软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(2):390−412 [doi: 10.13328/j.cnki.jos.004708]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于程序频谱的动态缺陷定位方法研究*

陈 翔[1,2], 鞠小林[1], 文万志[1], 顾 庆[2]

[1](南通大学 计算机科学与技术学院,江苏 南通 226019)

[2](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

通讯作者: 陈翔, E-mail: xchencs@ntu.edu.cn

摘 要: 基于程序频谱的动态缺陷定位是软件自动化调试研究中的一个热点问题,通过搜集测试用例的程序频谱和执行结果,基于特定模型以定位缺陷语句在被测程序内的可能位置.对近些年来国内外学者在该研究领域取得的成果进行系统总结:首先,给出预备知识和基本假设;随后,提出缺陷定位研究框架并识别出框架内一系列可影响缺陷定位效果的内在影响因素,包括程序频谱构造方式、测试套件构成和维护、内在缺陷数量、测试用例预言设置、用户反馈和缺陷修复开销等;接着,对实证研究中采用的评测指标和评测程序进行总结和分析;然后,对缺陷定位方法在一些特定测试领域中的应用进行总结;最后,对该领域未来值得关注的研究方向进行了展望.

关键词: 软件调试;缺陷定位;程序频谱;测试用例;测试用例预言

中图法分类号: TP311

中文引用格式: 陈翔,鞠小林,文万志,顾庆.基于程序频谱的动态缺陷定位方法研究.软件学报,2015,26(2):390−412. http://www.jos.org.cn/1000-9825/4708.htm

英文引用格式: Chen X, Ju XL, Wen WZ, Gu Q. Review of dynamic fault localization approaches based on program spectrum. Ruan Jian Xue Bao/Journal of Software, 2015,26(2):390−412 (in Chinese). http://www.jos.org.cn/1000-9825/4708.htm

Review of Dynamic Fault Localization Approaches Based on Program Spectrum

CHEN Xiang[1,2], JU Xiao-Lin[1], WEN Wan-Zhi[1], GU Qing[2]

[1](School of Computer Science and Technology, Nantong University, Nantong 226019, China)

[2](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

Abstract: Program spectrum based dynamic fault localization is an active research topic in the domain of software automatic debugging. It aims to localize pontential faults in a faulty program based on a specific model which is constructed on execution behaviors and results of test cases. This survey offers a systematic overview of existing research achievements of the domestic and foreign researchers in recent years. First, some preliminary knowledge and basic assumptions are presented. Next, a research framework is proposed and important influencing factors which can affect the effectiveness of fault localization are identified. These factors include program spectrum construction, test suite maintenance and composition, number of faults, test case oracle, user feedback, and fault removal cost. In addition, the evaluation metrics and subject objects used in previous empirical studies are analyzed. Furthermore, classical applications of fault localization in some specific application domains are summarized. Finally a perspective of the future work in this research area is discussed.

Key words: software debugging; fault localization; program spectrum; test case; test case oracle

软件调试是软件开发和维护过程中的一项重要任务,其核心活动包括缺陷的发现、定位、理解和移除.软

* 基金项目: 国家自然科学基金(61202006, 61373012); 江苏省高校自然科学研究项目(12KJB520014); 江苏省研究生培养创新工程(CXZZ120935); 南通市应用研究计划(BK2014055, BK2014056); 南京大学计算机软件新技术国家重点实验室开放课题(KFKT2012B29); 大学生创新训练计划(201310304087X, 2014075)

收稿时间: 2014-02-12; 修改时间: 2014-04-15; 定稿时间: 2014-08-24; jos 在线出版时间: 2014-12-12

CNKI 网络优先出版: 2014-12-12 13:53, http://www.cnki.net/kcms/detail/11.2560.TP.20141212.1353.002.html

件企业内部的开发实践表明,这项任务难度较高且费时费力.其中,缺陷定位(fault localization)是软件调试过程中成本较为高昂的一项重要活动.当开发人员在测试程序中发现存在测试用例执行失败时,传统缺陷定位方法是从中选出某一失败测试用例,然后依次在不同可疑语句处设置程序断点,执行该测试用例,观察程序断点处的变量取值,直至找到真正缺陷语句为止.但这类方法借助手工方式,存在定位缺陷语句代价高、未能充分利用测试用例的执行行为和执行结果等不足.近些年来,国内外研究人员对高效缺陷定位方法展开了深入的研究并取得了一定的研究进展.根据是否需要执行测试用例,可将缺陷定位方法划分为静态缺陷定位方法和动态缺陷定位方法.其中,静态缺陷定位方法不需要执行测试用例,主要采用代码审查方法,重点对被测程序的内在结构(例如控制依赖关系、数据依赖关系或类型约束等)进行分析,以确定缺陷语句在被测程序内的可能位置;而动态缺陷定位方法则需要分析被测程序的内在结构,搜集测试用例的执行行为和结果,基于特定模型以确定缺陷语句在被测程序内的可能位置,最终生成缺陷定位报告以辅助开发人员进行软件调试.其中,在动态缺陷定位方法中,基于程序频谱的动态缺陷定位(spectrum based dynamic fault localization,简称 SFL)是目前一种主流研究思路,也是本综述重点关注的研究问题.

虞凯等人在 2011 年对自动缺陷定位进行了系统的总结[1],我们以该综述的分析为基础,大量吸收了近年来国内外学者取得的最新研究成果,重点对自动缺陷定位中的一个重要子问题(即 SFL 问题)进行了较为全面的分析和梳理.本文的主要贡献可总结如下:

(1) 完整并系统地分析了 SFL 近些年来取得的研究成果,通过搜集并分析 100 余篇在国内外权威期刊和会议上发表的相关文献,提出 SFL 研究框架并识别出框架内的影响因素,这些影响因素包括程序频谱的构造方式、测试套件(test suite)的构成和维护、内在缺陷的数量、测试用例预言的设置、用户反馈和缺陷修复开销等.针对每个影响因素,我们系统分析了研究动机、核心问题和已有的解决方案等.

(2) 总结了 SFL 实证研究中常用的评测指标和评测程序,并对评测程序的使用趋势进行了分析.

(3) 总结了 SFL 方法在一些特定测试领域中的应用,这些测试领域包括并发程序测试、Web 应用测试、图形用户界面测试、嵌入式系统测试和回归测试等.

(4) 对比文献[1]中引用的参考文献,我们大量补充了近年来国内外的研究成果.经过统计,共新增相关研究文献近 80 篇.

本文第 1 节给出预备知识和基本假设.第 2 节提出 SFL 方法研究框架.第 3 节依次对框架内的一系列影响因素进行深入分析.第 4 节对已有 SFL 实证研究内容进行汇总,重点关注有效性评测指标和实证研究中使用的评测程序.第 5 节对 SFL 方法在一些特定测试领域中的应用进行总结.最后总结全文,并对未来值得关注的研究方向进行初步的探讨.

1 预备知识和基本假设

1.1 预备知识

SFL 在定位可疑语句时,需要执行大量测试用例.通过依次搜集测试用例的程序频谱和执行结果,基于特定模型给出缺陷语句在被测程序内的可能位置.假设 $T=\{t_1,t_2,\ldots,t_n\}$为缺陷程序 P 的配套测试套件,其中,第 j 个测试用例 t_j 可用一个有序对$\langle i_j,o_j\rangle$表示,其中,i_j 表示测试用例 t_j 的实际输入,o_j 表示 t_j 的预期输出.假设 $\hat{o}_j$ 为测试用例的实际输出,若测试用例的实际输出 $\hat{o}_j$ 与预期输出 o_j 保持一致,则称该测试用例为成功测试用例(passed test case);否则,称该测试用例为失败测试用例(failed test case).因此,根据测试用例在程序 P 上的执行结果,可以将测试套件 T 划分为集合 T_p 和 T_f,其中,T_p 包含所有成功测试用例,T_f 包含所有失败测试用例.

1.2 基本假设

目前,大部分 SFL 研究均建立在一定的假设基础上.Steimann 等人将已有假设总结为如下 4 个基本假设[2]:

- 假设 1:缺陷具有偶然正确(coincidental correctness)属性,即,缺陷语句既可能被成功测试用例覆盖到,也可能被失败测试用例覆盖到.

- 假设 2:每个失败测试用例至少会覆盖到 1 个缺陷语句.
- 假设 3:因为无法事先预知到被测程序内缺陷分布的先验概率,所以在软件调试时无法利用缺陷分布信息来指导开发人员进行缺陷定位.
- 假设 4:完美缺陷检测假设,即,在评估 SFL 方法有效性时,假设开发人员在代码审查时可以准确判断出可疑语句是否为缺陷语句,并可以随后有效移除该缺陷.

2 研究框架

开发人员在早期尝试构造出两个输入相似度高的测试用例,其中一个为成功测试用例,另一个为失败测试用例.其假设是:若测试用例的输入相似度高,则执行行为也应高度相似.因此,可以通过比较两者执行行为间的差异来有效辅助缺陷定位.

随后,Renieris 等人[3]进一步假设配套测试套件 T 内包含单个失败测试用例 t_f 和多个成功测试用例 T_p,假设用 $ps(t)$ 表示测试用例 t 覆盖的基本语句块集.他们首先给出两种基准方法:集合并(set union)法和集合交(set intersection)法,其中,集合并法返回的可疑语句块集为 $ps(t_f)-\bigcup_{t_p\in T_p} ps(t_p)$,其含义是返回仅被失败测试用例覆盖的语句块集;集合交法返回的可疑语句块集为 $(\bigcap_{t_p\in T_p} ps(t_p)-ps(t_f))^C$,其含义是返回被每个成功测试用例均覆盖且未被失败测试用例覆盖的语句块集的补集.随后,他们提出最近邻(nearest neighbor)法,即,根据指定距离准则(例如海明距离)选出与失败测试用例 t_f 的程序频谱距离最近的成功测试用例 t'_p,其返回的可疑语句块集为 $ps(t_f)-ps(t'_p)$.但上述方法均假设成功测试用例不会执行到缺陷语句,即,没有考虑到缺陷具有偶然正确属性.

但在实际软件测试过程中,配套测试套件 T 中一般包含大量成功测试用例和失败测试用例.因此,很多研究人员尝试借助统计分析方法,通过计算程序实体的怀疑率(即含有缺陷的可能性)来辅助缺陷定位.本文通过分析 SFL 已有的研究成果,给出了如图 1 所示的研究框架.

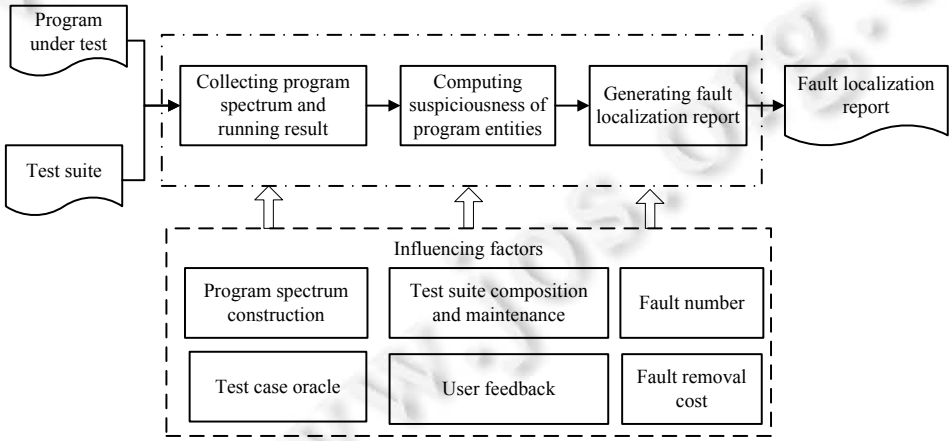

Fig.1 Research framework of SFL

图 1 SFL 研究框架

具体来说,通过在含有缺陷的被测程序上依次执行测试套件中的测试用例,搜集测试用例的程序频谱和执行结果,基于特定模型,计算出各个程序实体的怀疑率,将程序实体按怀疑率取值从高到低进行排序并生成缺陷定位报告.通过分析上述流程,我们进一步识别并总结出多个可影响缺陷定位效果的内在因素,这些因素包括:程序频谱构造方式、测试套件构成和维护、被测程序内部包含的缺陷数、测试用例预言构造、用户反馈以及缺陷修复开销等.其中,

- 程序频谱构造方式决定了对程序行为特征的描述,不同的程序频谱构造方式决定了不同的程序实体类型、程序实体的怀疑率取值、缺陷定位方式和构造开销.
- 测试套件构成因素重点关注其中偶然正确测试用例、相似测试用例和测试用例权重设置对缺陷定位效果的影响.
- 测试套件维护因素关注常见测试套件维护策略(例如测试用例优先级排序方法、测试用例生成方法和测试套件缩减方法等)对缺陷定位效果的影响.
- 被测程序内部包含的缺陷数量因素主要关注两个方面:首先是分析缺陷数量对缺陷定位效果的影响,其次是分析缺陷之间存在的干扰现象对缺陷定位效果的影响.
- 测试用例预言是测试用例的重要构成部分,对该因素的研究也重点关注两个方面:首先是在测试用例预言不存在的场景下,如何进行有效缺陷定位;其次是在自动生成的大量测试用例中,在保障缺陷定位效果时,如何选择出典型测试用例进行测试用例预言的设置.
- 用户反馈因素关注如何充分利用开发人员的测试和调试经验,提出有效的交互式测试框架.
- 缺陷修复开销因素则考虑将缺陷修复的开销纳入到缺陷定位效果的评估中,并分析已有缺陷定位方法对缺陷修复效率的影响.

