ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2012/06(06)-0481-14
DOI: 10.3778/j.issn.1673-9418.2012.06.001

E-mail: fcst@vip.163.com
http://www.ceaj.org
Tel: +86-10-51616056

软件故障定位技术进展*

鞠小林[1,2,3+],姜淑娟[1],张艳梅[1],董国伟[2]
1. 中国矿业大学 计算机科学与技术学院,江苏 徐州 221116
2. 中国信息安全测评中心,北京 100085
3. 南通大学 计算机科学与技术学院,江苏 南通 226019

Advances in Fault Localization Techniques*

JU Xiaolin[1,2,3+], JIANG Shujuan[1], ZHANG Yanmei[1], DONG Guowei[2]
1. School of Computer Science and Technology, China University of Mining and Technology, Xuzhou, Jiangsu 221116, China
2. China Information Technology Security Evaluation Center, Beijing 100085, China
3. School of Computer Science and Technology, Nantong University, Nantong, Jiangsu 226019, China
+ Corresponding author: xiaolinju@gmail.com

JU Xiaolin, JIANG Shujuan, ZHANG Yanmei, et al. Advances in fault localization techniques. Journal of Frontiers of Computer Science and Technology, 2012, 6(6): 481-494.

Abstract: Fault localization is a kind of time consuming and labor intensive work during debugging. To reduce the cost of debugging and assist the developers to locate and repair program faults, fault localization techniques navigate the fault program fragments by examining the source code and analyzing software behavior and test results. This paper reviews the recent achievements in the field of fault localization. And then, for many representative methods of fault localization, it gives a detail analysis by categories of their basic principles and the modeling methods. It also discusses the contribution of each work and the major difference between them, and presents some commonly-used evaluation benchmarks and metrics. Finally, it concludes the future research consideration of fault localization techniques.

Key words: program analysis; software debugging; fault diagnosis; fault localization; localization metrics

* The National Natural Science Foundation of China under Grant Nos. 90818021, 60970032, 61100047 (国家自然科学基金); the Natural Science Foundation of Jiangsu Province of China under Grant No. BK2008124 (江苏省自然科学基金); the Graduate Training Innovative Projects Foundation of Jiangsu Province under Grant No. CX10B_157Z (江苏省研究生培养创新工程项目).
Received 2012-02, Accepted 2012-04.

摘 要:故障定位是调试过程中一项耗时费力的工作。为了降低调试成本,并辅助开发人员定位和修复软件故障,软件故障定位技术通过审查源代码、分析测试过程的软件行为和测试结果来定位包含故障的代码片段。综述了近期故障定位领域相关成就,分类介绍了各种代表性的故障定位方法的基本原理和建模技术,讨论了这些故障定位技术的贡献以及它们之间的主要区别,给出了常用的故障定位效果基准测试集和度量方法,展望了故障定位技术的研究方向。

关键词:程序分析;软件调试;故障诊断;故障定位;定位度量

文献标识码:A **中图分类号:**TP311

1 引言

软件调试需要程序员进行大量的人机交互[1]。故障定位是调试过程中最为耗时和费力的活动之一[2],它通过审查源程序语义和结构,结合分析程序的执行过程及结果,辅助开发人员找到软件故障位置。研究人员提出了一系列自动化的故障定位方法,这些方法可分为静态方法和动态方法。静态方法利用程序的依赖关系、类型约束等信息来分析程序中的可能故障点;动态方法则通过测试程序,跟踪程序的执行轨迹和覆盖信息来进行故障定位。高效地定位软件故障可减轻程序员手工排查程序语句的工作量,提升调试速度和效率。

本文调研了近30年来出现的具有代表性的软件故障定位方法,分类并说明了其原理及模型,讨论了这些方法的主要成果并对它们进行了比较,给出了常用的评测集和故障定位评价方法,最后展望了软件故障定位的进一步研究方向。

2 故障定位技术概述

2.1 问题的提出

根据IEEE标准定义,故障是隐藏在程序中不正确的指令、过程或数据定义[3]。故障是由程序员的失误引入的,一次失误可能导致一个或多个软件故障,这些故障隐藏于程序中的一条语句或分散的若干语句之中。故障定位的粒度,可以是程序语句,也可以是程序的分支、函数、方法或类等基本块。语句和基本块统称为程序实体。

定义程序P由m个程序实体组成,记为$P=\{s_1, s_2, \ldots, s_m\}$[4]。定义$S'=\{s'_1, s'_2, \ldots, s'_k\}$为程序$P$存在的故障,其中$S'\subseteq P$,$s'_j(1\leqslant j\leqslant k)$为存在故障的程序实体。故障定位是找出$P$中所有的存在故障程序实体集合$S'$的过程。

不妨假设P中的程序实体都有可能出错,从而将精确定位故障转化为计算所有程序实体出错的可能性,可得到一个程序实体的故障怀疑度降序列表$Rank=[s'_1, s'_2, \ldots, s'_m]$,$s'_j\in P(1\leqslant j\leqslant m)$。简言之,故障定位的关键任务就是求得程序实体怀疑度排名列表,程序员调试时依次检查排名列表中的程序实体,直至找到真正软件故障。

2.2 相关研究

人们提出了一系列用于调试的自动故障定位方法[5-21]。Weiser[5]最早提出了可将程序切片用于软件调试中。Al-Khanjari等人[6-10]进一步研究了在故障定位中应用切片技术。Renieris和Reiss[11]比较了程序成功和失败执行的相似执行序列的差异,提出了基于相似程序频谱的最近邻查询(nearest neighbor queries,NNQ)方法。Jones等人[12]认为语句的故障怀疑度与其执行失败的次数正相关,提出了用不同颜色表示语句故障可疑程度的方法。Han等人[13]对谓词在成功和失败执行时的取值模式进行建模,量化每个谓词取值模式与故障之间统计相关性,提出了基于假设检验的SOBER方法。Cellier[14]将形式概念分析用于故障定位。Ali等人[15]应用数据挖掘中分类技术进行故障定位。国内学者在故障定位领域也进行了相关研究,取得了一些研究成果。虞凯等人[16]对软件错误的动态定位技术进行了比较全面的总结,并提出了一种基于多频谱的定位技术[17];王新平等人提出了基于程序执行轨迹的故障定位方法[18];徐宝文等人[19]提出了基于组合测试的调试方法;严俊等人[20]也对基于组合测试的故障定位技术进行了总结;郝丹

等人[21]考虑了测试用例之间的相似性对故障定位的影响，提出了基于相知相似度故障定位（similarity-aware fault localization，SAFL）方法。

3 软件故障定位技术的分类

依据定位过程“是否需要运行软件”准则，将故障定位技术分为两大类：

（1）基于静态分析的故障定位（static analysis-based fault localization，SABFL）。依据程序语言的语法和语义，静态分析软件结构和程序实体之间依赖关系以发现违反系统约束的程序实体，从而定位故障。

（2）基于测试的故障定位（test-based fault localization，TBFL）。设计测试用例并运行软件，依据程序执行轨迹及输出结果进行故障定位。

