软件错误自动定位关键科学问题及研究进展

王克朝[1),2)] 王甜甜[1)] 苏小红[1)] 马培军[1)]

[1)](哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

[2)](哈尔滨学院软件学院 哈尔滨 150086)

摘 要 程序源代码中的缺陷是导致软件不可靠的一个主要原因.软件错误自动定位技术通过计算机分析程序源代码或执行过程中产生的运行状态,检测程序中的异常并将其独立出来作为需要进一步调试的可疑代码,从而缩小缺陷代码的搜索范围,辅助开发人员更快地识别缺陷语句并分析软件失效的产生原因.为了清晰地分析软件错误定位领域的关键科学问题,文中首先定义了"失效-错误定位-理解"模型,然后形式化地描述了软件错误定位相关概念;接下来,调研国内外最新研究进展,统计分析了发展趋势;重点分析了各种错误定位方法的基本思想、优缺点及其对关键问题的解决情况;最后总结了尚待解决的难点问题,指出了未来可能的研究方向.

关键词 软件失效;软件缺陷;错误定位;自动化调试;程序分析

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2015.02262

Key Scientific Issues and State-Art of Automatic Software Fault Localization

WANG Ke-Chao[1),2)] WANG Tian-Tian[1)] SU Xiao-Hong[1)] MA Pei-Jun[1)]

[1)](*School of Computer Science and Technology, Harbin Institute of Technology, Harbin* 150001)

[2)](*School of Software, Harbin University, Harbin* 150086)

Abstract Bugs in source code is a major cause of software unreliability. Automatic fault localization is a technique which detects program anomalies by analyzing the source code or runtime states of the software. It can isolate anomalies as suspicious code lines which need further debugging, so as to narrow down the searching space. With automatic fault localization techniques, software developers can identify bugs more quickly and understand the root cause of the software failure more deeply. In order to facilitate the analysis of the key issues in software fault localization, a "failure-fault localization-comprehension" model is defined, and the related concepts are formally described. Then, state-art of software fault localization is reviewed, and the development trend is statistically analyzed. Next, the main ideas, advantages and disadvantages of typical software fault localization methods are demonstrated in detail. Finally, key scientific problems which need further researched are summarized.

Keywords software failure; software bug; fault localization; automatic debugging; program analysis

---

收稿日期:2014-05-21;最终修改稿收到日期:2015-03-26.本课题得到国家自然科学基金(61202092,61173021)、高等学校博士学科点专项科研基金(20112302120052)、哈尔滨科技创新人才研究专项资金(RC2013QN010001)、黑龙江省普通高校青年学术骨干项目(1254G037)、黑龙江省教育科学"十二五"规划青年专项课题(GJD1214038)资助.**王克朝**,男,1980 年生,博士研究生,讲师,中国计算机学会(CCF)会员,主要研究方向为软件错误定位.E-mail: erickcwang@126.com. **王甜甜**,女,1980 年生,博士,副教授,主要研究方向为软件自动化调试、计算机辅助教学. **苏小红**,女,1966 年生,博士,教授,主要研究领域为软件缺陷检测. **马培军**,男,1963 年生,博士,教授,主要研究领域为软件工程、信息融合.

1　引　言

随着软件系统越来越复杂，软件经常不像人们预期的那样运行，换句话说，软件不总是可靠地运行，从而对计算机应用系统带来不利影响，甚至造成巨大的经济损失和灾难性的后果. 因此，保证软件的高可靠性已成为系统开发和维护工作的一个不可或缺的重要方面.

导致软件不可靠的一个主要原因是程序源代码中的缺陷. 程序设计是一项复杂的活动，很难推导程序中所有可能的执行路径，以及预见可能影响程序的环境因素. 即使程序看起来正确执行，仍然可能在极少情况下或特定条件满足时产生失效. 因此如何检测并消除软件缺陷是目前亟需解决的一个问题.

软件测试和调试协同工作可以有效检测并消除软件缺陷：测试用于暴露软件缺陷，调试用于消除这些软件缺陷. 然而软件调试过程中消除软件缺陷的速度往往跟不上软件测试过程中发现软件缺陷的速度. 目前已有很多自动化软件测试工具，然而，软件调试却大多采用设置断点等人工分析的方法，这很困难并且耗时，因为当程序开发人员在程序执行过程中发现软件失效时，可能已经离错误点很远了，需要花费大量的时间和精力来查找导致失效的程序代码.

如果能实现软件自动化调试，即由计算机自动找到程序代码中的缺陷位置、分析软件失效的产生原因，则可以更有效地确保软件可靠性，提高软件质量以及软件开发与维护效率. 因此近年来，软件自动化调试技术，特别是软件错误自动定位方法得到了广泛的关注.

文献[1]分析了软件失效机理和软件故障产生原因，讨论了软件故障模型. 文献[2]分类介绍了各种代表性错误定位技术的原理以及建模方法，给出了常用的评测基准集和评价标准，并展望了进一步研究方向.