3 框架内影响因素分析

3.1 程序频谱构造方式

程序频谱因为可以有效描述程序行为特征,所以被研究人员广泛用于辅助程序缺陷定位.Reps 等人在分析千年虫缺陷时,首次提出程序频谱的概念[4].随后,Harrold 等人进一步分析了程序频谱间差异与回归缺陷间的相关性,他们发现:可检测出回归缺陷的测试用例,其在修改前后程序上的程序频谱存在差异的可能性较大;反之则不然[5].目前,程序频谱关注的程序实体类型可以为语句、分支、函数或谓词等,其搜集方式可以是程序实体是否被覆盖过、程序实体的具体覆盖次数或程序实体在执行前后的状态等.

根据程序频谱的构造开销和关注的程序实体类型,本文将已有构造方式分为 3 类:(1) 轻量级程序频谱构造方式.该类程序频谱在构造时仅简单统计测试用例的程序实体覆盖信息.(2) 重量级程序频谱构造方式.该类程序频谱借助重量级程序分析方法,通过分析程序实体之间的控制或数据依赖关系,基于控制流图、程序依赖图、或信息流图以构建复杂程序频谱.(3) 其他程序频谱构造方式.与前两类程序频谱构造方式不同,该类程序频谱在构造时另辟蹊径,例如关注方法调用序列或程序实体的执行时间等.通过对已有的研究成果进行汇总,我们发现:轻量级程序频谱构造方式占主流,其次是重量级程序频谱构造方式,但针对这类构造方式的研究呈上升趋势,而针对其他程序频谱构造方式的研究目前还比较少.

3.1.1 轻量级程序频谱构造方式

轻量级程序频谱构造方式仅需简单统计测试用例的程序实体覆盖信息.当程序实体为语句、语句块或函数时,基于各个测试用例的程序频谱和执行结果,可将程序实体 s 相关的统计数据用一个四元组 $N(s)=\langle n_{ep}(s),n_{ef}(s),n_{np}(s),n_{nf}(s)\rangle$ 来表示.其中,$n_{ep}(s)$ 和 $n_{ef}(s)$ 分别表示覆盖程序实体 s 的成功测试用例数和失败测试用例数,$n_{np}(s)$ 和 $n_{nf}(s)$ 分别表示未覆盖程序实体 s 的成功测试用例数和失败测试用例数.根据四元组 $N(s)$,易知测试套件 T 中所有成功测试用例数 $n_p=n_{ep}(s)+n_{np}(s)$,所有失败测试用例数 $n_f=n_{ef}(s)+n_{nf}(s)$,所有测试用例数 $n=n_p+n_f$.随后,研究人员在设定怀疑率公式时均基于该四元组,且一般基于 Wong 等人总结的如下假设[6]:

- 假设 5:程序实体的怀疑率与该程序实体被成功测试用例覆盖的次数成反比.
- 假设 6:程序实体的怀疑率与该程序实体被失败测试用例覆盖的次数成正比.
- 假设 7:程序实体的怀疑率与该程序实体未被失败测试用例覆盖的次数成反比.
- 假设 8:在设定怀疑率公式时,应该为假设 6 设置更高的权重.

Jones 等人首次提出 Tarantula 怀疑率计算公式[7,8].他们认为,相对于被更多成功测试用例覆盖的程序实体,被更多失败测试用例覆盖的程序实体,其含有缺陷的可能性更高.给定程序实体 s,其计算公式为

$$Tarantula(s)=\frac{n_{ef}(s)/n_f}{n_{ef}(s)/n_f+n_{ep}(s)/n_p} \tag{1}$$

该公式的取值范围介于 0,1 之间,取值为 1 表示怀疑率最高,取值为 0 则表示怀疑率最低.由公式(1)不难发现:与最近邻法[3]相比,Tarantula 法充分考虑了缺陷程序实体偶尔被成功测试用例覆盖到的场景.

当计算出各个程序实体的怀疑率后,开发人员可以将程序实体按怀疑率取值从大到小进行排序并依次审查,直至找到缺陷语句为止.此外,他们还计算出每个程序实体 s 的置信度(confidence)[7],其取值为

$$confidence(s)=\max(n_{ef}(s)/n_f,n_{ep}(s)/n_p) \tag{2}$$

当多个程序实体具有相同怀疑率取值时,测试人员可借助置信度完成对这些程序实体的进一步排序.

Abreu 等人随后提出两个不同的怀疑率计算公式:Jaccard 和 Ochiai.其中,

- Jaccard 怀疑率计算公式[9,10]受聚类分析研究领域的启发,其计算公式为

$$Jaccard(s)=\frac{n_{ef}(s)}{n_f+n_{ep}(s)} \tag{3}$$

- Ochiai 怀疑率计算公式[9,10]受分子生物学研究领域的启发,其计算公式为

$$Ochiai(s)=\frac{n_{ef}(s)}{\sqrt{n_f\times(n_{ef}(s)+n_{ep}(s))}} \tag{4}$$

在实证研究中,他们发现,采用 Ochiai 公式,其缺陷定位效果要优于 Jaccard 公式和 Tarantula 公式.

此外,Wong 等人进一步分析了成功测试用例数对缺陷定位效果的影响.他们认为,语句被成功测试用例覆盖的次数越多,该语句的覆盖次数对怀疑率的贡献度越应逐渐减小[11].为了验证上述观点,他们提出了如下 3 组怀疑率计算公式:

$$Wong1(s)=n_{ef}(s) \tag{5}$$

$$Wong2(s)=n_{ef}(s)-n_{ep}(s) \tag{6}$$

$$Wong3(s)=\begin{cases} n_{ef}(s)-n_{ep}(s), & \text{if } n_{ep}(s)=0,1,2 \\ n_{ef}(s)-2-(n_{ep}(s)-2)\times 0.1, & \text{if } 2<n_{ep}(s)\leqslant 10 \\ n_{ef}(s)-2.8-(n_{ep}(s)-10)\times 0.001, & \text{if } n_{ep}(s)>10 \end{cases} \tag{7}$$

其中,公式(5)仅考虑了语句被失败测试用例覆盖的次数;公式(6)同时考虑了语句被失败测试用例和成功测试用例覆盖的次数;公式(7)则在公式(6)的基础上,根据语句被成功测试用例覆盖的次数不同,将其分成 3 个区间,并分别赋予不同的权重.

随后,Wong 等人提出了 DStar 方法[6].他们通过修改 Kulczynski 系数给出如下的怀疑率计算公式:

$$Dstar(s)=n_{ef}(s)^*/(n_{ef}(s)+n_{ep}(s))\ (*=2,2.5,\ldots) \tag{8}$$

在实证研究中他们发现,DStar 方法要优于其他缺陷定位方法,例如 Tarantula 和 Ochai 等.其缺陷定位效果随指数(*)的取值不断增大而提高,当指数取值超过 33 时,其效果几乎保持不变.

已有的研究一般借助实证研究来比较不同怀疑率公式的缺陷定位效果,但实证研究所得结论受到采用的评测指标、评测程序以及实验平台搭建方式等因素的影响.因此,Naish 等人借助理论分析来进一步比较 33 个不同怀疑率公式,并发现某些公式间存在等价关系[12].他们精心设计出一个简单单缺陷程序片段 If-Then-Else-2(简称 ITE2),该程序可充分体现缺陷定位中的两个重要场景:

(1) 存在噪声,即存在与错误行为强相关的正常行为;

(2) 信号微弱,即错误行为很难被监测到.

基于 ITE2 程序.他们发现一个最优公式 $Op1$ 并予以证明,其计算公式为

$$Op1(s)=\begin{cases} -1, & n_{np}(s)>0 \\ n_{np}(s), & \text{otherwise} \end{cases} \tag{9}$$

随后,在基于西门子套件和 space 程序的实证研究中,他们发现该公式同样优于其他怀疑率公式.

此外,他们还提出另一个最优公式 $Op2$,其计算公式为

$$Op2(s)=n_{ef}(s)-n_{ep}(s)/(n_p+1) \tag{10}$$

该公式通过考虑所有成功测试用例数,可更适用于多缺陷程序的缺陷定位.谢晓园等人同样通过理论分析来比较不同怀疑率计算公式间的优劣[13].他们的理论分析框架基于 4 个假设:每个测试用例均包含测试用例预言、完美缺陷检测假设、测试用例执行结果具有确定性和测试套件能覆盖所有可执行语句.随后,他们通过将被测程序内语句怀疑率与缺陷语句怀疑率进行比较,将所有语句划分为 3 个集合:大于缺陷语句怀疑率的语句集 S_B、等于缺陷语句怀疑率的语句集 S_F 和小于缺陷语句怀疑率的语句集 S_A.最后,通过比较 S_B 内包含的语句数来对不同怀疑率公式进行比较.通过分析 30 个不同怀疑率公式,他们发现了 2 组(共 5 个)最优公式,这 2 组最优公式也包含了 Naish 等人发现的两个最优公式(即公式(9)和公式(10)).

目前,大部分怀疑率公式来自其他研究领域,例如化学、生物信息学和统计学等.Yoo 则另辟蹊径,借助遗传规划(genetic programing),基于 Unix 工具集程序中的 92 个缺陷,在无需人工干预的前提下,推导出 30 个不同怀疑率计算公式[14].随后,谢晓园等人对这 30 个不同公式进行理论分析,发现了其中 4 个最优公式[15].

Wong 等人提出一种基于交叉表的统计方法 Crosstab[16].该方法属于具有良好定义的统计分析方法.首先,搜集测试用例的覆盖信息和执行结果;随后,针对每个语句构造交叉表并计算其怀疑率;最后,将所有语句按怀疑率从大到小进行排序.

交叉表是一种常用的分类汇总表,针对语句 s,其交叉表的构造见表 1.

Table 1 Cross table for statement s

表 1 针对语句 s 构造的交叉表

	Statement s is covered	Statement s is not covered	Σ
Successful executions	$n_{ep}(s)$	$n_{np}(s)$	n_p
Failed executions	$n_{ef}(s)$	$n_{nf}(s)$	n_f
Σ	$n_{ep}(s)+n_{ef}(s)$	$n_{np}(s)+n_{nf}(s)$	n

基于上述交叉表,可提出假设 H0,即,程序执行的结果独立于语句 s 的覆盖.随后,通过计算卡方统计量$\chi^2(s)$来进行假设检验.但在计算$\chi^2(s)$统计量时,随着样本数量的增加,其取值会提高.解决该问题的一种方法是,转而计算语句 s 的列联系数 $M(s)=\dfrac{\chi^2(s)/n}{\sqrt{(row-1)\times(col-1)}}$,其中,$row$ 和 col 分别表示分类变量在行和列上的数目.随后,给出语句 s 的怀疑率计算公式为

$$Crosstab(s)=\begin{cases} M(s), & \varphi(s)>1 \\ 0, & \varphi(s)=1 \\ -M(s), & \varphi(s)<1 \end{cases} \tag{11}$$

其中,$\varphi(s)=(n_{ef}(s)/n_f)/(n_{ep}(s)/n_p)$.当$\varphi(s)=1$ 时,表示语句 s 对成功测试用例和失败测试用例的贡献相同,其怀疑率为 0;当$\varphi(s)>1$ 时,表示语句 s 对失败测试用例的贡献更大,其怀疑率为 $M(s)$.当$\varphi(s)<1$ 时,表示语句 s 对成功测试用例的贡献更大,其怀疑率为$-M(s)$.

也有研究人员通过观察谓词(predicate)在成功测试用例和失败测试用例中取值模式的异常情况,对缺陷定位进行研究.Liblit 等人[17,18]针对部署后软件,搜集软件实际使用过程中产生的信息,随后,通过分析这些信息来确定软件是否异常终止或具有分段故障(segmentation fault),并用于诊断缺陷根源.该方法可以有效避免内部测试的不足,更好地体现软件的实际使用场景,但代码插桩会降低软件运行性能.为解决该问题,借助稀疏的随机抽样.其抽样方法是:程序中任何语句集均可被设置为插桩点,即,每次程序执行到插桩点时,通过模拟伯努利过程随机决定是否插桩.在他们的研究中,主要考虑的插桩对象包括程序分支、函数返回值和动态不变式.基于上述研究工作,Liblit 等人[18]随后提出 CBI(cooperative bug isolation)法来识别出与程序缺陷相关的谓词,即,通过搜集在成功测试用例和失败测试用例中谓词取值是否为真来计算谓词的怀疑率.给定谓词 p,令 $F_T(p)$和 $S_T(p)$分别表示谓词 p 在所有失败测试用例中取值为真的次数和在所有成功测试用例中取值为真的次数,而 $F(p)$和 $S(p)$分别表示谓词 p 被所有失败测试用例覆盖的次数和被所有成功测试用例覆盖的次数,则谓词 p 的怀疑率计算公

式为

$$CBI(p)=\frac{2}{\dfrac{1}{\dfrac{F_T(p)}{F_T(p)+S_T(p)}-\dfrac{F(p)}{F(p)+S(p)}}+\dfrac{1}{\dfrac{\log(F(p))}{\log(|T_f|)}}} \tag{12}$$

该怀疑率计算公式可视为求调和平均数,主要用于平衡两个方面的因素:谓词的特异性(specificity)和谓词的灵敏性(sensitivity).其中,谓词的特异性是指在成功测试用例中谓词没有错误预测到存在的缺陷,灵敏性是指在失败测试用例中谓词被观察到的比例.Arumuga 等人进一步对 CBI 进行扩充,并提出了复合布尔谓词(compound boolean predicate)的概念[19],即,将多个有控制或数据依赖关系的谓词组合起来.Chilimbi 等人则提出了 HOLMES 法[20],该方法进一步采用完全路径(full path)来搜集程序行为特征.他们还提出了一种可迭代的、缺陷导向的方法以减少搜集过程中的执行时间和存储开销.