3.1 基于静态分析的故障定位

静态分析指不运行程序而分析程序源码或目标码。目前主要有四种基于静态分析的故障定位方法：面向语句的方法、符号执行方法、形式化方法和指针分析方法。

3.1.1 面向语句的方法

面向语句的方法依据程序设计语言的基本约束，检测程序的语法、控制结构以及数据类型等问题，定位故障并提出警告或给出修复建议。有两种实现方式：一种是将语法分析、类型分析等分析技术集成到编译器或集成开发环境[22]，随程序编辑过程进行静态分析以识别故障；另一种方式通过独立分析工具或开发环境的外部插件，静态分析程序语句以定位故障。

FindBugs[23]是一个开源的用于静态分析Java源程序及字节码的框架，采用不同技术设计故障或缺陷检测器，并定义相应的故障或缺陷模式，在程序编辑和调试时检测并定位Java源程序的故障，同时给出修复建议。FindBugs预定义了300多种故障和缺陷模式，例如文件未关闭、缺少必要的或多余的null check等。赵建军等人[24]在FindBugs基础上，定义了用于检测面向方面（aspect-oriented）系统的17种故障模式，并设计了用于检测AspectJ故障的XFindBugs系统。

面向语句的方法一般用于编辑或调试程序时定位与程序语法结构、变量类型以及系统约束有关的故障，但不能发现与程序逻辑有关的故障，如不可达路径、程序死锁等。

3.1.2 符号执行方法

符号执行[25]指不执行程序，用符号（如x）作为变量的值，对程序各条路径模拟执行，跟踪被模拟的各条执行路径上变量的值（实际上是符号x的表达式），从而得到一组关于变量初始值的约束，称为路径条件。只有当路径条件满足时，对应路径才是可执行的。需用约束求解方法来判定路径条件是否满足，并需要考虑两种情形：（1）如果得到的路径条件是布尔表达式，则该约束求解问题是NP问题；（2）如果得到的路径条件是一组数值表达式，则可用线性规划求解。把分析工作限定在实际可达路径上，能为程序员提供更为准确的故障相关的上下文信息。

King[25]最早在调试中使用符号执行方法，实验表明符号执行可以较好地适用于顺序程序的调试。Young等人[26]提出了结合符号执行的并发分析技术，用于检测并发程序中的故障。他们首先基于Taylor算法[26]生成一组程序流图，并为每个流图分配一个Token（代表控制线程），然后用路径表达式表示符号执行的值，路径条件表示符号执行条件，自上而下符号执行程序流图，检查访问冲突、死锁等故障。

符号执行避免了构造测试用例和运行程序，但是也面临两个主要难题：

（1）复杂的结构语义和操作语义建模问题。符号执行要对被分析代码的结构语义和操作语义进行建模，构建一个虚拟机模型。符号执行是路径敏感的分析方法，因此一般为每条路径创建一个专属虚拟机模型，以保证路径之间相互独立。编程语言中使用的数据结构和操作语句具有复杂性和灵活性，使得建模变得十分困难，而模型准确程度将直接影响静态分析结果精度。

（2）路径爆炸问题。程序可执行路径随程序规模的增大呈指数增长，分析工作量随之急剧增加[27]。

3.1.3 形式化方法

形式化方法采用数学及逻辑方法描述和验证软

件系统。描述内容包括系统性质和行为。系统性质指系统必须满足的约束,系统行为指为实现系统功能所执行的动作。描述系统性质称为规约,描述系统行为称为建模。形式化验证主要包括两类方法:一类是以逻辑推理为基础的定理证明,将软件验证问题转化为数学定理证明问题,判断程序是否满足指定属性[28];另一类是以穷尽搜索为基础的模型检验[29],通过显式状态搜索或隐式不动点计算来验证有穷状态并发/实时系统的模态/命题性质。

Flanagan等人[30]利用条件验证和自动定理证明技术设计了静态检查Java代码的检查器(ESC/Java),可在编译Java代码时检测常见的源代码故障。在程序中添加简单的标注语言,形式化表示程序决策,ESC/Java检查这些带标注语言的源代码,对标注语言表示的决策与实际程序代码不一致部分给出警告信息,或者给出潜在运行时故障警告。

定理证明和模型检验可用于检测软件死锁等故障,但由于需要先建立软件的形式化模型,增加了额外的工作量。此外它们与符号执行一样,面临着状态空间爆炸问题,因此在实际软件故障定位中应用较少。

3.1.4 指针分析方法

指针分析[31]用于确定指针值以及所指对象的存储位置。程序运行遇到空指针解引用(null pointer dereference,NPR)时会引发异常。NPR问题是指针分析研究的重点内容。研究者针对NPR问题开展了多方面研究,提出的NPR分析方法可以分为两类:

第一类方法是自动推导出空指针标注。Cousot等人最先提出指针分析[32],定义了一个抽象域概念来描述程序空值变量,通过抽象解释形式证明了抽象域描述的正确性[33]。Hubert等人[34]提出了一种空指针分析方法,通过扩展抽象域推导出非空字段,基于约束分析以识别NPR。抽象域只能描述一般指针变量空值情形,不能处理对象指针空值情形。

第二类方法需要预先给程序代码段添加空指针标注。Male等人[35]对类的检查算法进行了扩展,通过检查类和过程内传递的标注进行指针分析。这种方法将指针分析的范围局限在类或过程内部,借助指针分析工具进行全局分析,可以获得较高的局部分析精度,但得到的结果可能不正确或不完整。Spoto[36]提出了一种用抽象解释构造和证明上下文敏感的静态空指针分析方法,该方法构造二叉决策图,对Java字节码和Java代码进行基于布尔表达式的非空标注推断。

指针分析往往借助符号执行或实际执行程序,而且指针问题常与内存泄露、访问越界等相关。受可判定性问题的制约,加上分析时间和存储空间等限制,多数的指针分析采用"近似"或者"简化"策略,导致分析结果不够精确[27]。

静态分析可以提早发现和定位程序故障,减少后期维护成本。此外,静态分析理论知识完备,已开发了一些商用或开源静态分析工具(如FindBugs、JLINT、PMD、ESC/Java等)。静态分析能发现动态分析难以发现的故障,但静态分析技术也具有局限性,需要改进和完善。

3.2 基于测试的故障定位

基于测试的故障定位通过选择测试用例,运行待测程序,跟踪并分析程序执行过程及结果,从而定位软件故障。根据定位故障利用的信息及定位方式,基于测试的故障定位技术可以分为以下三类:基于执行覆盖的故障定位(coverage-based fault localization,CBFL)、基于依赖关系的故障定位(dependency-based fault localization,DBFL)、基于模型的故障定位(model-based fault localization,MBFL)。

3.2.1 基于执行覆盖的故障定位

CBFL跟踪程序执行轨迹及执行结果,计算程序实体的故障怀疑度。程序执行轨迹可以通过程序的输出、插装代码或者平台接口获取[27]。根据计算怀疑度的方法和策略,将故障定位方法分为以下三类:集合运算法、概率统计法和怀疑度计算优化。