不同于以上文献，本文重点分析软件错误定位领域的关键科学问题，提出了“失效-错误定位-理解”模型，并从这一角度论述各种已有方法对关键科学问题的解决程度，详细分析了尚待进一步研究的难点问题. 还针对顶级国际期刊和国际会议以及国内一级学报，统计分析了软件错误定位研究的发展趋势.

2　“失效-错误定位-理解”模型

为了清晰地分析软件错误定位领域中的关键科学问题，本课题在 Zeller[3] 提出的“缺陷-感染-失效”模型基础上定义了“失效-错误定位-理解”模型，如图 1 所示.

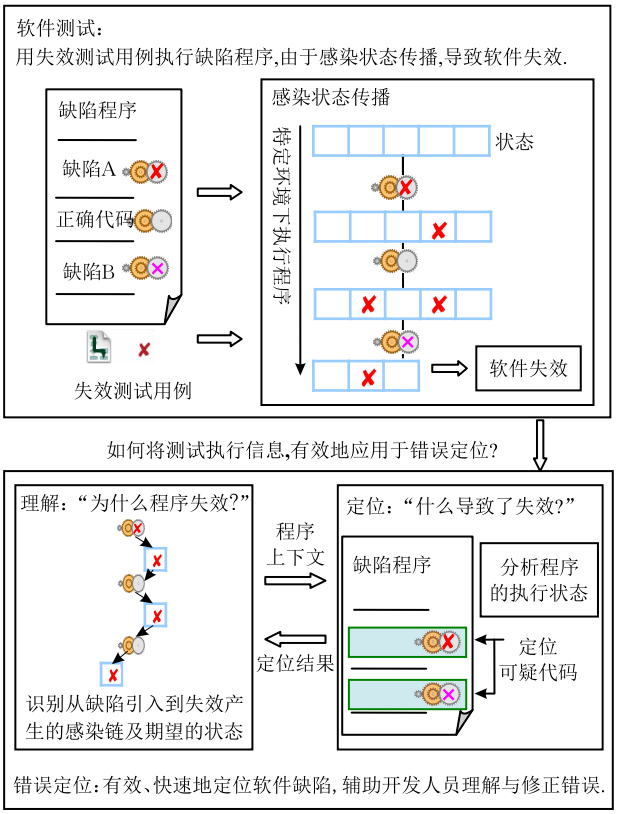

图 1　失效-错误定位-理解模型

软件开发人员在源代码中引入缺陷(defect)，在测试过程中：特定环境下，当用给定测试用例的输入执行程序时，缺陷代码可能会导致感染(infection)，生成错误的程序状态→感染被传播(infection propagation)→导致软件失效(failure).

在开发人员发现软件失效后，则需要调试程序：对于给定的测试用例集合，观察程序的执行状态，理解分析失效产生的上下文，识别可疑程序代码以及“缺陷→感染→失效”链，查找失效的根源，并消除软件缺陷，使失效不再发生.

错误定位及理解过程还可以是一个迭代过程，初步的定位的结果可以为理解程序“为什么会失效”提供参考，而程序分析及理解的结果可以用于提高错误定位的有效性.

然而，由于程序状态空间规模巨大，感染可能遍

布程序的执行,但是在开发人员看来却不相关.此外还有可能是多个缺陷共同作用引发失效,并且多个缺陷间也可能相互干扰,例如,缺陷 A 可能掩盖了缺陷 B,导致即使执行了缺陷 B 也不会产生失效.这都使理解"什么导致了失效"以及"为什么程序会失效"成为调试过程中最困难的任务,所需花费的时间和资源也比其他步骤要多得多,因此,这两个问题是软件错误定位领域亟需解决的关键科学问题.

在解决上述问题的过程中,以下两个关键因素决定了错误定位方法的可用性:

(1)有效性(effectiveness).根据定位结果,开发人员审查代码、真正查找到缺陷所需的工作量(expense).错误定位结果越准确,人工审查工作量越小,则该方法越有效.

(2)效率(efficiency).错误定位方法的计算复杂度及时间开销.

理想情况下,软件错误定位方法应该快速、准确地报告缺陷的位置,并且为开发人员进一步理解和修正缺陷提供充分的信息.

由图 1 还可以看出,错误定位与测试紧密相关,测试阶段提供的测试用例及其执行信息的质量也是决定错误定位质量的一个关键因素.因此"如何最大化地利用软件测试过程中提供的有效诊断信息,提高错误定位的有效性和效率"也是亟需解决的一个关键科学问题.

3 软件错误定位相关概念

根据定位过程中是否需要运行程序,可以将软件错误定位技术划分为两类:基于静态程序分析的技术和基于测试的技术.本文重点研究基于测试的软件错误定位技术,相关概念定义如下.

定义 1. 程序.程序 P 由 m 个程序实体组成,记为 $P=\{s_1,s_2,\cdots,s_m\}$,这里的程序实体指的是语句或者基本块(分支、函数、类等).

定义 2. 测试用例集.$T=\{t_k=(i_k,o_k)\mid 1\leq k\leq n\}$,其中 $t_k=(i_k,o_k)$ 表示测试用例,它的输入为 i_k,期望的输出为 o_k.