Liblit 等人提出的 CBI 方法[18]仅简单统计谓词是否被测试用例执行到.Liu 等人发现:在程序的单次执行过程中,大部分谓词可能被多次执行.通过搜集谓词的执行次数,他们提出了 SOBER 方法[21,22].该方法首先搜集谓词 p 在单次执行中取值为真的次数 $n_t(p)$和取值为假的次数 $n_f(p)$,并计算出谓词取值为真的概率 $n_t(p)/(n_t(p)+n_f(p))$;随后,通过依次执行所有测试用例,可以分别统计出在所有成功测试用例和失败测试用例中谓词p取值为真的概率$f_s(p)$和$f_f(p)$;最后,给出谓词 p 的怀疑率计算公式:

$$SOBER(p)=-\log(sim(f_s(p),f_f(p))) \tag{13}$$

其中,sim 函数可计算出$f_s(p)$和$f_f(p)$的相似程度.实证研究结果表明:SOBER 方法通过搜集谓词的执行次数,在一些评测程序上的缺陷定位效果要优于 CBI 方法.郑征等人则通过搜集谓词的判断顺序,提出一种基于谓词执行序列的缺陷定位方法[23].他们发现:通过增大谓词执行信息量的搜集,在一些评测程序上可进一步提高缺陷定位效果.随后他们发现:若固定上述方法中谓词执行信息量的搜集强度,则存在信息利用不足或信息过分利用等问题,并会削弱缺陷相关谓词与程序失效之间的关联度.因此,他们提出一种自适应缺陷定位方法.该方法通过分析测试用例的谓词执行情况来为每个谓词动态选择合适的信息搜集强度.实验结果表明,该方法具有较好的定位效果和定位稳定性[24].

3.1.2 重量级程序频谱构造方式

与简单搜集测试用例的程序实体覆盖信息不同,重量级程序频谱构建需要进一步分析程序实体间的控制或数据依赖关系,搜集并汇总这些依赖关系在程序执行时被测试用例覆盖的情况,以提高缺陷定位效果.但这类方式在提高缺陷定位效果的同时,也需要额外消耗更多的时间和空间开销.

Masri 借助信息流分析来辅助缺陷定位[25],信息流分析针对程序实体间的关系进行建模,其最早由信息安全领域的研究人员提出.Masri 使用的动态信息流分析属于一种重量级程序分析方法,考虑的程序实体间的依赖关系包括:动态直接控制依赖、动态直接数据依赖和 3 种过程间的动态依赖关系.随后,他们提出了基于信息流的 Tarantula 方法,并与基于语句、分支或定义使用对的 Tarantula 方法进行了比较.结果表明:在大部分情况下,基于信息流的 Tarantula 方法要优于其他方法.

张震宇等人发现,布尔表达式求值中的短路现象会影响到缺陷定位效果.因此,他们提出了 DES(debugging through evaluation sequences)方法[26,27].该方法重点分析了短路求值和求值序列对缺陷定位效果的影响,并将每个谓词排名最高的取值序列作为该谓词的排名.实证研究结果表明:DES 方法在仅需花费较小的额外性能开销下就可以提高基于谓词的缺陷定位效果.随后,张震宇等人基于控制流图挖掘出关键语句块链(key block chain,简称 KBC),挖掘出的 KBC 内包含的各个执行序列有着丰富的上下文信息.同时,他们提出了一系列降噪规则.这些规则是能够对大多数怀疑率公式进行降噪的形式化方法.通过运用上述两个关键技术,他们提出了一种新的缺陷定位框架[28,29].

除此之外,张震宇等人[30]还提出了一种基于缺陷传播分析的 CP 模型.通过将程序抽象为基于基本块的控制流图,分别统计控制流图中的边被成功和失败测试用例覆盖的次数的百分比,并以此计算每条边的怀疑率;随后,从控制流图出口节点开始,根据给定的一组公式,后向遍历并递归计算每个基本块的怀疑率;最终,依据所得

基本块的怀疑率列表依次检查并定位缺陷.与 CP 模型类似,虞凯等人[31]提出了 LOUPE 模型,利用多种程序特征谱模型,通过考虑控制依赖与数据依赖关系来捕捉程序的异常行为.在怀疑率估计阶段,针对控制依赖缺陷类型和数据依赖缺陷类型分别建立可疑度计算模型 CDBug 和 DDBug,并利用 Ochiai 公式计算语句的怀疑率.该方法可在预先不知道程序所包含缺陷类型的情况下,取两个模型计算出的怀疑率的最大值作为该语句的怀疑率.

Baah 等人借助概率图模型,以程序依赖图为基础,提出概率程序依赖图 PPDG(probabilistic program dependence graph)[32],通过搜集测试用例的执行信息,可以预测出程序实体间的条件统计相关性和独立性.PPDG 从结构和统计角度可以指出成功测试用例和失败测试用例间的差异程度,同时,其包含的上下文信息可有效推测出缺陷语句所在位置.随后,他们使用因果推理方法对上述工作做进一步的扩展[33,34].

程序切片(program slicing)是代码分析的一种重要工具,因此,也有研究人员将程序切片应用到 SFL 研究中.Agrawal 等人首先提出了动态切片(dynamic slice)的概念及构造方法[35],随后,他们提出一种基于执行切片(execution slice)和砍片(dicing)的缺陷定位方法[36].Wong 等人则同时借助执行切片和语句块间的数据依赖关系来提高缺陷定位效果[37,38].张翔宇等人通过分析动态切片中相关变量取值的变化,提出可削减(pruning)动态切片的缺陷定位方法[39].随后,他们通过更改动态切片相关谓词的输出,进一步提出一种可定位遗漏执行缺陷的方法[40].

毛晓光等人[41,42]首先分析程序的静态后向切片,并通过与执行切片执行集合交操作构造出程序的近似动态切片,并利用统计方法计算语句的怀疑率.文万志等人[43]通过计算程序执行失败时的动态切片,并统计切片中的语句在每次测试过程中被覆盖的次数,基于 Tarantula 公式提出新的怀疑率计算公式.随后,他们提出一种基于层次切片谱的缺陷定位方法.该方法通过依次在不同粒度层次(即包、类、方法和语句)上分析程序内的依赖关系以缩小缺陷语句的查找范围;随后,在此基础上建立了层次切片谱模型,并提出一种新的怀疑率计算方法[44].鞠小林等人[45]同样计算程序执行失败时的动态切片(又称全切片),并与程序执行成功时的执行切片构成混合程序谱,同时,在谢晓园等人提出的理论分析框架[13]基础上提出一种新的最优怀疑率计算公式 HSS.

3.1.3 其他程序频谱构造方式

也有研究人员借助其他程序频谱构造方式来进行缺陷定位,例如考虑面向对象程序特征、测试用例执行时间或同时考虑多种不同程序频谱等.

针对面向对象程序,Dallmeier 等人[46]提出了基于方法调用序列的 Ample(analyzing method patterns to locate errors)方法.他们认为:仅出现在成功测试用例或失败测试用例的方法调用子序列都应该被怀疑;同样,删除的或新增的方法调用子序列也同样应该被怀疑.具体来说,给定一个类 C 以及一个失败测试用例的方法调用序列 c_f 和一个成功测试用例的方法调用序列 c_p,假设方法调用子序列长度设置为 k,则 c_f 和 c_p 可产生的子序列集分别为 $S(c_f)$ 和 $S(c_p)$.然后,对于没有同时出现的子序列 s,将怀疑率 $w(s)$ 设置为 1,否则设置为 0,则类 C 的怀疑率可以用它包含的子序列的怀疑率的均值进行计算,即

$$Ample(C)=\frac{\sum_{s\in S_c} w(s)}{|S_C|} \tag{14}$$

其中,$S_C=S(c_f)\cup S(c_p)$.Ample 方法也可以通过多组成功测试用例来计算 $S(c_p)$,并用于怀疑率计算.在实证研究中他们发现,子序列长度 k 的取值会影响到实际定位效果.他们建议,将 k 的取值限定在 5~10 之间.徐宝文等人同样考虑了面向对象程序特征(例如封装、继承等)对缺陷定位效果的影响,提出了一种结合类可疑度信息的缺陷定位方法[47].

Yilmaz 等人[48]则另辟蹊径,借助时间频谱来辅助缺陷定位,并提出了 TWT(time will tell)方法.他们首先搜集成功测试用例和失败测试用例的时间频谱,并基于成功测试用例的时间频谱构建模型;随后,在该模型上评估失败测试用例和成功测试用例间的时间频谱差异,即,若发现一种方法在失败测试用例中的执行时间比在成功测试用例中的执行时间长或者短,则该方法很可能与程序缺陷相关.

Santelices 等人[49]分析了程序实体粒度的设置对缺陷定位效果的影响.具体来说,他们考虑了 3 种不同的程序实体粒度:语句覆盖、分支覆盖和定义使用对,并比较了三者之间的缺陷定位效果.结果表明:并不存在一种实

体粒度,其缺陷定位效果总是优于其他实体粒度.基于上述结果,他们提出了一种可同时考虑多个程序实体粒度的缺陷定位方法.首先计算出每个程序实体的怀疑率,对于分支或定义使用对,借助 3 条规则,可以将怀疑率取值映射到相关联的语句上;在完成上述映射后,提出了 3 种不同的组合策略(即 max-SBD,avg-SBD 和 avg-BD),最终计算出每个语句的可疑率.实证研究结果表明:虽然 max-SBD 策略并不比使用单一覆盖类型的效果有效,但 avg-SBD 和 avg-BD 策略都要优于单一覆盖类型,且具有统计显著性.他们的研究工作表明,综合使用多种程序实体有助于提高缺陷定位效果.同样地,Debroy 和 Wong 发现,不存在一种缺陷定位方法,在所有评测程序上的定位效果均是最优.他们提出一种基于多数同意的策略,可以将多种不同缺陷定位方法的结果有效地进行组合[50].

Perez 等人发现:在搜集测试用例的程序频谱时,对被测程序进行语句级别的代码插桩(code instrument)时存在计算开销较大的问题.因此,他们提出一种动态代码覆盖(dynamic code coverage)方法,可以有效缓解上述代码插桩开销.以 Java 编程语言为例,他们首先在粗粒度级别(即包级别)对代码进行插桩,当识别出可疑程序模块后,对这些程序模块依次在更细粒度级别(例如可依次考虑类级别和方法级别)进行代码插桩,直到最终在可疑方法内完成语句级的代码插桩并定位到可疑语句上[51].结果表明,该方法可以平均缩减 29%的执行时间和 77%的缺陷定位报告规模.

徐宝文等人[52]提出了 SIQ 方法.该方法将程序覆盖信息和测试用例执行结果均视为事件,随后计算各个事件的信息量,并结合语句的执行信息,动态调整其在怀疑率计算中的权重,从而提高缺陷定位效果.他们的实证研究结果表明:与已有方法相比,SIQ 方法表现出更好的缺陷定位效果及稳定性.

3.2 测试套件构成和维护

3.2.1 测试套件构成的影响

本节主要分析测试套件构成对缺陷定位效果的影响,重点分析偶然正确测试用例的影响、相似测试用例的影响和测试用例权重设置的影响.

缺陷定位效果会受到偶然正确测试用例的影响.根据 Voas 提出的 PIE 模型[53]可知:若测试用例触发软件失效,则必须满足 3 个必要条件:

(1) 测试用例执行到缺陷语句;

(2) 缺陷语句的执行造成随后程序内部状态出错;

(3) 错误的内部状态通过传播影响到程序的输出.

而偶然正确测试用例仅保证满足必要条件(1)和(2),而必要条件(3)不满足.为缓解偶然正确测试用例对缺陷定位效果的影响,其研究面临如下的挑战:首先,我们无法事先预知缺陷语句在被测程序内的准确位置;其次,我们无法从成功测试用例 T_p 中精确识别出偶然正确测试用例.

王欣明等人假设开发人员预先知道被测程序的缺陷类型,并通过这些缺陷类型的模式来排除偶然正确测试用例对缺陷语句的覆盖[54].具体来说,他们借助上下文模式(context pattern)对测试用例的覆盖信息进行提炼:

- 首先,基于后向动态切片移除掉与程序输出结果无关的程序实体,但上述方法不能捕获诸如遗漏语句之类的缺陷.他们发现:当缺陷语句被执行并引发缺陷时,执行前后的动态控制流和数据流一般与特定模式匹配,他们称其为上下文模式.
- 随后,他们借助事件表达式和扩展有限状态机描述了 12 个经典上下文模式.这些模式可用于匹配常见的 13 类缺陷类型.
- 最后,借助 Tarantula 怀疑率计算公式算出程序实体的怀疑率,并返回缺陷定位报告.