(1)集合运算法

集合运算法认为出现在失败执行轨迹中的程序实体可能存在故障,采用集合的并和交运算缩小怀疑范围[11,39]。一次失败运行轨迹为Tr_{f},一组成功运行轨迹为$Tr_{\mathrm{p}}^{1}, Tr_{\mathrm{p}}^{2}, \ldots, Tr_{\mathrm{p}}^{k}$,并和交运算的结果分别为$Tr_{\mathrm{f}}-\bigcup_{i=1}^{k} Tr_{\mathrm{p}}^{i}$和$\bigcap_{i=1}^{k} Tr_{\mathrm{p}}^{i}-Tr_{\mathrm{f}}$。通常并集比交集得

到的可疑故障集规模大，程序员需要检查更多的程序实体，但故障查全率比后者高。

Renieris 和 Reiss[11]提出了最近邻执行轨迹 NNQ 故障定位方法。对于一个失败的执行轨迹 Tr_f 和许多成功的执行轨迹 $Tr_p^1, Tr_p^2, \ldots, Tr_p^k$，根据距离准则从 $Tr_p^1, Tr_p^2, \ldots, Tr_p^k$ 中选择一条与 Tr_f 距离最近的一条 Tr_p^m，可疑故障集用 $Tr_f - Tr_p^m$ 表示。计算轨迹距离可用海明距离、夹角余弦或者排列距离。其中，海明距离只关注程序实体是否被执行，不能反映两个执行轨迹间真正的相似性。夹角余弦距离直接用各程序实体被执行的次数参与计算，受循环执行的影响较大。排列距离只考虑执行次数的相对关系，能较好地体现执行轨迹间的相似性[25]。NNQ 法分离出一部分程序实体作为可疑故障集合。

（2）概率统计法

概率统计法认为程序中每一个可执行实体都可能存在故障，但可疑程度不同，可以根据概率统计原理，计算每个程序实体存在故障的概率。

Jones 等人[12]认为程序语句存在故障的可能性与它被失败用例执行的次数正相关，提出了用红-黄-绿等颜色可视化表示语句存在故障的可疑程度，红色表示高度可疑，绿色表示不怀疑。并且给出了颜色计算公式，实现了可视化故障定位工具 Tarantula。Jones 等人[37]进一步给出了语句 s 的怀疑度计算公式如下：

$$Tarantula(s)=\frac{\frac{T_f(s)}{|T_f|}}{\frac{T_p(s)}{|T_p|}+\frac{T_f(s)}{|T_f|}} \tag{1}$$

其中，$T_f(s)$ 和 $T_p(s)$ 分别表示语句 s 失败执行和成功执行的次数；$|T_p|$ 和 $|T_f|$ 分别表示测试通过和未通过的测试用例数。Tarantula 方法需要一个较大规模的测试集，并且至少需要一次通过测试和一次未通过测试。怀疑度的取值范围从 0 到 1，数值越大，表示语句出错的概率越大。计算每个可执行语句的怀疑度，调试人员依怀疑度降序检查语句，直至找到真正故障。Tarantula 方法容易推广到基本块、函数、方法等程序实体。此外，Tarantula 能一定程度地容忍故障语句偶尔出现在成功的执行覆盖中，而且这种容忍有时能提升 Tarantula 的故障定位有效性[37]。

Tarantula 方法提出之后，研究者陆续提出了一些类似的怀疑度计算方法[17-18]，但故障定位精度并没有得到明显提高。Abreu 等人[38]引入分子生物学领域的相似系数（similarity coefficients）Ochiai，定义语句 s 的怀疑度计算公式如下：

$$Ochiai(s)=\frac{T_f(s)}{\sqrt{|T_f|\times(T_p(s)+T_f(s))}} \tag{2}$$

Abreu 等人[38]也通过实验证明了 Ochiai 的定位精度稍优于 Tarantula 方法，并且在定位多故障时与 Tarantula 方法相比，定位性能得到了提升[39]。

Tarantula 方法与 Ochiai 方法只考虑语句在成功执行的轨迹和失败执行的轨迹中的频率，没有考虑语句与执行结果之间的关系。Wong 等人[40-41]提出了一个定义良好的统计方法 Crosstab。该方法分别统计成功执行与失败执行情况下，可执行语句 s 分别被覆盖的次数，并构建该语句的交叉表，进而对 s 计算 χ^2 统计量 $\chi^2(s)$。$\chi^2(s)$ 表示语句 s 与测试结果关联程度，在给定显著性水平 χ_σ^2 下进行假设检验，可以判断语句 s 是否独立于测试结果。$\chi^2(s)$ 随着样本数量增多而上升，为方便度量怀疑度，转而求语句 s 的相依系数 $M(s)=\frac{\chi^2(s)\big/(|T_p|+|T_f|)}{\sqrt{(row-1)\times(col-1)}}$，语句 s 的怀疑度定义如式（3）所示：

$$Crosstab(s)=\begin{cases}M(s), & \text{if } \varphi(s)>1\\ 0, & \text{if } \varphi(s)=1\\ -M(s), & \text{if } \varphi(s)<1\end{cases} \tag{3}$$

其中，$\varphi(s)=(T_f(s)/|T_f|)/(T_p(s)/|T_p|)$。当 $\varphi(s)=1$ 时，则 $\chi^2(s)=0$，语句 s 对软件成功执行和失败执行的贡献相同；当 $\varphi(s)>1$ 时，则语句 s 对失败执行的贡献较大，语句 s 可能存在故障，怀疑度为 $M(s)$；反之，当 $\varphi(s)<1$ 时，则 s 语句对成功执行的贡献较大，怀疑度为 $-M(s)$。

与上述基于语句的执行轨迹定位方法不同，文献[13,42]提出了基于谓词覆盖的故障定位技术。该技术考虑到程序中谓词取值（真/假）与其后某个故障的出现有着必然联系。例如，一段程序的谓词 P 的真

分支上存在故障F,假分支语句正常,那么测试该程序时,只有谓词P取真时,程序执行失败,故障F才会暴露出来。Liblit等人[42]提出了一种互操作分离故障技术,称为Liblit05。在程序分支、函数返回及标量位置进行插装,执行测试用例,捕获谓词的覆盖信息。谓词P的怀疑度计算公式如下:

$$Liblit05(P)=\frac{2}{\dfrac{1}{\dfrac{F(P_{\mathrm{T}})}{S(P_{\mathrm{T}})+F(P_{\mathrm{T}})}-\dfrac{F(P_{\mathrm{O}})}{S(P_{\mathrm{O}})+F(P_{\mathrm{O}})}}+\dfrac{1}{\log(F(P_{\mathrm{O}}))}}\quad(4)$$

$$\log(|T_{\mathrm{f}}|)$$

其中,P_{T}表示P取值为真;$S(P_{\mathrm{T}})$和$F(P_{\mathrm{T}})$分别表示P取值为真时程序成功执行和失败执行的次数;$S(P_{\mathrm{O}})$和$F(P_{\mathrm{O}})$分别表示P在成功和失败执行中被观察(覆盖)到的次数。

(3)怀疑度计算优化

集合运算和概率统计方法依据执行覆盖简单地计算语句或谓词的怀疑度,没考虑其他影响因素,如测试用例的选择问题、执行轨迹存在的"噪音"问题、方法或函数的执行顺序等。一些研究者对这些问题展开讨论,并给出了更为精确的计算怀疑度的方法。

