLIER P,DUCASSÉ M,FERRÉ S,et al. Multiple Fault Localization with Data Mining[C]// SEKE. 2011:238-243.

[74] ZHANG S,ZHANG C. Software bug localization with markov logic[C]// Companion Proceedings of the 36th International Conference on Software Engineering. 2014:424-427.

[75] POLISETTY S,MIRANSKYY A,BAŞAR A. On usefulness of the deep-learning-based bug localization models to practitioners[C]// Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering. 2019:16-25.

[76] LAM A N,NGUYEN A T,NGUYEN H A,et al. Bug localization with combination of deep learning and information retrieval[C]// IEEE/ACM 25th International Conference on Program Comprehension(ICPC 2017). IEEE,2017:218-229.

[77] XIAO Y,KEUNG J,MI Q,et al. Improving bug localization with an enhanced convolutional neural network[C]// 24th Asia-Pacific Software Engineering Conference(APSEC 2017). IEEE,2017:338-347.

[78] XIAO Y,KEUNG J,BENNIN K E,et al. Improving bug localization with word embedding and enhanced convolutional neural networks[J]. Information and Software Technology,2019,105:17-29.

[79] QI B,SUN H,YUAN W,et al. DreamLoc:A deep relevance matching-based framework for bug localization[J]. IEEE Transactions on Reliability,2021,71(1):235-249.

[80] SONG L H,HAN Y. Deep_FGDL:A Fine-grained Defect Localization Method Based on Deep Neural Network[J]. Journal of North China University of Technology,2024,36(1):1-11.

[81] SHEN Z W,NIU F F,LI C Y,et al. Software bug location method combining information retrieval and deep model features[J]. Journal of Software,2024,35(7):3245-3264.

[82] LI Z,WU Y H,WANG H F,et al. Review of software multiple fault localization approaches[J]. Chinese Journal of Computers,2022,45(2):256-288.

[83] WANG Z,FAN X Y,ZOU Y G,et al. Genetic Algorithm Based Multiple Faults Localization Technique[J]. Journal of Software,2016,27(4):879-900.

[84] ZHENG Y,WANG Z,FAN X,et al. Localizing multiple software faults based on evolution algorithm[J]. Journal of Systems and Software,2018,139:107-123.

[85] CAO H,WANG F,DENG M,et al. Multiple fault localization based on ant colony algorithm via genetic operation[J]. Journal of King Saud University-Computer and Information Sciences,2023,35(8):101668.

[86] DO H,ELBAUM S,ROTHERMEL G. Supporting controlled experimentation with testing techniques:An infrastructure and its potential impact[J]. Empirical Software Engineering,2005,10:405-435.

[87] YU K,LIN M,GAO Q,et al. Locating faults using multiple spectra-specific models[C]// Proceedings of the 2011 ACM Symposium on Applied Computing. 2011:1404-1410.

[88] JEFFREY D,GUPTA N,GUPTA R. Fault localization using value replacement[C]// Proceedings of the 2008 International Symposium on Software Testing and Analysis. 2008:167-178.

[89] NESSA S,ABEDIN M,WONGW E,et al. Software fault localization using n-gram analysis[C]// Wireless Algorithms,Systems,and Applications:Third International Conference(WASA 2008). Dallas,TX,USA. Springer Berlin Heidelberg,2008:548-559.

[90] HUTCHINS M,FOSTER H,GORADIA T,et al. Experiments on the effectiveness of dataflow-and control-flow-based test adequacy criteria[C]// Proceedings of 16th International Conference on Software Engineering. IEEE,1994:191-200.

[91] HARDER M, MELLEN J, ERNST M D. Improving test suites via operational abstraction[C]// 25th International Conference on Software Engineering. IEEE, 2003: 60-71.

[92] WEN M, CHEN J, TIAN Y, et al. Historical spectrum based fault localization[J]. IEEE Transactions on Software Engineering, 2019, 47(11): 2348-2368.

[93] ZOU D, LIANG J, XIONG Y, et al. An empirical study of fault localization families and their combinations[J]. IEEE Transactions on Software Engineering, 2019, 47(2): 332-347.

[94] MENG X, WANG X, ZHANG H, et al. Improving fault localization and program repair with deep semantic features and transferred knowledge[C]// Proceedings of the 44th International Conference on Software Engineering. 2022: 1169-1180.

[95] ZENG M, WU Y, YE Z, et al. Fault localization via efficient probabilistic modeling of program semantics[C]// Proceedings of the 44th International Conference on Software Engineering. 2022: 958-969.

[96] JUST R, JALALI D, ERNST M D. Defects4J: A database of existing faults to enable controlled testing studies for Java programs[C]// Proceedings of the 2014 International Symposium on Software Testing and Analysis. 2014: 437-440.

[97] KOCHHAR P S, XIA X, LO D, et al. Practitioners' expectations on automated fault localization[C]// Proceedings of the 25th International Symposium on Software Testing and Analysis. 2016: 165-176.

[98] LIU C, FEI L, YAN X, et al. Statistical debugging: A hypothesis testing-based approach[J]. IEEE Transactions on Software Engineering, 2006, 32(10): 831-848.

[99] WONG E, WEI T, QI Y, et al. A crosstab-based statistical method for effective fault localization[C]// 2008 1st International Conference on Software Testing, Verification, and Validation. IEEE, 2008: 42-51.

[100] PENG Z, XIAO X, HU G, et al. ABFL: an autoencoder based practical approach for software fault localization[J]. Information Sciences, 2020, 510: 108-121.

[101] LEE J, KIM D, BISSYANDÉ T F, et al. Bench4bl: reproducibility study on the performance of ir-based bug localization[C]// Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. 2018: 61-72.

[102] WIDYASARI R, PRANA G A A, HARYONO S A, et al. Real world projects, real faults: evaluating spectrum based fault localization techniques on Python projects[J]. Empirical Software Engineering, 2022, 27(6): 147.

[103] REZAALIPOUR M, FURIA C A. An empirical study of fault localization in Python programs[J]. Empirical Software Engineering, 2024, 29(4): 92.

[104] JIANG J, WANG Y, CHEN J, et al. Variable-Based Fault Localization via Enhanced Decision Tree[J]. ACM Transactions on Software Engineering and Methodology, 2023, 33(2): 1-32.

[105] YOUSOFVAND L, SOLEIMANI S, RAFE V. Automatic bug localization using a combination of deep learning and model transformation through node classification[J]. Software Quality Journal, 2023, 31(4): 1045-1063.

[106] HAN J, HUANG C, SUN S, et al. bjXnet: an improved bug localization model based on code property graph and attention mechanism[J]. Automated Software Engineering, 2023, 30(1): 12.

[107] QIAN J, JU X, CHEN X. GNet4FL: effective fault localization via graph convolutional neural network[J]. Automated Software Engineering, 2023, 30(2): 16.

[108] CAO J, LI M, CHEN X, et al. Deepfd: Automated fault diagnosis and localization for deep learning programs[C]// Proceedings of the 44th International Conference on Software Engineering. 2022: 573-585.

FANG Jinqiu, born in 2004, undergraduate. Her main research interest is software fault localization.

ZHAO Haiyong, born in 1981, Ph.D, associate professor, master's supervisor. His main research interest is software fault localization.