Masri 等人提出了多种可从成功测试用例 T_p 中识别出偶然正确测试用例的启发式方法[55],但这类方法存在一定的错判率.贺韬等人同样发现,偶然正确测试用例的存在会降低缺陷定位效果.因此,他们提出了 Muffler 方法.该方法使用程序变异分析来降低偶然正确测试用例的影响,并最终提高缺陷定位结果[56].与已有的工作相比,该方法不需要预先知道被测程序内的缺陷类型,也不需要从成功测试用例中识别出偶然正确测试用例.张震宇等人同样考虑了偶然正确测试用例对缺陷定位效果的影响,但与上述方法不同,他们在执行缺陷定位时仅使用失败测试用例并借助趋势预测来计算语句的怀疑率[57].即,通过搜集失败测试用例的覆盖信息,采用最小二乘

拟合,为每个语句构造拟合直线,并基于该拟合直线来计算相应语句的怀疑率.

郝丹等人重点分析了相似测试用例对缺陷定位效果的影响,并提出 SAFL 方法[58].假设被测程序包含 m 个可执行语句,他们首先搜集测试用例的覆盖信息和执行结果,构造出执行矩阵 $E=(e_{ij})$,其中,e_{ij} 表示第 i 个测试用例是否覆盖到第 j 个语句,$e_{i(m+1)}$表示第 i 个测试用例的执行结果;随后,根据执行矩阵,构造出量化矩阵 $F=(f_{ij})$,其中,f_{ij} 表示第 j 个语句对第 i 个测试用例的隶属度,根据粗糙集理论,第 j 个语句的怀疑率可通过计算其在失败测试用例和所有测试用例中的隶属度的比值进行确定,其取值越大,表示该语句含有缺陷的可能性越高.具体计算公式为

$$SAFL(j)=\frac{\sum_{k=1}^{m}\max(\{f_{ik}\mid e_{ij}>0\wedge e_{i(m+1)}=0\wedge 1\leqslant i\leqslant n\})}{\sum_{k=1}^{m}\max(\{f_{ik}\mid e_{ij}>0\wedge 1\leqslant i\leqslant n\})} \tag{15}$$

Wong 等人认为:随着测试用例数的增加,成功测试用例的权重应该分阶段逐渐减少.他们提出了 3 组怀疑率公式:Wong1,Wong2 和 Wong3[11].此后他们认为:失败测试用例的权重也应该分阶段逐渐减少[59],即,语句的怀疑率为覆盖该语句的失败测试用例权重之和减去覆盖该语句的成功测试用例权重之和.实证研究结果表明,上述两种策略均可在一定程度上提高缺陷定位效果.随后,徐宝文等人提出一种通过增大边际权重来进行缺陷定位的方法[60].该方法将失败测试用例的边际权重引入到怀疑率计算过程中,即,针对某一程序实体,令失败测试用例的权重随着对该程序实体覆盖次数的增加而提高.

3.2.2 测试套件维护方法的影响

本节主要分析常见测试套件维护方法对缺陷定位效果的影响,重点分析测试用例优先排序方法、测试用例生成方法和测试套件缩减方法等.

测试用例优先级排序(test case prioritization,简称 TCP)通过设定特定排序准则对测试用例进行排序,以优化其执行次序,其中,一种典型优化目标是最大化测试套件的早期缺陷检测能力.目前,已有的研究工作主要集中在两个方向:

(1) 分析已有的 TCP 方法对缺陷定位效果的影响;

(2) 以提高缺陷定位效果为优化目标来指导测试用例的排序.

针对第 1 个研究方向,姜博等人重点考察了 TCP 方法的内在影响因素对缺陷定位效果的影响,重点考察的因素包括排序策略、程序实体粒度设置和测试时间.结果表明:

(1) 当测试时间有限时,TCP 方法是辅助缺陷定位的一种有效方法;

(2) 在完成测试用例排序后,若执行的测试用例数较少,则对缺陷定位效果的影响较大;

(3) 在已执行的测试用例中,若包含的失败测试用例较多,则有助于提高缺陷定位效果;

(4) 在考虑的排序策略中,随机策略效果最好,Additonal 策略和自适应随机策略其次,Total 策略最差;

(5) 测试用例执行时搜集的程序实体粒度对缺陷定位效果的影响不显著[61−63].

随后,姜博等人进一步分析了测试充分性准则这一因素对缺陷定位效果的影响[64,65],他们发现:满足 MC/DC 覆盖准则的测试套件的缺陷定位效果要优于分支覆盖准则,而满足语句覆盖准则的测试套件的缺陷定位效果最弱.

针对第 2 个研究方向,Gonzalez-Sanchez 等人提出一种可提高缺陷定位效果的基于信息增益的 TCP 方法.但该方法存在如下不足:

(1) 对方法中的参数取值需要精确估算;

(2) 算法复杂度呈指数级;

(3) 采用在线方式完成测试用例的排序,即,在选出一个测试用例时,需要分析已选择测试用例的执行结果[66,67].

为了解决上述问题,他们提出了 RAPTOR 方法.该方法在对测试用例进行排序时,考虑了对相似语句执行模式的约简[68].Yoo 等人[69]则认为,TCP 方法和缺陷定位可以互为补充.他们提出了如下迭代过程:首先,按序执行测试用例,直到检测到首个内在缺陷为止;随后,修复该缺陷并继续按序执行余下测试用例,直到检测到下一个

缺陷为止.针对上述迭代过程,他们提出了缺陷定位优先级(fault localization prioritization)问题,即,在发现缺陷后,对余下的测试用例如何设定最优排序策略,以最大化早期缺陷检测能力.他们借助香农信息论提出 FLINT 方法.该方法不仅可以对语句进行排序,而且还可以对测试用例进行排序.

传统的测试用例生成方法研究以最大化代码覆盖率为目标;而面向缺陷定位的测试用例生成方法研究则以提高缺陷定位效果为目标,并重点解决生成的测试套件规模与缺陷定位效果间存在的折中问题.Baudry 等人[70]通过分析有效缺陷定位时所需的信息,提出动态基本语句块的概念,并基于动态基本语句块,提出面向调试的测试(test-for-diagnosis)准则,最终提出的解决方案可有效缓解测试套件规模和缺陷定位效果间存在的折中问题.该方案以提高动态基本语句块的数量为目的,来逐步添加测试用例.Artzi 等人[71]首先提出多种测试用例相似性准则,以度量两个测试用例的执行特征的相似性;随后提出相应的有导向的测试用例生成方法,可生成与失败测试用例的执行特征相似的测试用例.实证研究结果表明:基于 Ochiai 怀疑率公式,相对于以最大化代码覆盖率的有导向方法,基于路径约束(path constraint)相似性准则的有导向方法,在生成的测试套件规模和构造时间上均具有较大优势.Campos 等人同样认为,已有 SFL 方法的缺陷定位效果取决于配套测试套件的规模和包含的测试用例的多样性.他们基于信息熵来指导测试用例的生成并提出 ENTBUG 方法,该方法融合基于搜索的测试用例生成方法,同时借助信息熵来指定适应值函数和遗传算法的设定[72].

测试套件缩减(test suite reduction,简称 TSR)在满足对指定测试需求的覆盖前提下,通过识别并移除冗余测试用例来降低测试开销.郝丹等人在分析一些 SFL 方法(例如 Dice 法[36]和 Tarantula 法[7,8])时发现,测试套件中存在的冗余测试用例可能会降低这些 SFL 方法的缺陷定位效果.因此,有必要在执行缺陷定位时使用 TSR 方法对测试套件进行预处理[73].随后,Yu 等人进一步深入分析了测试套件缩减对缺陷定位效果的影响[74].除了传统的基于语句的 TSR 策略,他们还额外考虑了基于向量(即,考虑语句执行次序)的 TSR 策略.实证研究结果表明:在执行基于语句的 TSR 策略时,测试套件规模和缺陷定位效果间存在折中现象.即,虽然大幅度缩减了测试套件规模,但也会显著降低缺陷定位效果;而在执行基于向量的 TSR 策略时,测试套件规模的缩减程度虽然较小,但对缺陷定位效果的影响可以忽略不计,甚至有时会略微提高.

3.3 缺陷数量

在实际软件的开发和维护过程中,当被测程序内部含有多个缺陷时,已有怀疑率公式(例如 Tarantula 等)在计算程序实体怀疑率时均假设这些缺陷之间具有独立性,即,某一失效的产生仅与一个缺陷相关.但该假设很难成立.Jones 等人[7]首先分析了缺陷定位效果与缺陷数量的关系,他们初步发现,缺陷定位效果与缺陷数量成反比关系.随后,DiGiuseppe 等人在实证研究中基于 3 个程序生成 13 000 个多缺陷版本,发现上述结论并不明显成立,即,缺陷数量对缺陷定位效果的影响可以忽略不计[75].

Debroy 等人则深入分析了缺陷间干扰现象[76],假设被测程序内部包含缺陷 f_1 和 f_2,他们将该现象细分为两类:

(1) 构造性干扰,即,若 f_1 或 f_2 单独存在,则测试用例 t 不会触发软件失效;若 f_1 和 f_2 同时存在,则测试用例 t 会触发软件失效.

(2) 破坏性干扰,即,若 f_1 或 f_2 单独存在,则测试用例 t 会触发软件失效;若 f_1 和 f_2 同时存在,则测试用例 t 不会触发软件失效.

DiGiuseppe 等人也深入分析了缺陷间的干扰现象[75].他们在实证研究中均确认了这种现象的大量存在.失效聚类(failure cluster)是一种可有效缓解该现象的方法,其核心思想是:将失败测试用例进行聚类,使得同一聚类内的测试用例与同一缺陷有关.在 Liblit 等人[17,18]的研究工作基础上,Zheng 等人发现,多缺陷问题受到谓词的稀疏采样性和谓词间的交互复杂性的影响.他们基于双聚类算法,提出一种可迭代的集体投票机制,完成对失败测试用例的聚类[77].Liu 等人发现,与同一缺陷相关的失败测试用例的程序频谱可能差异性较大.他们提出一种新的失败测试用例相似性度量指标,即,给定缺陷定位方法,若两个失败测试用例的程序频谱建议到相似缺陷定位效果,则认为这两个失败测试用例相似[78,79].Jones 等人借助并行计算的思想来减小缺陷定位开销:当被测程序内部包含多个缺陷时,他们首先将失败测试用例按针对的缺陷执行聚类分析;随后,依次将各个聚类内的失败

测试用例与所有成功测试用例组合成新的测试套件,并分配给不同的开发人员,以支持同时调试多个缺陷[80];随后,Servant 等人进一步将缺陷提交给相关开发人员进行修复[81].

基于模型的调试技术通过分析逻辑关系,可构建出一个从失败测试用例运行信息开始推理的模型.其优点在于可以精确定位到出错原因,但其缺点是检测代价大,需要借助更多上下文信息来进行辅助.Abreu 等人将基于模型的调试技术与 SFL 方法相结合,提出一种基于贝叶斯推理方法来解决多缺陷定位问题.该方法可以推断出候选程序实体及包含缺陷的概率,从而为开发人员提供了丰富的调试信息.但该方法本身具有指数级复杂度,为减少计算开销,他们提出一种启发式方法,可以快速生成需要优先审查的程序实体[82,83].文万志等人[84]结合程序切片技术提出了一种基于条件执行切片谱的多缺陷定位方法.这种方法根据不同的谓词条件进行分类,最终降低了因不同缺陷引起的失败运行的叠加影响.

3.4 测试用例预言

测试用例预言(test case oracle)通过提供测试用例的预期结果与实际结果的对比机制来判断测试用例的执行结果是成功还是失败.已有的 SFL 方法均假设每个测试用例包含测试预言,但该假设在一些程序(例如编译器程序、机器学习算法、仿真程序等)上很难成立.在这类程序中,大部分测试用例需要借助手工方式来构造测试用例预言,且构造代价较为高昂.

蜕变测试(metamorphic testing)是部分解决测试用例预言问题的一种有效方法:通过分析被测程序特征,可获得一组蜕变关系(metamorphic relationship);在已有测试用例的基础上,根据蜕变关系生成一组新的相关测试用例;执行这些测试用例,并检查测试用例的输出与已有测试用例的输出是否满足特定蜕变关系.谢晓园等人基于蜕变测试和程序切片提出了蜕变切片(metamorphic slice)的概念,通过搜集测试用例的蜕变切片并分析蜕变关系的违反情况来辅助缺陷定位[85,86].同样地,毛晓光等人[87]在他们提出的基于后向切片的统计缺陷定位方法[41,42]基础上提出了蜕变后向切片(metamorphic backward slice)的概念.上述研究工作均可部分地解决测试用例预言不存在的问题.

一些研究人员则从另一个角度来关注测试用例预言问题.他们发现:虽然借助一些自动测试用例生成技术可以生成大量测试用例,但测试预言却需要手工构造并且代价高昂.为了在保证缺陷定位效果的前提下选出少量典型测试用例进行测试预言构造,张洪宇等人提出了一种基于多样性最大化加速(diversity maximization speedup)的策略,可有效选出这些典型测试用例[88].郝丹等人则假设存在一个失败测试用例,他们在搜集完自动生成的所有测试用例的覆盖信息后,提出 3 种不同的测试用例选择策略来选出这些典型测试用例.结果表明:这 3 种策略均可以在不显著影响缺陷定位效果的前提下,仅需为选出的少量(低于 10%)测试用例设置测试用例预言[89].