上述基于测试的故障定位方法忽略了测试用例之间的相似性,郝丹等人[21]指出应该消除相似测试用例对故障定位精度的影响,提出了相知相似度故障定位SAFL方法。该方法将测试用例集视为一个模糊集,基于概率理论计算语句隶属于失败测试用例和所有测试用例的隶属度,用两者的比值作为该语句的怀疑度。具体计算步骤如下:

①语句覆盖信息和测试结果用执行矩阵$\boldsymbol{E}=(e_{ij})$来表示。其中,e_{ij}表示测试用例t_i时,语句s_j的执行信息,被执行到为1,否则为0;$e_{i(m+1)}$表示测试用例t_i时,程序P的执行结果,测试通过为1,否则为0。

②计算模糊矩阵$\boldsymbol{F}=(f_{ij})$。其中,f_{ij}表示语句s_j对测试用例t_i状态的贡献度。

$$\boldsymbol{F}=(f_{ij})=\begin{cases}\dfrac{1}{\sum_{k=1}^{m}e_{ik}}, & \text{if } e_{ij}=1\\ 0, & \text{otherwise}\end{cases}\quad(5)$$

③用语句s_j在失败执行和所有执行中的贡献度的比值表示其故障怀疑度,比值增大,语句s_j出错的可能性就随之上升。计算公式如下:

$$SAFL(s_j)=\frac{\sum_{k=1}^{m}\max(f_{ik}|e_{ij}>0\wedge e_{i(m+1)}=0)}{\sum_{k=1}^{m}\max(f_{ik}|e_{ij}>0)}\quad(6)$$

SAFL方法计算怀疑度的时间复杂度为$O(m\times n)$,m为语句数,n为测试用例数量。

Naish等人[43]提出了一段C语言表示的单故障程序片段ITE2(If-Then-Else-2),描述故障发生的两种情形:①故障与程序逻辑和测试数据强相关,只要执行逻辑或测试数据满足一定条件,故障就必然产生;②只是偶尔满足故障发生的条件,一般很难检测到故障行为。基于ITE2模型的分析,Naish等人提出了两种理想度量怀疑度公式:

$$O(s)=\begin{cases}-1, & |T_{\mathrm{f}}|>T_{\mathrm{f}}(s)\\ |T_{\mathrm{p}}|-T_{\mathrm{p}}(s), & \text{otherwise}\end{cases}\quad(7)$$

$$O^{\mathrm{p}}(s)=T_{\mathrm{f}}(s)-\frac{T_{\mathrm{p}}(s)}{|T_{\mathrm{p}}|+1}\quad(8)$$

与上述方法不同,Dallmeier等人[44]提出了一种面向对象的分析方法——模式的故障定位方法(analyzing methods patterns to locate errors,Ample),跟踪待测软件的一次失败执行和多次成功执行,记录每次执行的类方法调用序列,比较成功执行和失败执行的序列,对出现在失败执行序列和所有成功执行序列的方法集合赋予最低怀疑度(如0),对只出现在失败执行序列或所有成功执行序列中的方法集合赋予最高怀疑度(如1),其余的方法子序列将其在成功或失败执行序列中出现的次数与成功执行次数的比值作为怀疑度。一个类的怀疑度则可以用类属方法子序列怀疑度的加权平均值表示,计算公式如下:

$$Ample(C)=\frac{1}{|r|}\sum_{s\in r}w(s)$$

$$r=tr_{\mathrm{f}}\cup tr_{\mathrm{p}}^{1}\cup tr_{\mathrm{p}}^{2}\cup\ldots\cup tr_{\mathrm{p}}^{n}\quad(9)$$

其中,r表示程序的运行集合;$tr_{\mathrm{p}}^{i}(1<i<n)$和$tr_{\mathrm{f}}$分别表示程序执行失败和执行成功时的方法调用序列;$s$为类$C$的方法子序列;$w(s)$是类$C$的方法子序列的权重。Ample方法可以用于面向对象程序类级故障定位。

研究表明，没有一种单一的覆盖类型能很好地适应于所有故障类型，不同覆盖类型只能适用于不同类型的故障定位。Santelices等人[45]提出了一种使用多重覆盖的轻量级故障定位组合策略。首先用Tarantula方法计算语句、分支和定值-使用对的怀疑度，将语句按其怀疑度排序，然后按三条规则将分支（定值-使用对）关联到相应的条件语句（定值语句），并按照max-SBD、avg-SBD和avg-BD策略得到每个语句的怀疑度。max-SBD、avg-SBD分别为三类程序实体的最大怀疑度和平均怀疑度，avg-BD为与该语句关联的分支和定值-使用对的怀疑度平均值。该方法只是组合计算了语句的怀疑度值，本质上与Tarantula方法相同。实验表明[45]，avg-SBD和avg-BD比单一覆盖技术有效，且具有统计显著性优势，但通常情况下单一覆盖比max-SBD更有效。

3.2.2 基于依赖关系的故障定位

控制依赖和数据依赖是程序实体之间的基本依赖关系，可以利用这些依赖关系辅助故障定位。事实上，Santelices等人[45]提出的基于组合覆盖故障定位方法已经部分考虑了控制依赖和数据依赖关系对故障定位的影响。Baah等人[46]在程序依赖图的基础上，通过增加节点和边建立概率依赖图（probabilistic program dependence graph，PPDG），根据程序一次失败执行的节点状态信息计算PPDG上节点的条件概率，并提出一种RankCP算法求得所有节点的条件概率升序列表。条件概率越小，节点的故障怀疑度越大。

利用程序切片进行故障定位是另一种基于依赖关系的定位方法。程序切片是一组数据流相关的语句集合[5]，表明了程序语句之间数据依赖和控制依赖关系，其规模一般小于原始程序，从而方便调试人员理解程序，已被广泛应用于程序分析与调试[47]。若程序某个变量x的值出现错误，则故障可以定位在变量x的切片上[48]。Lyle和Weiser[49]提出程序切块（dice）方法。例如，变量x和y，若x值计算正确，而y值计算错误，则故障可能存在于y的切片除去x切片后的语句中。Agrawal在其博士论文中阐述了将切片用交集、并集和补集运算，得到可疑的故障语句集合的方法[50]。

基于依赖关系的故障定位是一类重量级的故障定位方法，需要分析程序实体之间的关系，构建PPDG或计算程序切片等，与基于覆盖的定位方法相比，需要更多的时间和空间开销。

3.2.3 基于模型的故障定位

基于模型的故障定位先给出一个程序或程序执行的行为模型，基于模型定位故障。主要的模型有统计模型、程序状态模型和时间频谱模型等。

（1）统计模型。Han等人提出基于统计模型SOBER故障定位方法[13,51]。在测试中，一个谓词P取真值的次数为P_t，取假值的次数为P_f，谓词P的取值模式为$\omega(P)=P_t/(P_t+P_f)$。设所有成功执行时P的取值模式为$\omega(P|\theta_p)$，失败执行时P的取值模式为$\omega(P|\theta_f)$，Han等人认为若$\omega(P|\theta_p)\neq\omega(P|\theta_f)$，则$P$是与故障相关的，而且$\omega(P|\theta_p)$与$\omega(P|\theta_f)$差异越大，$P$是故障的可能性越大。谓词$P$的怀疑度计算公式如下：

$$SOBER(P)=\log_{\alpha}\left(\frac{\sigma_P}{k\times e^{-\frac{Z^2}{2}}}\right) \tag{10}$$