3.5 用户反馈

传统的手工调试方式通过选择可疑语句、对其设置断点、观察相关变量取值,最终确定缺陷语句,其调试效果与开发人员的经验和调试技术密切相关.为提高传统手工调试方法效果,郝丹等人提出交互式测试框架[90]:

(1) 基于已有缺陷定位方法,每次为开发人员推荐一个可疑语句,开发人员随后借助手工调试方式判断该可疑语句是否是缺陷语句:若不是缺陷语句,则继续判断缺陷语句是否已先于该语句执行;随后,基于上述反馈信息,该框架重新计算各个语句的怀疑率并继续推荐新的可疑语句.

(2) 针对开发人员可能会对可疑语句做出错误的判定,提出一种鲁棒性方法,可确保该框架在推荐时不会遗漏真正缺陷语句.

同样地,张洪宇等人也提出一种交互式缺陷定位框架.在该框架内,不需要开发人员判断框架推荐的审查语句是否已先于该语句执行,而仅需用户将该审查语句标记为有缺陷或无缺陷;随后,利用该反馈信息计算剩余语句的怀疑率[91].

3.6 缺陷修复开销

大部分研究工作在评估缺陷定位效果时以需要审查的代码数占所有代码数的比例作为评测指标,其取值

越低,表示缺陷定位效果越好.但该评测指标基于完美缺陷检测假设.该假设认为,开发人员在缺乏对可疑语句上下文的理解情况下,通过独立审查,即可判断该语句是否为缺陷语句并进行有效缺陷移除.但 Parnin 等人发现:对于开发人员来说,理解并移除缺陷的调试行为具有高复杂性;同时他们发现:在实际软件调试过程中,自动缺陷定位方法的效果并不一定优于传统的手工调试方法[92].

程序自动化修复技术借助特定修改规则可以完成对一些可疑语句的修复,因此,可以借助 SFL 方法来提高程序修复效率.毛晓光等人提出一个新的评测指标 NCP(number of candidate patches),其返回程序修复过程中,在寻找到有效补丁前生成的候选补丁数,其取值越低,表示所采用的 SFL 方法在程序修复过程中的效果越好.他们在 GenProg 工具[93]基础上考虑了 15 种不同的 SFL 方法,在实证研究中发现:需要审查代码数较少的 SFL 方法,并不一定可以在代码修复过程中取得更好的效果[94].

Debroy 等人借助变异分析(mutation analysis),不需要人为干涉就可以自动完成缺陷修复.他们首先借助特定的 SFL 方法将语句按含有缺陷的可能性进行排序;随后,按序在语句上执行变异操作、生成变异体,直到变异体是正确的程序版本为止,实证研究结果表明:借助他们采用的 8 种变异算子(mutation operator),可以正确修复其中 20.7%的缺陷;随后,它们分析了 Tarantula 和 Ochiai 这两种怀疑率计算公式对缺陷修复效率的影响,发现 Ochiai 公式除了具有更好的缺陷定位效果以外,在他们所提出的方法中也具有更高的缺陷修复效率;最后,它们分析了变异算子的选择对缺陷修复效率的影响,发现与算数运算符、逻辑运算符和关系运算符相关的变异算子具有更好的缺陷修复效果[95].

4 已有实证研究分析

虽然 Naish 等人[12]和谢晓园等人[13]借助理论分析对缺陷定位效果进行了分析,但理论分析一般建立在一系列假设的基础上,且这些假设在实际的软件测试中并不一定满足.因此,绝大部分研究人员采用实证研究的方式来验证所提出的解决方案的有效性.接下来,依次总结实证研究中采用的评测指标和常用评测程序.

4.1 评测指标

根据不同的代码审查策略,研究人员提出了基于程序依赖图的评测指标 T-Score 和基于语句排序的评测指标 Score 和 Expense.

当缺陷定位报告返回的是可疑语句集 R 时,研究人员提出基于程序依赖图(program dependence graph,简称 PDG)的评测指标 T-Score[3].程序依赖图可同时描述程序内的控制依赖和数据依赖关系.针对语句 n,首先定义 k 依赖集 $DS_k(n)$.该集合包括从语句 n 出发,采用广度优先搜索策略,前向或后向遍历依赖边,且所有距离为 k 的语句集.假设 $DS_*(R)$为包含缺陷语句的最小依赖集,则该评测指标的计算公式为

$$1-|DS_*(R)|/|PDG| \tag{16}$$

其中,$|PDG|$返回对应程序依赖图内包含的所有节点数,其取值越接近 1,表示定位效果越好;其取值越接近 0,则表示定位效果越差.

假设一个简单缺陷程序的 PDG 如图 2 所示,其中,e_1 为缺陷语句.同时存在 3 个缺陷定位报告,分别是 $R_1=\{e_2,e_4\}$,$R_2=\{e_2\}$,$R_3=\{e_8\}$,则根据评测指标计算公式(16),可分别计算出 $DS_*(R_1)=\{e_1,e_2,e_3,e_4,e_8\}$,其缺陷定位报告 R_1 的评测分为 0.375;$DS_*(R_2)=\{e_1,e_2,e_3,e_4\}$,其缺陷定位报告 R_2 的评测分为 0.5;$DS_*(R_3)$包含 PDG 中所有语句,其缺陷定位报告 R_3 的评测分为 0.因此,我们可以认为缺陷定位报告的定位效果:$R_2>R_1>R_3$.

在基于语句排序的评测指标中,研究人员将程序实体按怀疑率的取值从高到低进行排序,并依次审查,直至找到缺陷语句.假设被测程序 $P=\{e_1,e_2,\ldots,e_k\}$,缺陷程序实体为 e_f,一种指标是返回缺陷程序实体 e_f在排序列表中所处的位置(即 *rank* 值).当存在 $m(m>0)$个程序实体与缺陷程序实体 e_f 的怀疑率取值相等时,目前有两种方式计算 *rank* 值[9,10,96].假设怀疑率取值高于 e_f的程序实体数为 l,则

(1) 方式 1 考虑最坏情况,即,将缺陷程序实体的 *Rank* 值 $Rank(e_f)$设置为 $l+m+1$.

(2) 方式 2 考虑平均情况,即,将缺陷程序实体的 *Rank* 值 $Rank(e_f)$设置为 $l+(m+1)/2$.但方式 2 可能存在漏报问题.

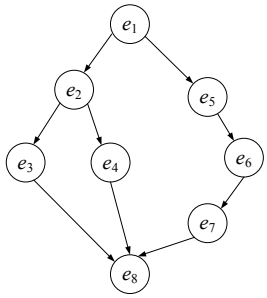

Fig.2 Program dependence graph of a faulty program

图 2 缺陷程序对应的程序依赖图

也有研究人员基于缺陷程序实体的 *Rank* 值,从百分比的角度出发提出两种互补的评测指标 *Score* 和 *Expense*[80],其中,

- *Score* 评测指标返回不需要审查的程序实体占所有程序实体的百分比,其计算公式为

$$Score=(1-Rank(e_f)/k)\times 100\% \tag{17}$$

- *Expense* 评测指标(在有的文献中也称为 EXAM 指标[59])则返回在找到缺陷语句 e_f 前必须审查的程序实体占所有程序实体的百分比,其计算公式为

$$Expense=(Rank(e_f)/k)\times 100\% \tag{18}$$

4.2 评测程序

在缺陷定位研究中,研究人员一般借助实验室或开源软件界的评测程序进行研究.我们对论文中采用的评测程序进行了汇总,并罗列出使用次数超过 1 次的所有评测程序,最终结果见表 2.表 2 依次罗列了程序名称、程序规模、程序功能描述、首次使用时间和累计使用次数等,其中,将代码行数超过 10 000 行的程序称为大规模程序,将代码行数在 1 000 行~10 000 行之间的程序称为中规模程序,将代码行数小于 1 000 行的程序称为小规模程序.例如,通过表 2 我们可知:space 程序属于中等规模程序,首次使用时间为 2002 年,截止到目前为止,累计使用次数为 31 次,其中大部分实证研究采用的实验对象均可以从内布拉斯加大学林肯分校(University of Nebraska-Lincoln)的 SIR 库中下载[97].在表 2 中,我们将评测程序按照累计使用次数从高到低进行排序.

Table 2 Subjects used in previous empirical studies

表 2 已有实证研究中使用的评测程序

Name	Size	Description	First used	Cumulative number
Siemens suite	Small	Small programs developed by Siemens	2003	65
space	Medium	An interpreter for an array definition language	2002	33
grep	Large	Searches input files for a pattern	2006	28
gzip	Medium	Data compression/decompression	2009	25
sed	Large	Text processor	2009	24
flex	Large	A lexical analyzer generator	2009	18
nanoXML	Medium	A small XML parser for Java	2005	8
ant	Large	A Java-based build tool	2008	8
XMLsecurity	Large	Implements security standards for XML	2008	8
Unix utity	Small	Unix utities	2008	6
make	Large	Build manager for C programming language	2010	5
Jtopas	Medium	Facilitates users to tokenize and parse arbitrary text data	2011	5
Jmeter	Large	Performance testing tool	2011	5
BC	Large	A calculator program that accepts scripts written in the bc language	2006	4
DC	Medium	Desk calculator	2005	2
TCC	Large	Tiny C compiler	2005	2
Jaligner	Medium	An algorithm for biological local pair-wise sequence alignment	2010	2
seqmap	Medium	A bioinformatics tool in C++	2011	2
SimpleJavaApp	Small	A displaying and editing book listing application	2012	2
Tetris	Medium	Teris game	2012	2
JHSA	Large	A JAVA hierarchical slicing tool	2012	2

我们根据表 2 对评测程序的使用趋势进行分析,得出如下结论:

(1) 西门子套件(Siemens suite)是研究人员最为认可的评测程序,超过 80%的论文均采用西门子套件的程序作为主要评测程序.西门子套件共包括 7 个小规模 C 语言程序,其中,tcas 是防止航空器空中相撞系统,schedule 和 schedule2 是优先级调度器,tot_info 针对指定的输入数据生成统计信息,print_tokens 和 print_tokens2 是词法分析器,replace 程序完成模式匹配和替换.这些程序早期被西门子研究院用于研究控制流和数据流覆盖准则的缺陷检测能力[98].

(2) 但西门子套件中包含的 7 个程序均是小规模程序,同时,缺陷采用手工注入方式,难以体现实际开发过程中的软件缺陷真实特征.为确保实证研究的结论具有一般性,绝大部分研究人员会额外采用一些中等规模甚至大规模的程序,其中常采用的程序包括 space 程序和 Unix 工具程序(包括 grep,gzip,sed 和 flex).其中,space 程序为欧洲航天局开发的针对数组定义语言(array definition language,简称 ADL)的一个解释器,它共包含 6 000 多行可执行代码、1 万多个测试用例、38 个单缺陷版本,每个缺陷版本都来自于软件的实际开发过程,因此,这些缺陷均具有一定的代表性[7].在 Unix 工具程序中,grep 程序实现了在某一文件内匹配指定模式,它包含 9 000 多行代码、809 个测试用例、17 个单缺陷版本;gzip 程序实现了数据的压缩和解压缩,它包含 5 000 多行代码、217 个测试用例、55 个单缺陷版本;sed 程序实现了文件处理,它包含了 9 000 多行代码、370 个测试用例、17 个单缺陷版本;flex 程序实现了词法分析功能,它包含了 1 万多行代码、567 个测试用例、21 个单缺陷版本[30].在采用上述 C 语言编程实现的评测程序进行实证研究时,绝大部分研究人员借助 gcov 工具完成代码的插桩和测试用例覆盖信息的搜集.

(3) 随着面向对象软件的日益增多以及基于 Java 的相关开源工具的不断出现(例如,程序分析工具 Soot 和 WALA 等,代码覆盖率搜集工具 EclEmma,Cobertura 和 codecover 等,测试用例生成工具 Randoop 和 EvoSUITE 等),研究人员也逐渐在实证研究中采用 Java 语言实现的评测程序,其中以 NanoXML,Ant,XML-security,Jtopas 和 Jmeter 为代表.其中,NanoXML 实现了一个轻量级 XML 解析器,包含约 8 000 行代码;Ant 实行了一个基于 Java 的程序构建工具,其功能类似于 Unix 操作系统中的 make 工具;Jmeter 实现了一个 Java 桌面应用程序,用于执行性能测试;XML-security 实现了 XML 中的安全协议;JTopas 实现了一个用于解析文本数据的 Java 类库.后 4 个程序的规模在 2 000 行~80 000 行之间,且在 SIR 库中均配备了大量基于 Junit 测试框架的测试用例.除此之外,还有一部分评测程序来自软件企业或开源社区,例如,Abreu 等人采用了一个嵌入式软件系统[99],Jones 等人采用了一个面向方面的框架 AspectJ[81].他们通过挖掘 AspectJ 的代码历史存档,共提取出 10 454 个版本,其中每个版本平均配备 1 585 个测试用例.

5 特定应用领域的缺陷定位方法研究

目前,对缺陷定位的研究已经从传统的软件测试领域拓展到到不同特定应用领域,包括并发程序、Web 应用、SQL 数据库应用、Web 服务、嵌入式系统和回归测试等.

部署在多核 CPU 上的并发系统给传统缺陷定位提出了新的挑战.Park 等人针对线程间与数据访问模式(data- access pattern)相关的缺陷进行了深入研究,并开发出 Falcon 工具.具体来说:首先,对线程间的内存访问序列进行监测;随后,搜集测试用例覆盖的数据访问模式和分析执行结果;最后,基于 Tarantula 怀疑率公式计算出各个数据访问模式的怀疑率[100].随后,他们开发出 Unicorn 工具,该工具可以识别出造成并发缺陷的可疑内存访问,并对其按怀疑率大小进行排序[101].

Artzi 等人针对 Web 应用开发出 Apollo 工具.在该工具内,他们将 Tarantula,Ochiai 和 Jaccard 等怀疑率计算公式应用到 PHP 编程语言实现的 Web 应用中,同时,提出了多种有效的测试用例生成策略以提高缺陷定位效果[71,102,103].蔡开元等人为提高图形用户界面(graphical user interface,简称 GUI)软件的缺陷定位效果,提出了一种新颖的测试用例生成方法.具体来说,他们设计出 4 种新颖变异算子对已有 GUI 失败测试用例中的事件序列执行变异操作,以生成新的测试用例[104].

Clark 等人对数据库应用中的 SQL 相关缺陷的定位进行了研究[105],应用程序借助 SQL(structured query

language),可以完成对数据的查询、更新、添加和删除.SQL 命令不仅内部包含控制依赖和数据依赖,而且其执行结果会影响程序随后的计算结果和执行路径.他们提出的方法可以对应用程序中的 SQL 命令的执行进行监控,并完成对 SQL 命令和属性怀疑率的计算.

BPEL(business process execution language)是一种广泛使用的 Web 服务组合语言,其包含特有的语法和语义.孙昌爱等人基于 Tarantula 怀疑率公式提出两种框架,可有效定位 BPEL 中的集成缺陷和交互缺陷[106].

针对嵌入式系统,Azzeddine 等人提出了相应的缺陷定位方法并集成到 CoMET 工具中.他们借助 While 模型抽象出嵌入式系统内在的循环特性,从而将整个系统的一次失败执行转化为 While 模型中循环体的多次执行,并将最后一次执行视为失败执行,从而构建出 While 模型中循环体的程序频谱,进而利用 Tarantula 和 Ochiai 等怀疑率计算公式,计算出循环体中每个程序实体的怀疑率[107].

因为现代软件具有内在的动态性、可配置性和可移植性,当软件完成部署后,可能会产生在内部测试时未触发的新失效,Jin 等人称这类失效为部署后失效(field failure).部署后失效具有难以预测和重现的特点,因此,他们提出一种新颖的方法 BugREDUX.该方法可以在内部测试时重现部署后失效,且不需要用户付出额外开销,并可有效避免侵犯用户隐私[108].在上述研究工作的基础上,他们进一步提出 F^3(fault localization for field failures)方法来定位这类缺陷.该方法可以:(1) 生成多个与部署后失效相似的失败测试用例和成功测试用例;(2) 可以有效地融合已有的缺陷定位方法[109].

在软件开发和维护的过程中,因移除软件内在缺陷、完善已有功能、代码重构或提高运行性能均会执行代码修改.回归测试可用于保证代码修改的正确性,并避免代码修改对其他程序模块产生的副作用.虞凯等人将回归测试中因代码修改而无意中引入的缺陷统称为回归缺陷,他们对 Delta Debugging 方法[110,111]应用到回归缺陷定位时的效果进行了评估,结果发现:在定位大概 2/3 的回归缺陷时,通过 Delta Debugging 方法隔离出的代码可以为缺陷定位提供直接或间接的有益线索;而在定位其他回归缺陷时,则存在隔离出的代码过多,甚至有时无关的问题[112].随后,他们结合覆盖率分析和 Delta Debugging 方法提出一种新的方法,可有效隔离出与缺陷相关的代码修改[113].张令明等人则提出了 FaultTracer 方法,该方法将代码修改视为一系列原子代码修改(atomic change),结合 SFL 方法和修改影响分析(change impact analysis)方法,将原子代码修改按照怀疑率取值进行排序.结果表明:仅需审查少量原子代码修改即可定位到缺陷语句[114,115].随后,在 FaultTracer 方法的基础上,他们借助变异分析来进一步提高缺陷定位效果[116].

6 未来研究展望

根据上述分析可以看出:基于程序频谱的动态缺陷定位问题已经得到学术界和工业界的广泛关注,并取得了大量研究成果.本文对已有的研究进行总结,并提出了基于程序频谱的动态缺陷定位研究框架,对框架内的影响因素、实证研究中采用的评测指标、评测程序以及缺陷定位在特定测试领域中的应用进行了系统的总结和分析.从本文的总结中可以看出:SFL 问题是软件调试中的一个重要研究热点,具有丰富的理论价值和应用前景.尤其令人欣喜的是,近些年来,国内不少高校和研究所(例如北京大学、南京大学、中国科学院软件所、国防科技大学、北京航空航天大学、浙江大学和东南大学等)针对该问题的研究十分活跃,在国内外权威期刊和会议上发表了大量研究成果,并得到国外同行的认可.

但就目前的研究现状分析来看,虽然有研究人员[12,13,15]借助理论分析识别出了一些最优怀疑率公式,但所得结论基于多个较强假设,很难适用于实际大规模复杂软件.同时,一些研究人员在实证研究中也发现,并不存在一种 SFL 方法,可以在所有评测程序上的定位效果均达到最优[49,50].所以,仍然需要研究人员在该领域继续展开深入研究,并提出更为有效的新颖解决方案.我们认为,该领域还存在如下值得进一步研究的问题,主要包括:

(1) 与缺陷修复进行结合

在评价缺陷定位方法有效性时,大部分研究人员均采用了完美缺陷检测假设.但该假设并不合理,即,生成的缺陷定位报告对开发人员的实际调试帮助极其有限.为了将 SFL 研究成果更好地用于指导软件的实际调试,可以从以下 3 点展开研究:首先,我们可以将缺陷定位方法与自动化的缺陷修复方法[117,118]进行结合;其次,借助

软件可视化技术,通过分析与可疑语句间存在控制或数据依赖关系的其他语句集并进行标记,可辅助开发人员对可疑语句进行判断,若该语句是缺陷语句,甚至可以辅助开发人员对该缺陷进行移除;最后,借助形式化概念分析(formal concept analysis),可以根据测试用例的程序频谱构建相应概念格,以层次结构展示语句和测试的泛化和特化关系,从而辅助开发人员对缺陷上下文的理解.但当面向大规模软件时,构建概念格的计算开销较大,一种可行方案是采用分层思想进行构建,例如,先构建类层次的概念格,随后针对可疑的类构建方法层次的概念格,最后针对可疑方法构建相应语句层次的概念格并定位到可疑语句.

(2) 与回归测试进行结合

虽然国内外研究人员已经对该研究问题进行了探索,我们在第 5 节对已经取得的研究成果进行了总结,但回归测试下的缺陷定位方法仍可从以下两点继续展开深入研究:首先,可以进一步借助修改影响分析或程序切片研究领域取得的最新研究成果来精确识别出与代码修改存在控制或数据依赖关系的语句集,以减小代码审查的开销;其次,如何对已有的测试套件进行有效演化以辅助该场景下的缺陷定位,也同样值得关注.目前,大部分研究工作均复用修改前程序的配套测试套件,但在执行代码修改后会造成原有的部分测试用例无法适用于修改后程序,对这类测试用例需要移除或者尝试进行修复.同时,有些代码修改也会产生新的测试需求,因此,需要额外设计新的测试用例来完成对这些测试需求的覆盖.

(3) 与缺陷预测方法进行结合

缺陷预测通过分析历史开发版本,基于软件度量来构造分类模型,并用该模型来预测各个程序实体的缺陷倾向性.常用的度量指标包括代码规模、内聚性、耦合性和环路复杂度等.基于缺陷预测,可以识别出一系列缺陷倾向性高的程序实体,对这类程序实体,在设置缺陷定位方法时应设置更高的权重.

(4) 寻找新的应用场景

通过第 5 节的分析可以看出:缺陷定位方法已经初步成功应用于并发测试、Web 应用测试、数据库应用程序测试等特定领域,并取得了一定的研究成果.除了需要对这些特定测试领域的缺陷定位方法继续展开深入的研究以外,我们还需要积极寻找可以使缺陷定位方法发挥效果的新兴测试领域,例如云计算平台和物联网应用等.分析这些新领域的场景特征,并充分利用缺陷定位的已有研究成果,有助于提出高质量的解决方案.

(5) 新的软件开发范式下的缺陷定位方法研究

现有的研究大多针对传统软件开发方法下的应用程序,例如,我们通过第 4 节的分析发现,现有的评测程序一般集中于西门子套件、space 程序和 Unix 工具程序.然而,随着脚本语言的流行和软件开发方法的革新,出现了诸如动态代码、基于 Web 服务的 SOA 应用、基于模型的软件开发等崭新形态的软件开发范式.我们有必要开展针对此类软件展开缺陷定位研究,并搜集相关评测程序.

(6) 将已有研究成果与企业的实际软件开发流程进行紧密结合

目前,绝大部分研究均采用对照实验(controlled experiment)方式进行有效性评估,采用的评测程序规模较小,其一般都集中于少数典型程序,所以得到的结论不具有一般性,难以适用于企业内部的大型软件产品.同时,所开发出的缺陷定位工具并未充分与软件企业目前的开发、测试和调试流程紧密结合,其自动化程度、用户界面的友好性、与 IDE 工具的集成以及工具的鲁棒性有待改进.

若能对上述研究问题提出有效的解决方案,则将进一步推动该领域的研究进展,并大幅度提高企业内的测试和调试效率,从而缓解项目测试预算有限和软件产品高质量需求间的矛盾.

致谢 感谢各位审稿专家提出的宝贵意见.

References:

[1] Yu K, Lin MX. Advances in automatic fault localization techniques. Chinese Journal of Computers, 2011,34(8):1411−1423 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01411]

[2] Steimann F, Frenkel M, Abreu R. Threats to the validity and value of empirical assessments of the accuracy of coverage-based fault locators. In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2013. 314−324. [doi: 10.1145/2483760.2483767]

[3] Renieris M, Reiss SP. Fault localization with nearest neighbor queries. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2003. 30−39. [doi: 10.1109/ASE.2003.1240292]

[4] Reps T, Ball T, Das M, Larus J. The use of program profiling for software maintenance with applications to the year 2000 problem. In: Proc. of the European Software Engineering Conf. on Held Jointly with the Int'l Symp. on Foundations of Software Engineering. 1997. 432−449. [doi: 10.1007/3-540-63531-9_29]

[5] Harrold MJ, Rothermel G, Sayre K, Wu R, Yi L. An empirical investigation of the relationship between spectra differences and regression faults. Software Testing, Verification and Reliability, 2000,10(3):171−194. [doi: 10.1002/1099-1689(200009)10:3<171::AID-STVR209>3.0.CO; 2-J]

[6] Wong WE, Debroy V, Gao RZ, Li YH. The DStar method for effective software fault localization. IEEE Trans. on Reliability, 2013,62(4):1−19. [doi: 10.1109/TR.2013.2285319]

[7] Jones JA, Harrold MJ, Stasko J. Visualization of test information to assist fault localization. In: Proc. of the Int'l Conf. on Software Engineering. 2002. 467−477. [doi: 10.1145/581339.581397]

[8] Jones JA, Harrold MJ. Empirical evaluation of the tarantula automatic fault-localization technique. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2005. 273−282. [doi: 10.1145/1101908.1101949]

[9] Abreu R, Zoeteweij P, Gemund AJCV. An evaluation of similarity coefficients for software fault localization. In: Proc. of the Pacific Rim Int'l Symp. on Dependable Computing. 2006. 39−46. [doi: 10.1109/PRDC.2006.18]

[10] Abreu R, Zoeteweij P, Gemund AJCV. On the accuracy of spectrum-based fault localization. In: Proc. of the Testing: Academic and Industrial Conf. on Practice and Research Techniques. 2007. 89−98. [doi: 10.1109/TAIC.PART.2007.13]

[11] Wong WE, Qi Y, Zhao L, Cai KY. Effective fault localization using code coverage. In: Proc. of the Annual Int'l Computer Software and Applications Conf. 2007. 449−456. [doi: 10.1109/COMPSAC.2007.109]

[12] Naish L, Lee HJ, Ramamohanarao K. A model for spectra-based software diagnosis. ACM Trans. on Software Engineering and Methodology, 2011,20(3):1−32. [doi: 10.1145/2000791.2000795]

[13] Xie XY, Chen TY, Kuo FC, Xu BW. A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization. ACM Trans. on Software Engineering and Methodology, 2013,22(4):31:1−31:40. [doi: 10.1145/2522920.2522924]

[14] Yoo S. Evolving human competitive spectra-based fault localisation techniques. In: Proc. of the Int'l Conf. on Search Based Software Engineering. 2012. 244−258. [doi: 10.1007/978-3-642-33119-0_18]