其中，底数$\alpha=|R|\neq 0,1$，建议取值为e或10；方差$\sigma_P^2=\frac{1}{n-1}\sum_{i=1}^{n}(x_i-\mu_p)^2$；期望$\mu_p=\frac{1}{n}\sum_{i=1}^{n}x_i$；$x_i$是谓词$P$第$i$次成功执行时取真值的概率；系数$k=\sqrt{\frac{|T_f|}{2\pi}}$，$Z=\frac{Y-\mu_p}{\sigma_P/\sqrt{m}}$，$Y=\frac{1}{m}\sum_{i=1}^{m}y_i$，$y_i$是谓词$P$第$i$次失败执行时取真值的概率。

虽然SOBER和Liblit05都是基于谓词的故障定位方法，但Liblit05只考虑谓词取真值情形，如果某个谓词在所有执行中都有一次取值为真，那么Liblit05技术就失去区分该谓词的能力。Liblit05与Tarantula本质上一致，都是基于覆盖次数计算怀疑度，而SOBER利用假设检验建模来量化谓词P的取值偏差以作为谓词怀疑度。然而谓词之间可能存在相互联系，即前面一个谓词赋值影响到后面谓词取值，称为谓词相关性。一旦谓词之间存在相关性，后面谓词便成为干扰项，影响真正含故障谓词的检出。SOBER模型未考虑这种相关性，默认谓词之间是相互独立的。

(2)程序状态模型。程序状态由程序变量、系统参数、内部事件、用户输入等组成,随程序指令执行而变化。可根据测试时成功执行和失败执行的程序状态变化定位故障。

徐宝文等人[19]用值向量表示待测系统和测试用例,将系统测试过程视为值向量的变化过程,提出了一种根据测试结果和附加测试用例的测试结果进行故障定位的方法。该方法假定软件系统故障仅由系统的某些取值模式引起,并且这些模式的任意父模式也会引发同样故障。为了进一步确定出哪个取值模式导致了软件故障,需要为每个发生故障的测试用例设计多个附加测试用例。运行这组测试用例,根据结果就可以确定出很小范围的导致故障的因素。但是,实际系统有可能出现与模型假设相反的情况,例如某个模式引发一个系统故障,但它的某些父模式不会引发这个故障。

Cleve和Zeller[52]将Delta Debugging应用到程序状态上,基本思想是比较程序执行成功和失败时状态空间上那些在时间和空间上与故障有关的变量及其变量值的"差异"。首先由程序的变量构成程序状态图;然后使用公共子图识别成功和失败执行程序状态的"差异";最后在程序状态图中定位"差异",找到导致故障的变量,跟踪变量的值,直至找到产生错误的语句。

(3)时间频谱模型。时间频谱跟踪记录程序实体(如方法、函数)一次执行耗费的时间。如果一个程序实体在失败执行中耗费时间显著多于(或少于)其在成功执行中耗费的时间,则认为该程序实体中可能存在故障,且可疑程度与时间偏移程度正相关。

Yilmaz等人[53]提出TWT(time will tell)方法,将时间频谱用于查找函数中的故障。TWT方法用高斯混合模型描述程序行为,用成功执行的时间频谱进行聚类,然后对每个类使用高斯分布建模。故障定位时,先将失败执行的时间频谱作为所建高斯混合模型的输入,得到的输出值表示失败执行与成功执行在该方法调用处的偏移程度,用该偏移值作为类中该方法的怀疑度。TWT方法定位的粒度是程序中的方法,最终得到所有方法的怀疑度排名列表。

4 故障定位的有效性和效率度量

4.1 评测数据集

研究故障定位时,需要用到一些主题程序以度量故障定位的有效性,这些主题程序以及相关的测试用例、故障信息称为评测数据集。评测数据集可从SourceForge(http://sourceforge.net)、SIR(http://sir.unl.edu)等开源网站获得。

SIR在软件测试领域影响范围非常广,全世界超过380个高校和科研机构使用SIR的评测数据集开展研究,超过210篇论文使用了SIR的程序和数据集。SIR提供了典型的程序、精心设计的测试用例和人工构建的不同故障版本,测试数据齐全。评测数据集由Java、C和C#程序组成,其中部分程序含有原发故障(real fault),大多数程序是通过人工注入故障或变异方式以构建不同故障版本,每个版本只含一个故障,研究人员还设计了相应的测试用例[54]。目前,SIR提供46组Java程序,包括ant、jmeter等;15组C程序,包括Siemens Suite、Space和一些Unix应用程序;C#程序只有一名为tcas-csharp的小程序。这些源程序、对应错误版本程序以及测试用例等文档均可以在SIR网站获得。

SourceForge提供了更多程序设计语言的开源程序,但这类程序往往需要研究人员自己设计测试用例。郝丹等人[21]从SourceForge下载了3个C语言源程序进行故障定位比较实验。Artzi等人[55-56]使用了SourceForge网站上4个不同类型的开源程序用于研究程序类型对故障定位效果的影响。Ali等人[15]用开源C++程序Concordance开展故障定位有效性研究。开源中国OSChina(http://www.oschina.net)也有大量可用的开源程序。

4.2 有效性度量

4.2.1 查全率和查准率

故障定位有效性可以用信息检索领域的查准率(precision)和查全率(recall)来度量,计算方法如下:

$$Precision=\frac{tp}{tp+fp} \tag{11}$$

$$Recall=\frac{tp}{tp+fn} \tag{12}$$

查全率是相关信息/系统中的相关信息
查准率是相关信息/检索出的相关信息

其中，tp（true positive）为正确报告故障数；fp（false positive）为误报故障数；tn（true negative）为报告的正确语句数；fn（false negative）为漏报正确语句数。也可以用查准率和查全率的调和平均数F（F-measure）度量故障定位技术的有效性。

$$F=2\times\frac{Precision\times Recall}{Precision+Recall} \tag{13}$$

一般只检查故障定位技术得到的怀疑度降序列表中前面k%语句。显然增加k的值，可以检出更多的故障，提升了查全率，但这样需要检查更多的语句，查准率反而下降了。因此在实际故障定位时，要平衡查全率和查准率，在调试过程中精心选择k值，一般建议选择一组k值[57]，k=1，5，10，15。

上述查全率和查准率的定义假设程序员每次都要检查完怀疑度列表的前面k%的语句。事实上，如果程序员调试过程中发现列表中第i个语句存在故障$(1\leqslant i\leqslant k)$，则不需要检查余下的语句。因此，查全率和查准率度量故障定位的有效性并不精确，计算所得值比实际值低。