[15] Xie XY, Kuo FC, Chen TY, Yoo S, Harman M. Provably optimal and human-competitive results in SBSE for spectrum based fault localization. In: Proc. of the Int'l Conf. on Search Based Software Engineering. 2013. 224−238. [doi: 10.1007/978-3-642-39742-4_17]

[16] Wong E, Wei TT, Qi Y, Zhao L. A crosstab-based statistical method for effective fault localization. In: Proc. of the Int'l Conf. on Software Testing, Verification, and Validation. 2008. 42−51. [doi: 10.1109/ICST.2008.65]

[17] Liblit B, Aiken A, Zheng AX, Jordan MI. Bug isolation via remote program sampling. In: Proc. of the Conf. on Programming Language Design and Implementation. 2003. 141−154. [doi: 10.1145/780822.781148]

[18] Liblit B, Naik M, Zheng AX, Aiken A, Jordan MI. Scalable statistical bug isolation. In: Proc. of the Conf. on Programming Language Design and Implementation. 2005. 15−26. [doi: 10.1145/1064978]

[19] Nainar PA, Chen T, Rosin J, Liblit B. Statistical debugging using compound boolean predicates. In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2007. 5−15. [doi: 10.1145/1273463.1273467]

[20] Chilimbi TM, Liblit B, Mehra K, Nori AV, Vaswani K. HOLMES: Effective statistical debugging via efficient path profiling. In: Proc. of the Int'l Conf. on Software Engineering. 2009. 34−44. [doi: 10.1109/ICSE.2009.5070506]

[21] Liu C, Yan X, Fei L, Han JW, Midkiff SP. SOBER: Statistical model-based bug localization. In: Proc. of the European Software Engineering Conf. on Held Jointly with Int'l Symp. on Foundations of Software Engineering. 2005. 286−295. [doi: 10.1145/1081706.1081753]

[22] Liu C, Fei L, Yan X, Han JW, Midkiff SP. Statistical debugging: A hypothesis testing-based approach. IEEE Trans. on Software Engineering, 2006,32(10):831−848. [doi: 10.1109/TSE.2006.105]

[23] Li W, Zheng Z, Hao P, Gao YC, Rao PF, Gong C. Predicate execution-sequence based fault localization algorithm. Chinese Journal of Computers, 2013,36(12):2406−2419 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2013.02406]

[24] Hao P, Zheng Z, Zhang ZY, Gao YC, Gong C, Xue YZ. Self-Adaptive fault localization algorithm based on predicate execution information analysis. Chinese Journal of Computers, 2014,37(3):500−511 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2013.00500]

[25] Masri W. Fault localization based on information flow coverage. Software Testing, Verification and Reliability, 2010,20(2):121−147. [doi: 10.1002/stvr.409]

[26] Zhang ZY, Jiang B, Chan WK, Tse TH. Debugging through evaluation sequences: A controlled experimental study. In: Proc. of the Annual Int'l Computer Software and Applications Conf. 2008. 128−135. [doi: 10.1109/COMPSAC.2008.207]

[27] Zhang ZY, Jiang B, Chan WK, Tse TH, Wang X. Fault localization through evaluation sequences. Journal of Systems and Software, 2010,83(2):174−187. [doi: 10.1016/j.jss.2009.09.041]

[28] Xu J, Zhang ZY, Chan WK, Tse TH, Li SP. A general noise-reduction framework for fault localization of Java programs. Information and Software Technology, 2013,55(5):880−896. [doi: 10.1016/j.infsof.2012.08.006]

[29] Xu J, Chan WK, Zhang ZY, Tse TH, Li SP. A dynamic fault localization technique with noise reduction for Java programs. In: Proc. of the Int'l Conf. on Quality Software. 2011. 11−20. [doi: 10.1109/QSIC.2011.32]

[30] Zhang ZY, Chan WK, Tse TH, Jiang B, Wang XM. Capturing propagation of infected program states. In: Proc. of the Joint Meeting of the European Software Engineering Conf. and the Symp. on The Foundations of Software Engineering. 2009. 43−52. [doi: 10.1145/1595696.1595705]

[31] Yu K, Lin MX, Gao Q, Zhang H, Zhang XY. Locating faults using multiple spectra-specific models. In: Proc. of the Symp. on Applied Computing. 2011. 1404−1410. [doi: 10.1145/1982185.1982490]

[32] Baah GK, Podgurski A, Harrold MJ. The probabilistic program dependence graph and its application to fault diagnosis. In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2008. 189−200. [doi: 10.1145/1390630.1390654]

[33] Baah GK, Podgurski A, Harrold MJ. Causal inference for statistical fault localization. In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2010. 73−84. [doi: 10.1145/1831708.1831717]

[34] Baah GK, Podgurski A, Harrold MJ. Mitigating the confounding effects of program dependences for effective fault localization. In: Proc. of the Joint Meeting of the European Software Engineering Conf. and the Symp. on the Foundations of Software Engineering. 2011. 146−156. [doi: 10.1145/2025113.2025136]

[35] Agrawal H, Horgan JR. Dynamic program slicing. In: Proc. of the Conf. on Programming Language Design and Implementation. 1990. 246−256. [doi: 10.1145/93542.93576]

[36] Agrawal H, Horgan JR, London S, Wong WE. Fault localization using execution slices and dataflow tests. In: Proc. of the Int'l Symp. on Software Reliability Engineering. 1995. 143−151. [doi: 10.1109/ISSRE.1995.497652]

[37] Wong WE, Qi Y. An execution slice and inter-block data dependency-based approach for fault localization. In: Proc. of the Asia-Pacific Software Engineering Conf. 2004. 366−373. [doi: 10.1109/APSEC.2004.26]

[38] Wong WE, Qi Y. Effective program debugging based on execution slices and inter-block data dependency. Journal of Systems and Software, 2006,79(7):891−903. [doi: 10.1016/j.jss.2005.06.045]

[39] Zhang XY, Gupta N, Gupta R. Pruning dynamic slices with confidence. In: Proc. of the Conf. on Programming Language Design and Implementatio. 2006. 169−180. [doi: 10.1145/1133981.1134002]

[40] Zhang XY, Tallam S, Gupta N, Gupta R. Towards locating execution omission errors. In: Proc. of the Conf. on Programming Language Design and Implementation. 2007. 415−424. [doi: 10.1145/1250734.1250782]

[41] Lei Y, Mao XG, Dai ZY, Wang CS. Effective statistical fault localization using program slices. In: Proc. of the Annual Int'l Computer Software and Applications Conf. 2012. 1−10. [doi: 10.1109/COMPSAC.2012.9]

[42] Mao XG, Lei Y, Dai ZY, Qi YH, Wang CS. Slice-Based statistical fault localization. Journal of Systems and Software, 2014,89:51−62. [doi: 10.1016/j.jss.2013.08.031]

[43] Wen WZ, Li BX, Sun XB, Li J. Program slicing spectrum-based software fault localization. In: Proc. of the Int'l Conf. on Software Engineering and Knowledge Engineering. 2011. 213−218. [doi: 10.1109/ICSE.2012.6227049]

[44] Wen WZ, Li BX, Sun XB, Liu CC. Technique of software fault localization based on hierarchical slicing spectrum. Ruan Jian Xue Bao/Journal of Software, 2013,24(5):977−992 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4342.htm [doi: 10.3724/SP.J.1001.2013.04342]

[45] Ju XL, Jiang SJ, Chen X, Wang X, Zhang Y, Cao H. Hsfal: Effective fault localization using hybrid spectrum of full slices and execution slices. Journal of Systems and Software, 2014,90:3−17. [doi: 10.1016/j.jss.2013.11.1109]

[46] Dallmeier V, Lindig C, Zeller A. Lightweight defect localization for Java. In: Proc. of the European Conf. on Object-Oriented Programming. 2005. 528−550. [doi: 10.1007/11531142_23]

[47] Tu JX, Chen L, Xu L, Lu HM, Xu BW. Fault localization of object-oriented programs with considering class feature. Chinese Journal of Computers, 2013,36(12):2420−2428 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2013.02420]

[48] Yilmaz C, Paradkar A, Williams C. Time will tell: Fault localization using time spectra. In: Proc. of the Int'l Conf. on Software Engineering. 2008. 81−90. [doi: 10.1145/1368088.1368100]

[49] Santelices R, Jones JA, Yu Y, Harrold MJ. Lightweight fault-localization using multiple coverage types. In: Proc. of the Int'l Conf. on Software Engineering. 2009. 56−66. [doi: 10.1109/ICSE.2009.5070508]

[50] Debroy V, Wong WE. A consensus-based strategy to improve the quality of fault localization. Software: Practice and Experience, 2013,43(8):989−1011. [doi: 10.1002/spe.1146]

[51] Perez A, Abreu R, Riboira A. A dynamic code coverage approach to maximize fault localization efficiency. Journal of Systems and Software, 2014,90:18−28. [doi: 10.1016/j.jss.2013.12.036]

[52] Ding H, Chen L, Qian J, Xu L, Xu BW. Fault localization method using information quantity. Ruan Jian Xue Bao/Journal of Software, 2013,24(7):1484−1494 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4294.htm [doi: 10.3724/SP.J.1001.2013.04294]

[53] Voas JM. PIE: A dynamic failure-based technique. IEEE Trans. on Software Engineering, 1992,18(8):717−727. [doi: 10.1109/32.153381]

[54] Wang XM, Cheung SC, Chan WK, Zhang ZY. Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization. In: Proc. of the Int'l Conf. on Software Engineering. 2009. 45−55. [doi: 10.1109/ICSE.2009.5070507]

[55] Masri W, Assi RA. Cleansing test suites from coincidental correctness to enhance fault-localization. In: Proc. of the Int'l Conf. on Software Testing, Verification and Validation. 2010. 165−174. [doi: 10.1109/ICST.2010.22]

[56] He T, Wang XM, Zhou XC, Li WJ, Zhang ZY, Cheung SC. A software fault localization technique based on program mutations. Chinese Journal of Computers, 2013,36(11):2236−2244 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2013.02236]

[57] Zhang ZY, Chan WK, Tse TH. Fault localization based only on failed runs. IEEE Computer, 2012,45(6):64−71. [doi: 10.1109/MC.2012.185]

[58] Hao D, Zhang L, Pan Y, Mei H, Sun JS. On similarity-awareness in testing-based fault localization. Automated Software Engineering, 2008,15(2):207−249. [doi: 10.1007/s10515-008-0025-9]

[59] Wong WE, Debroy V, Choi B. A family of code coverage-based heuristics for effective fault localization. Journal of Systems and Software, 2010,83(2):188−208. [doi: 10.1016/j.jss.2009.09.037]

[60] Tan DG, Chen L, Wang ZY, Ding H, Zhou YM, Xu BW. Spectra-Based fault localization by increasing marginal weight. Chinese Jounal of Computers, 2010,33(12):2335−2342 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2010.02335]

[61] Jiang B, Zhang ZY, Tse TH, Chen T. How well do test case prioritization techniques support statistical fault localization. In: Proc. of Annual Int'l Computers Software and Applications Conf. 2009. 99−106. [doi: 10.1109/COMPSAC.2009.23]

[62] Jiang B, Chan WK. On the integration of test adequacy, test case prioritization, and statistical fault localization. In: Proc. of the Int'l Conf. on Quality Software. 2010. 377−384. [doi: 10.1109/QSIC.2010.64]

[63] Jiang B, Zhang ZY, Chan WK, Tse TH, Chen TY. How well does test case prioritization integrate with statistical fault localization? Information and Software Technology, 2012,54(7):739−758. [doi: 10.1016/j.infsof.2012.01.006]

[64] Jiang B, Chan WK, Tse TH. On practical adequate test suites for integrated test case prioritization and fault localization. In: Proc. of the Int'l Conf. on Quality Software. 2011. 21−30. [doi: 10.1109/QSIC.2011.37]

[65] Jiang B, Zhai K, Chan WK, Tse TH, Zhang ZY. On the adoption of MC/DC and control-flow adequacy for a tight integration of program testing and statistical fault localization. Information and Software Technology, 2013,55(5):897−917. [doi: 10.1016/j.infsof.2012.10.001]

[66] Gonzalez-Sanchez A, Piel E, Gross HG, Gemund AJCV. Prioritizing tests for software fault localization. In: Proc. of the Int'l Conf. on Quality Software. 2010. 42−51. [doi: 10.1109/QSIC.2010.28]

[67] Gonzalez-Sanchez A, Piel E, Abreu R, Gross HG, Gemund AJCV. Prioritizing tests for software fault diagnosis. Software: Practice and Experience, 2011,41(10):1105−1129. [doi: 10.1002/spe.1065]

[68] Gonzalez-Sanchez A, Abreu R, Gross HG, Gemund AJCV. Prioritizing tests for fault localization through ambiguity group reduction. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2011. 83−92. [doi: 10.1109/ASE.2011.6100153]

[69] Yoo S, Harman M, Clark D. Fault localization prioritization: Comparing information theoretic and coverage based approaches. ACM Trans. on Software Engineering and Methodology, 2013, 22(3):19:1−19:29. [doi: 10.1145/2491509.2491513]

[70] Baudry B, Fleurey F, Traon YL. Improving test suites for efficient fault localization. In: Proc. of the Int'l Conf. on Software Engineering. 2006. 82−91. [doi: 10.1145/1134285.1134299]

[71] Artzi S, Dolby J, Tip F, Pistoia M. Directed test generation for effective fault localization. In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2010. 49−60. [doi: 10.1145/1831708.1831715]

[72] Campos J, Abreu R, Fraser G, d'Amorim M. Entropy-Based test generation for improved fault localization. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2013. 257−267. [doi: 10.1109/ASE.2013.6693085]

[73] Hao D, Zhang L, Zhong H, Mei H, Sun JS. Eliminating harmful redundancy for testing-based fault localization using test suite reduction: An experimental study. In: Proc. of the Int'l Conf. on Software Maintenance. 2005. 683−686. [doi: 10.1109/ICSM.2005.43]

[74] Yu YB, Jones JA, Harrold MJ. An empirical study of the effects of test-suite reduction on fault localization. In: Proc. of the Int'l Conf. on Software Engineering. 2008. 201−210. [doi: 10.1145/1368088.1368116]

[75] DiGiuseppe N, Jones JA. On the influence of multiple faults on coverage-based fault localization. In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2011. 210−220. [doi: 10.1145/2001420.2001446]

[76] Debroy V, Wong WE. Insights on fault interference for programs with multiple bugs. In: Proc. of the Int'l Symp. on Software Reliability Engineering. 2009. 165−174. [doi: 10.1109/ISSRE.2009.14]

[77] Zheng AX, Jordan MI, Liblit B, Naik M, Aiken A. Statistical debugging: Simultaneous identification of multiple bugs. In: Proc. of the Int'l Conf. on Machine Learning. 2006. 1105−1112. [doi: 10.1145/1143844.1143983]

[78] Liu C, Han JW. Failure proximity: A fault localization-based approach. In: Proc. of the Int'l Symp. on Foundations of Software Engineering. 2006. 46−56. [doi: 10.1145/1181775.1181782]

[79] Liu C, Zhang XY, Han JW. A systematic study of failure proximity. IEEE Trans. on Software Engineering, 2008,34(6):826−843. [doi: 10.1109/TSE.2008.66]

[80] Jones JA, Bowring JF, Harrold MJ. Debugging in parallel. In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2007. 16−26. [doi: 10.1145/1273463.1273468]

[81] Servant F, Jones JA. Whosefault: Automatic developer-to-fault assignment through fault localization. In: Proc. of the Int'l Conf. on Software Engineering. 2012. 36−46. [doi: 10.1109/ICSE.2012.6227208]

[82] Abreu R, Zoeteweij P, Gemund AJCV. Spectrum-Based multiple fault localization. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2009. 88−99. [doi: 10.1109/ASE.2009.25]

[83] Abreu R, Zoeteweij P, Gemund AJCV. Simultaneous debugging of software faults. Journal of Systems and Software, 2011,84(4): 573−586. [doi: 10.1016/j.jss.2010.11.915]

[84] Wen WZ, Li BX, Sun XB, Qi SS. A technique of multiple fault localization based on conditional execution slicing spectrum. Journal of Computer Research and Development, 2013,50(5):1030−1043 (in Chinese with English abstract).

[85] Xie XY, Wong WE, Chen TY, Xu BW. Spectrum-Based fault localization: Testing oracles are no longer mandatory. In: Proc. of the Int'l Conf. on Quality Software. 2011. 1−10. [doi: 10.1109/QSIC.2011.20]

[86] Xie XY, Wong WE, Chen TY, Xu BW. Metamorphic Slice: An application in spectrum-based fault localization. Information and Software Technology, 2013,55(5):866−879. [doi: 10.1016/j.infsof.2012.08.008]

[87] Lei Y, Mao XG, Chen TY. Backward-Slice-Based statistical fault localization without test oracles. In: Proc. of the Int'l Conf. on Quality Software. 2013. 212−221. [doi: 10.1109/QSIC.2013.45]

[88] Gong L, Lo D, Jiang LX, Zhang HY. Diversity maximization speedup for fault localization. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2012. 30−39. [doi: 10.1145/2351676.2351682]

[89] Hao D, Xie T, Zhang L, Wang XY, Sun JS, Mei H. Test input reduction for result inspection to facilitate fault localization. Automated Software Engineering, 2010,17(1):5−31. [doi: 10.1007/s10515-009-0056-x]

[90] Hao D, Zhang L, Xie T, Mei H, Sun JS. Interactive fault localization using test information. Journal of Computer Science and Technology, 2009,24(5):962−974. [doi: 10.1007/s11390-009-9270-z]

[91] Gong L, Zhang HY, Jiang LX, Lo D. Interactive fault localization leveraging simple user feedback. In: Proc. of the Int'l Conf. on Software Maintenance. 2012. 67−76. [doi: 10.1109/ICSM.2012.6405255]

[92] Parnin C, Orso A. Are automated debugging techniques actually helping programmers? In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2011. 199−209. [doi: 10.1145/2001420.2001445]

[93] Le Goues C, Dewey-Vogt M, Forrest S, Weimer W. A systematic study of automated program repair: Fixing 55 out of 105 bugs for $8 each. In: Proc. of the Int'l Conf. on Software Engineering. 2012. 3−13. [doi: 10.1109/ICSE.2012.6227211]

[94] Qi YH, Mao XG, Lei Y, Wang CS. Using automated program repair for evaluating the effectiveness of fault localization techniques. In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2013. 191−201. [doi: 10.1145/2483760.2483785]

[95] Debroy V, Wong WE. Combining mutation and fault localization for automated program debugging. Journal of Systems and Software, 2014,90:45−60. [doi: 10.1016/j.jss.2013.10.042]

[96] Ali S, Andrews JH, Dhandapani T, Wang WT. Evaluating the accuracy of fault localization techniques. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2009. 76−87. [doi: 10.1109/ASE.2009.89]

[97] Do H, Elbaum S, Rothermel G. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. Empirical Software Engineering, 2005,10(4):405−435. [doi: 10.1007/s10664-005-3861-2]

[98] Hutchins M, Foster H, Goradia T, Ostrand T. Experiments of the effectiveness of dataflow- and controlflow-based test adequacy criteria. In: Proc. of the Int'l Conf. on Software Engineering. 1994. 191−200. [doi: 10.1109/ICSE.1994.296778]

[99] Abreu R, Zoeteweij P, Golsteijn R, Gemund AJCV. A practical evaluation of spectrum-based fault localization. Journal of Systems and Software, 2009,82(11):1780−1792. [doi: 10.1016/j.jss.2009.06.035]

[100] Park S, Vuduc RW, Harrold MJ. FALCON: Fault localization in concurrent programs. In: Proc. of the Int'l Conf. on Software Engineering. 2010. 245−254. [doi: 10.1145/1806799.1806838]

[101] Park S, Vuduc R, Harrold MJ. A unified approach for localizing non-deadlock concurrency bugs. In: Proc. of the Int'l Conf. on Software Testing, Verification and Validation. 2012. 51−60. [doi: 10.1109/ICST.2012.85]

[102] Artzi S, Dolby J, Tip F, Pistoia M. Practical fault localization for dynamic Web applications. In: Proc. of the Int'l Conf. on Software Engineering. 2010. 265−274. [doi: 10.1145/1806799.1806840]

[103] Artzi S, Dolby J, Tip F, Pistoia M. Fault localization for dynamic Web Applications. IEEE Trans. on Software Engineering, 2012, 38(2):314−335. [doi: 10.1109/TSE.2011.76]

[104] Yu ZX, Bai CG, Cai KY. Mutation-Oriented test data augmentation for GUI software fault localization. Information and Software Technology, 2013,55(12):2076−2098. [doi: 10.1016/j.infsof.2013.07.004]

[105] Clark SR, Cobb J, Kapfhammer GM, Jones JA, Harrold MJ. Localizing SQL faults in database applications. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2011. 213−222. [doi: 10.1109/ASE.2011.6100056]

[106] Sun CA, Zhai YM, Shang Y, Zhang ZY. BPEL debugger: An effective BPEL-specific fault localization framework. Information and Software Technology, 2013,55(12):2140−2153. [doi: 10.1016/j.infsof.2013.07.009]

[107] Azzeddine A, Mickael D, Ylies F, Lydie B. Fault localization in embedded software based on a single cyclic trace. In: Proc. of Int'l Symp. on Software Reliability Engineering. 2013. 148−157. [doi: 10.1109/ISSRE.2013.6698914]

[108] Jin W, Orso A. Bugredux: Reproducing field failures for in-house debugging. In: Proc. of the Int'l Conf. on Software Engineering. 2012. 474−484. [doi: 10.1109/ICSE.2012.6227168]

[109] Jin W, Orso A. F3: Fault localization for field failures. In: Proc. of the Int'l Symp. on Software Testing and Analysis. 2013. 213−223. [doi: 10.1145/2483760.2483763]

[110] Zeller A. Yesterday, my program worked. Today, it does not. Why? In: Proc. of the European Software Engineering Conf. on Held Jointly with the Int'l Symp. on Foundations of Software Engineering. 1999. 253−267. [doi: 10.1145/318774.318946]

[111] Zeller A, Hildebrandt R. Simplifying and isolating failure-inducing input. IEEE Trans. on Software Engineering, 2002,28(2): 183−200. [doi: 10.1109/32.988498]

[112] Yu K, Lin MX, Chen J, Zhang XY. Towards automated debugging in software evolution: evaluating delta debugging on real regression bugs from the developers' perspectives. Journal of Systems and Software, 2012,85(10):2305−2317. [doi: 10.1016/j.jss.2011.10.016]

[113] Yu K, Lin MX, Chen J, Zhang XY. Practical isolation of failure-inducing changes for debugging regression faults. In: Proc. of the Int'l Conf. on Automated Software Engineering. 2012. 20−29. [doi: 10.1145/2351676.2351681]

[114] Zhang LM, Kim M, Khurshid S. Localizing failure-inducing program edits based on spectrum information. In: Proc. of the Int'l Conf. on Software Maintenance. 2011. 23−32. [doi: 10.1109/ICSM.2011.6080769]

[115] Zhang LM, Kim M, Khurshid S. FaultTracer: A spectrum-based approach to localizing failure-inducing program edits. Journal of Software: Evolution and Process, 2013,25(12):1357−1383. [doi: 10.1002/smr.1634]

[116] Zhang LM, Zhang L, Khurshid S. Injecting mechanical faults to localize developer faults for evolving software. In: Proc. of the Int'l Conf. on Object Oriented Programming Systems Languages & Applications. 2013. 765−784. [doi: 10.1145/2509136.2509551]

[117] Jeffrey D, Feng M, Gupta N, Gupta R. Bugfix: A learning-based tool to assist developers in fixing bugs. In: Proc. of Int'l Conf. on Program Comprehension. 2009. 70−79. [doi: 10.1109/ICPC.2009.5090029]

[118] Weimer W, Nguyen T, Goues CL, Forrest S. Automatically finding patches using genetic programming. In: Proc. of the Int'l Conf. on Software Engineering. 2009. 364−374. [doi: 10.1109/ICSE.2009.5070536]

附中文参考文献:

[1] 虞凯,林梦香.自动化软件错误定位技术研究进展.计算机学报,2011,34(08):1411−1422. [doi: 10.3724/SP.J.1016.2011.01411]

[23] 李伟,郑征,郝鹏,高乙超,饶培峰,宫成.基于谓词执行序列的软件缺陷定位算法.计算机学报,2013,36(12):2406−2419. [doi: 10.3724/SP.J.1016.2013.02406]

[24] 郝鹏,郑征,张震宇,高乙超,宫成,薛云志.基于谓词执行信息分析的自适应缺陷定位算法.计算机学报,2014,37(3):500−511. [doi: 10.3724/SP.J.1016.2013.00500]

[44] 文万志,李必信,孙小兵,刘翠翠.一种基于层次切片谱的软件错误定位技术.软件学报,2013,24(5):977−992. http://www.jos.org.cn/1000-9825/4342.htm [doi: 10.3724/SP.J.1001.2013.04342]

[47] 涂径玄,陈林,许蕾,卢红敏,徐宝文.考虑类特性的面向对象错误定位.计算机学报,2013,36(12):2420−2428. [doi: 10.3724/SP.J.1016.2013.02420]

[52] 丁晖,陈林,钱巨,许蕾,徐宝文.一种基于信息量的缺陷定位方法.软件学报,2013,24(7):1484−1494. http://www.jos.org.cn/1000-9825/4294.htm [doi: 10.3724/SP.J.1001.2013.04294]

[56] 贺韬,王欣明,周晓聪,李文军,张震宇,张成志.一种基于程序变异的软件错误定位技术.计算机学报,2013,36(11):2236−2244. [doi: 10.3724/SP.J.1016.2013.02236]

[60] 谭德贵,陈林,王子元,丁晖,周毓明,徐宝文.通过增大边际权重提高基于频谱的错误定位效率.计算机学报,2010,33(12):2335−2342. [doi: 10.3724/SP.J.1016.2010.02335]

[84] 文万志,李必信,孙小兵,齐珊珊.基于条件执行切片谱的多错误定位.计算机研究与发展,2013,50(5):1030−1043.

陈翔(1980－),男,江苏南通人,博士,副教授,CCF 会员,主要研究领域为缺陷定位,回归测试,缺陷预测,组合测试.

文万志(1982－),男,博士,讲师,主要研究领域为缺陷定位.

鞠小林(1976－),男,博士生,讲师,CCF 会员,主要研究领域为缺陷定位.

顾庆(1972－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件质量保障,分布式计算.