4.2.2 需检查代码的比例

故障定位目的是使程序员只需检查少量代码就能找到故障，显然需要检查的代码占全部代码的百分比越小，定位效率越高。

Renieris和Reiss[11]提出了基于程序依赖图的度量方法。首先定义程序依赖图（program dependence graph，PDG）中节点的k阶依赖集（DS_k），表示在PDG中与该节点距离为k的所有节点集合。一个故障定位报告R可用$DS_*(R)$表示可能包含故障的最小节点集合，程序员需要检查这些节点以查找故障，用$T=1-|DS_*(R)|/|PDG|$表示故障定位报告的有效性。其中，$|DS_*(R)|$表示$DS_*(R)$集合中的节点个数；$|PDG|$表示PDG中节点个数。Renieres和Reiss的方法也被称为T-Score，该方法存在与查全率、查准率评测标准类似的不足，程序员不需要检查$DS_*(R)$中所有的节点。此外，对于故障集合$DS_*(R)$中的故障，有些故障容易识别，而有些故障则不容易识别，总体上T-Score低于实际值。

Jones和Harrold[37]提出了一种更为精确和直观的基于语句度量方法。该方法计算故障定位报告中真实故障语句之后的语句数占程序全部语句数的百分比。故障定位的有效性表示方法为$Score=1-|E|/|P|$。其中，$|E|$表示语句怀疑度排名列表中需要检查的语句数；$|P|$表示程序语句数。

多数故障定位技术可以产生语句怀疑度的排名列表，因此Jones和Harrold的这种度量标准被广泛使用。但该方法在遇到语句怀疑度相同的情形时，采取一种称为最后行（last line，LL）的保守策略，检查怀疑度相同的所有语句。对此Ali等人[15]简单修正了LL策略，提出了中间行（middle line，ML）策略，只检查具有相同怀疑度语句块的中间语句之前的语句。Ali等人的实验结果表明，ML方法比LL方法的得分提升了5%，但是ML方法增加了漏报的可能，即降低了查全率。

4.3 效率

时间和空间复杂度是衡量算法效率的两个指标。随着计算机技术的发展，算法的空间需求比较容易满足，相比之下，人们更多关注时间效率。故障定位的时间耗费主要集中在分析程序结构、构建模型、程序插桩、运行测试、输入输出以及计算怀疑度等工作上。在实验评价故障定位算法时，一般用产生故障定位报告所需时间作为故障定位技术时间效率的度量指标。文献[58]通过实验比较了故障定位工具Falcon在进行故障定位时，对运行时间、共享内存访问次数等方面的影响。实验表明，故障定位由于需要运行插桩代码、计算分析等，明显增加了程序运行的时间和存储空间。

得益于计算机硬件技术的提高，人们更多关注故障定位有效性，即查全率和查准率。一般地，基于静态分析故障定位技术查全率较高，基于动态测试的故障定位技术查准率较高，故障定位需要有机结合两类分析技术，以提升故障定位的有效性。

5 未来研究展望

5.1 故障性质的研究

研究者多数借助抽象和简化来研究故障定位，忽略故障固有的一些特性。而进一步研究故障的固

有特性及故障之间的关系有助于故障定位,研究内容包括:

(1)故障模式。故障模式是故障在程序中的表现形式。文献[59]将故障归纳为四类模式:系统失效、系统不停机、输出域内的不期望的输出以及超出输出域的非法输出。FindBugs[23]通过预定义故障模式辅助程序员定位故障,但其定义的故障模式只能适用于Java程序。研究并建立适用于多种程序语言的通用故障模式,有助于准确定位其他语言的程序故障。

(2)故障修复代价。故障修复代价包括定位故障、理解并修复故障的耗费。多数现有的定位技术假设每一个故障的修复代价相同,但事实上,不同的故障修复代价未必相同,比如修复条件判断错误比修复简单语句错误更加困难。因此,可以在故障定位策略设计时考虑故障修复代价,制定相应的风险策略。

(3)多重故障的影响。现有的故障定位技术往往假设程序中只有一个故障,现实中程序经常会存在多个故障,而且故障之间相互作用可能产生相互掩盖效果,使得软件测试不能有效暴露故障症状,从而影响了故障定位的有效性。

5.2 故障定位新方法和新技术研究

故障定位还要拓展思路,探索新的方法和技术。可以从以下几个角度开展研究:

(1)利用软件开发过程信息。软件开发过程中许多信息有助于故障定位,规格说明文档、测试过程中的堆栈信息和输出信息、调试信息以及历史版本调试信息都可以作为故障定位参考信息。Sinha等人[60]提出了一种跟踪Java运行时的堆栈信息定位程序故障的方法。

(2)利用程序语义信息。程序功能具备相应的语义,这些语义信息可帮助理解程序,从而有助于定位和修复故障。Cellier[14]将形式概念分析用于故障定位,通过测试覆盖信息,将关联故障和形式概念分析结合,建立程序的规则格(具备规则上下文的概念格),再利用失败执行信息在规则格上匹配满足规则上下文的节点,从而定位故障。形式概念分析适合于小规模程序的故障定位。

(3)应用人工智能和数据挖掘技术。故障定位本质上是查找故障的过程,人工智能领域的一些研究成果可以应用于故障定位。Ali等人[15]研究了数据挖掘技术在故障定位中的应用,选取了基于规则的五种分类器进行故障定位,度量它们的故障定位精度,并与Tarantula等方法进行比较,在增加足够的代价信息的前提下,数据挖掘技术可以很好地识别与故障相关的程序特征,但只能产生间接的定位信息。随着计算机技术的发展,软件规模越来越大,Binkley估计到2025年人们开发的代码将达到万亿行[61]。面对数量庞大的代码,数据挖掘、机器学习等人工智能技术将会在故障定位方面得到很好的应用。

6 结束语

故障定位是软件调试的关键,自动故障定位将开发人员从耗时费力的手工检查程序中解放出来,提升了调试的效率。本文综述了现有的故障定位技术,首先从静态分析和动态测试的角度,将现有的定位技术分为两大类;接着根据故障定位使用的具体技术进一步细分,详细阐述了现有的代表性定位技术的基本原理和适用范围,分析比较了这些定位技术的主要成果与不足,给出了评测基准集,并讨论了故障定位有效性和效率的度量准则;最后展望了故障定位的研究趋势,提出了若干研究方向。

References:

[1] Jones J A, Bowring F J, Harrold M J. Debugging in parallel[C]// Proceedings of the 2007 International Symposium on Software Testing and Analysis (ISSTA '07), London, UK, July 9-12, 2007. New York, NY, USA: ACM, 2007: 16-26.

[2] Ball T, Eick S G. Software visualization in the large[J]. Computer, 1996, 29(4): 33-43.

[3] IEEE Std 610.12-1990 IEEE standard glossary of software engineering terminology[S]. 1990: 1-84.

[4] Xie Yichen, Chou A, Engler D. ARCHER: using symbolic, path-sensitive analysis to detect memory access errors[J]. ACM SIGSOFT Software Engineering Notes, 2003, 28(5): 327-336.

[5] Weiser M. Programmers use slices when debugging[J].

Communications of the ACM, 1982, 25(7): 446-452.

[6] Weiser M, Lyle J. Experiments on slicing-based debugging aids[C]//Proceedings of the 1st Workshop on Empirical Studies of Programmers. Washington, DC, USA: Ablex Publishing Corporation,1986: 187-197.

[7] Wong W E, Qi Yu. An execution slice and inter-block data dependency-based approach for fault localization[C]//Proceedings of the 11th Asia-Pacific Software Engineering Conference (APSEC ’04), Busan, Korea, Nov 30-Dec 3, 2004. Washington, DC, USA: IEEE Computer Society, 2004: 366-373.

[8] Zhang Xiangyu, Gupta R, Zhang Youtao. Efficient forward computation of dynamic slices using reduced ordered binary decision diagrams[C]//Proceedings of the 26th International Conference on Software Engineering (ICSE ’04), Edinburgh, UK, May 23-28, 2004. Washington, DC, USA: IEEE Computer Society, 2004: 502-511.

[9] Al-Khanjari Z A, Woodward M R, Ramadhan H, et al. The efficiency of critical slicing in fault localization[J]. Software Quality Journal, 2005, 13(2): 129-153.

[10] Sun Jirong, Li Zhshu, Ni Jiancheng, et al. Software fault localization based on testing requirement and program slice[C]//Proceedings of the International Conference on Networking, Architecture and Storage (NAS 2007), Guilin, China, July 29-31, 2007: 168-176.

[11] Renieris M, Reiss S P. Fault localization with nearest neighbor queries[C]//Proceedings of the 18th IEEE International Conference on Automated Software Engineering, Quebec, Canada, Oct 6-10, 2003. Washington, DC, USA: IEEE Computer Society, 2003: 30-39.

[12] 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 (ICSE ’02), Orlando, Florida, May 19-25, 2002. New York, NY, USA: ACM, 2002: 467-477.

[13] Liu Chao, Yan Xifeng, Fei Long, et al. SOBER: statistical model-based bug localization[J]. ACM SIGSOFT Software Engineering Notes, 2005, 30(5): 286-295.

[14] Cellier P. Formal concept analysis applied to fault localization[C]//Proceedings of the 30th International Conference on Software Engineering (ICSE Companion ’08), Leipzig, Germany, May 10-18, 2008. New York, NY, USA: ACM, 2008: 991-994.

[15] Ali S, Andrews J H, Dhandapani T, et al. Evaluating the accuracy of fault localization techniques[C]//Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE ’09). Washington, DC, USA: IEEE Computer Society, 2009: 76-87.

[16] Yu Kai, Lin Mengxiang. Advances in fault localization techniques[J]. Chinese Journal of Computers, 2011, 34(8): 1411-1422.

[17] Yu Kai, Lin Mengxiang, Gao Qing, et al. Locating faults using multiple spectra-specific models[C]//Proceedings of the 2011 ACM Symposium on Applied Computing (SAC ’11), Taichung, Taiwan, China, Mar 21-24, 2011. New York, NY, USA: ACM, 2011: 1404-1410.

[18] Wang Xinping, Gu Qing, Chen Xiang, et al. Research on software fault localization based on execution trace[J]. Computer Science, 2009, 36(10): 168-171.

[19] Xu Baowen, Nie Changhai, Shi Liang, et al. A software failure debugging method based on combinatorial design approach for testing[J]. Chinese Journal of Computers, 2006, 29(1): 132-138.

[20] Yan Jun, Zhang Jian. Combinatorial testing: principles and methods[J]. Journal of Software, 2009, 20(6): 1393-1405.

[21] Hao Dan, Zhang Lu, Pan Ying, et al. On similarity-awareness in testing-based fault localization[J]. Automated Software Engineering, 2008,15(2): 207-249.

[22] Aho A V. Compilers: principles, techniques and tools[M]. 2nd ed. [S.l.]: Addison Wesley, 2007.

[23] Ayewah N, Pugh W, Morgenthaler J D, et al. Using FindBugs on production software[C]//Proceedings of the 22nd ACM Conference on Object-Oriented Programming Systems and Applications Companion (OOPSLA ’07), Quebec, Canada, 2007. New York, NY, USA: ACM, 2007: 805-806.

[24] Shen Haihao, Zhang Sai, Zhao Jianjun, et al. XFindBugs: extended FindBugs for AspectJ[C]//Proceedings of the 8th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering (PASTE ’08)，Atlanta, USA，Nov 9-10, 2008. New York, NY, USA: ACM, 2008: 70-76.

[25] King J C. Symbolic execution and program testing[J]. Communications of the ACM, 1976, 19(7): 385-394.

[26] Young M, Taylor R N. Combining static concurrency analysis with symbolic execution[J]. IEEE Transactions on Software Engineering, 1988, 14(10): 1499-1511.

[27] Mei Hong, Wang Qianxiang, Zhang Lu, et al. Software analysis: a road map[J]. Chinese Journal of Computers, 2009, 32 (9): 1697-1710.

[28] Detlefs D, Nelson G, Saxe J B. Simplify: a theorem prover for program checking[J]. Journal of the ACM, 2005, 52(3): 365-473.

[29] Clarke E. Model checking foundations of software technology and theoretical computer science[M]. Berlin, Heidelberg: Springer-Verlag, 1997.

[30] Flanagan C, Leino K R M, Lillibridge M, et al. Extended static checking for Java[J]. ACM SIGPLAN Notices, 2002, 37(5): 234-245.

[31] Suzuki N. Analysis of pointer rotation[J]. Communications of the ACM, 1982, 25(5): 330-335.

[32] Cousot P, Cousot R. Static determination of dynamic properties of programs[J]. ACM SIGSOFT Software Engineering Notes, 1977, 2(2): 77-94.

[33] Cousot P, Cousot R. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints[C]//Proceedings of the 4th Symposium on Principles of Programming Languages, Los Angeles, USA, 1977. New York, NY, USA: ACM, 1977: 238-252.

[34] Hubert L, Jensen T, Pichardie D. Semantic foundations and inference of non-null annotations formal methods for open object-based distributed systems[M]. Berlin, Heidelberg: Springer-Verlag, 2008.

[35] Male C, Pearce D J, Potanin A, et al. Java bytecode verification for @NonNull types[C]//Proceedings of the 17th International Conference on Compiler Construction (CC '08), Budapest, Hungary, 2008. Heidelberg, Berlin: Springer-Verlag, 2008: 229-244.

[36] Spoto F. Precise null-pointer analysis[J]. Software and Systems Modeling, 2011,10(2): 219-252.

[37] 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 (ASE '05), Long Beach, California, USA, Nov 7-11, 2005. New York, NY, USA: ACM, 2005: 273-282.

[38] Abreu R, Zoeteweij P, van Gemund A J C. On the accuracy of spectrum based fault localization[C]//Proceedings of Testing: Academic and Industrial Conference, Practice and Research Techniques, Windsor, London, UK, Sep 12-14, 2007. Washington, DC, USA: IEEE Computer Society, 2007: 89-98.

[39] Jeffrey D, Gupta N, Gupta R. Fault localization using value replacement[C]//Proceedings of the 2008 International Symposium on Software Testing and Analysis (ISSTA '08), Seattle, USA, July 20-24, 2008. New York, NY, USA: ACM, 2008: 167-178.

[40] Wong W E, Qi Yu, Zhao Lei, et al. Effective fault localization using code coverage[C]//Proceedings of the 31st Annual International Computer Software and Applications Conference (COMPSAC '07), Beijing, China, July 23-27, 2007. Washington, DC, USA: IEEE Computer Society, 2007: 449-456.

[41] Wong W E, Wei Tingting, Qi Yu, et al. A crosstab-based statistical method for effective fault localization[C]//Proceedings of the 1st International Conference on Software Testing, Verification, and Validation (ICST '08), Lillehammer, Norway, Apr 9-11, 2008. Washington, DC, USA: IEEE Computer Society, 2008: 42-51.

[42] Liblit B, Naik M, Zheng A X, et al. Scalable statistical bug isolation[J]. ACM SIGPLAN Notices, 2005, 40(6): 15-26.

[43] Naish L, Lee H J, Ramamohanarao K. A model for spectra-based software diagnosis[J]. ACM Transactions on Software Engineering and Methodology, 2011, 20(3): 11-43.

[44] Dallmeier V, Lindig C, Zeller A. Lightweight defect localization for Java[C]//LNCS 3586: Proceedings of the 19th European Conference on Object-Oriented Programming, Glasgow, UK, July 25-29, 2005. Berlin, Heidelberg: Springer-Verlag, 2005: 528-550.

[45] Santelices R, Jones J A, Yu Yanbing, et al. Lightweight fault-localization using multiple coverage types[C]//Proceedings of the 31st International Conference on Software

Engineering (ICSE '09), Vancouver, Canada, May 16-24, 2009. Washington, DC, USA: IEEE Computer Society, 2009: 56-66.

[46] Baah G K, Podgurski A, Harrold M J. The probabilistic program dependence graph and its application to fault diagnosis[J]. IEEE Transactions on Software Engineering, 2010, 36(4): 528-545.

[47] De Lucia A. Program slicing: methods and applications[C]// Proceedings of the 1st IEEE International Workshop on Source Code Analysis and Manipulation (SCAM 2001), Florence, Italy, Nov 10, 2011. Washington, DC, USA: IEEE Computer Society, 2001: 142-149.

[48] Lyle J R. Evaluating variations on program slicing for debugging[D]. University of Maryland, USA, 1984.

[49] Lyle J R, Weiser M. Automatic program bug location by program slicing[C]//Proceedings of the 2nd International Conference on Computers and Applications, Beijing, China, 1987: 877-883.

[50] Agrawal H. Towards automatic debugging of computer programs[D]. Purdue University, USA, 1991.

[51] Liu Chao, Fei Long, Yan Xifeng, et al. Statistical debugging: a hypothesis testing-based approach[J]. IEEE Transactions on Software Engineering, 2006, 32(10): 831-848.

[52] Cleve H, Zeller A. Locating causes of program failures[C]// Proceedings of the 27th International Conference on Software Engineering (ICSE '05), St Louis, USA, May 15-21, 2005. New York, NY, USA: ACM, 2005: 342-351.

[53] Yilmaz C, Paradkar A, Williams C. Time will tell: fault localization using time spectra[C]//Proceedings of the 30th International Conference on Software Engineering (ICSE '08), Leipzig, Germany, May 10-18, 2008. New York, NY, USA: ACM, 2008: 81-90.

[54] Do H, Elbaum S G, Rothermel G. Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact[J]. Empirical Software Engineering, 2005, 10(4): 405-435.

[55] Artzi S, Dolby J, Tip F, et al. Directed test generation for effective fault localization[C]//Proceedings of the 19th International Symposium on Software Testing and Analysis (ISSTA '10), Trent, Italy, July 6, 2010. New York, NY, USA: ACM, 2010: 49-60.

[56] Artzi S, Dolby J, Tip F, et al. Practical fault localization for dynamic Web applications[C]//Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE '10), Cape Town, South Africa, May 2-8, 2010. New York, NY, USA: ACM, 2010: 265-274.

[57] Wang Xinming, Cheung S C, Chan W K, et al. Taming coincidental correctness: coverage refinement with context patterns to improve fault localization[C]//Proceedings of the 31st International Conference on Software Engineering (ICSE '09), Vancouver, Canada, May 16-24, 2009. Washington, DC, USA: IEEE Computer Society, 2009: 45-55.

[58] Park S, Vuduc R W, Harrold M J. Falcon: fault localization in concurrent programs[C]//Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE '10), Cape Town, South Africa, May 2-8, 2010. New York, NY, USA: ACM, 2010: 245-254.

[59] DeMillo R A, Pan H, Spafford E H. Failure and fault analysis for software debugging[C]//Proceedings of the 21 Annual International Computer Software and Applications Conference (COMPSAC '97), Aug 11-15, 1997. Washington, DC, USA: IEEE Computer Society, 1997: 515-521.

[60] Sinha S, Shah H, Gorg C, et al. Fault localization and repair for Java runtime exceptions[C]//Proceedings of the 18th International Symposium on Software Testing and Analysis (ISSTA '09), Chicago, Illinois, USA, July 19-23, 2009. New York, NY, USA: ACM, 2009: 153-163.

[61] Binkley D. Source code analysis: a road map[C]//Proceedings of Future of Software Engineering (FOSE '07), Minneapolis, USA, May 23-25, 2007. Washington, DC, USA: IEEE Computer Society, 2007: 104-119.

附中文参考文献：

[16] 虞凯, 林梦香. 自动化软件错误定位技术研究进展[J]. 计算机学报, 2011, 34(8): 1411-1422.

[18] 王新平, 顾庆, 陈翔, 等. 基于执行轨迹的软件缺陷定位方法研究[J]. 计算机科学, 2009, 36(10): 168-171.

[19] 徐宝文, 聂长海, 史亮, 等. 一种基于组合测试的软件故障调试方法[J]. 计算机学报, 2006, 29(1): 132-138.

[20] 严俊, 张健. 组合测试: 原理与方法[J]. 软件学报, 2009, 20(6): 1393-1405.

[27] 梅宏, 王千祥, 张路, 等. 软件分析技术进展[J]. 计算机学报, 2009, 32(9): 1697-1710.

JU Xiaolin was born in 1976. He is a Ph.D. candidate at School of Computer Science and Technology, China University of Mining and Technology, and a lecturer at Nantong University. His research interest is software analysis and testing.

鞠小林(1976—),男,江苏南通人,中国矿业大学计算机科学与技术学院博士研究生,南通大学计算机科学与技术学院讲师,主要研究领域为软件分析与测试。

JIANG Shujuan was born in 1966. She is a professor and Ph.D. supervisor at School of Computer Science and Technology, China University of Mining and Technology, and the member of CCF. Her research interests include compilation techniques and software engineering, etc.

姜淑娟(1966—),女,山东莱阳人,中国矿业大学计算机科学与技术学院教授、博士生导师,CCF 会员,主要研究领域为编译技术,软件工程等。

ZHANG Yanmei was born in 1982. She is a Ph.D. candidate at School of Computer Science and Technology, China University of Mining and Technology. Her research interests include exception handling, software analysis and testing, etc.

张艳梅(1982—),女,河北唐山人,中国矿业大学计算机科学与技术学院博士研究生,主要研究领域为异常处理,软件分析与测试等。

DONG Guowei was born in 1983. He is an associate researcher. His research interest is software analysis and testing.

董国伟(1983—),男,山东泗水人,副研究员,主要研究领域为软件分析与测试。