定义 3. 成功测试用例.称 $t_k=(i_k,o_k)$ 是一个成功测试用例,当且仅当用输入 i_k 执行程序 P 时,$o'_k==o_k$,其中 o'_k 为实际输出结果,o_k 为期望的输出结果.成功测试用例集记为 T_p.

定义 4. 失效测试用例.称 $t_k=(i_k,o_k)$ 是一个失效测试用例,当且仅当用输入 i_k 执行程序 P 时,$o'_k\neq o_k$,其中 o'_k 为实际输出结果,o_k 为期望的输出结果.失效测试用例集记为 T_f.

定义 5. 运行时信息集.$\Gamma=\{\tau(t_k)\mid 1\leq k\leq n\}$ 为用测试用例集 T 执行程序 P 的运行时信息集,其中 $\tau(t_k)$ 表示程序 P 执行测试用例 t_k 的运行时信息,如语句覆盖、分支覆盖、数据流、执行路径等.

定义 6. 软件错误自动定位.$FL(P,T,\Gamma,S_{可疑})$,由计算机分析程序源代码 P 或用测试用例集 T 执行 P 的运行时信息集 Γ,检测程序中的异常情况,并将其独立出来作为需要进一步调试的可疑代码 $S_{可疑}\subseteq P$,从而将与软件失效无关的代码自动过滤掉,缩小缺陷代码的搜索范围.理想情况下,准确指出缺陷代码的所在位置.

4 研究趋势统计分析

为了调研国际上软件错误定位方法的研究进展,本文查阅了顶级国际期刊(TSE、TOSEM、SPE、JSS、STVR)和顶级国际会议(ICSE、PLDI、FSE、ASE)中软件错误定位的相关文献,共 71 篇.论文发表数目总体上呈上升趋势,近 5 年发表了较多的研究论文.

按研究方法将这些文献归类的情况如图 2 所示.本文将主要的软件错误定位方法划分为:覆盖分析(coverage analysis)[4-40]、程序切片(program slicing)[41-52]、依赖分析(dependence analysis)[53-60]、状态变更(state alteration)[61-68]、模型检验(model based)[69-70]、程序不变量分析(invariant analysis)[71-72]、变异分析(mutation based)[73-74].其中,覆盖分析方法自 2002 年提出后,因具有计算复杂度低、实现简单等优点得到了广泛的研究,论文数占了总数的近二分之一,特别是在 2009 年和 2010 年发表了多篇文献,之后也受到持续的关注.程序切片作为一种最早提出的应用于软件调试的技术[75]也发表了为数众多的论文,但 2010 年前的研究比较多.依赖分析作为覆盖分析的一种改进方法,于 2007 年提出,近年来均有相关研究.模型检验、程序不变量分析和变异分析方法的研究起步比较晚,论文较少.

为了调研国内软件错误定位方法的研究进展,本文查阅了计算机学报、软件学报、计算机研究与发展、电子学报、电子与信息学报中关于软件"错误定位"、"故障定位"、"缺陷定位"等关键词的文章,共

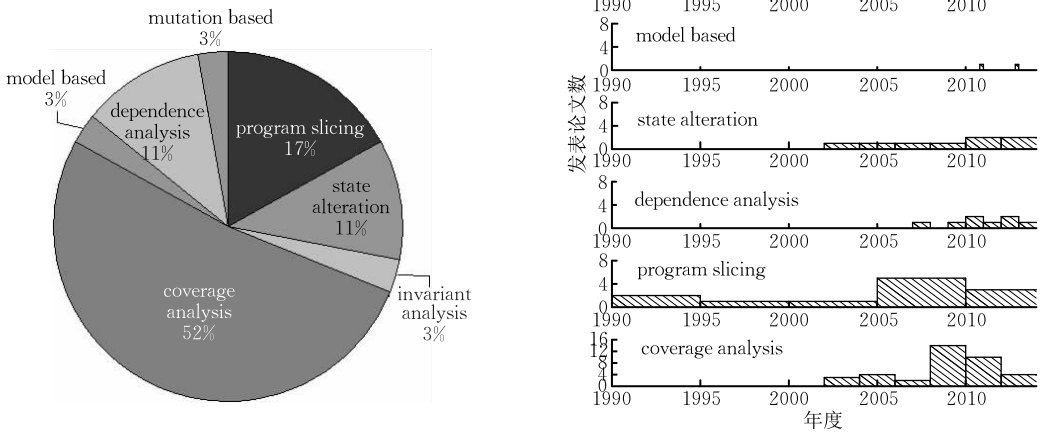

图 2　顶级国际期刊和会议中研究的软件错误定位方法统计

18 篇. 2006 年以前没有相关论文发表，大多数论文发表于 2010 年之后. 研究方法也是覆盖分析[76-81]、程序切片[82-83]和依赖分析[84-87]为主，其他方法还有变异分析[88]、组合测试[89-90]、机器学习[91-92]等.

综上所述，软件错误定位技术近年来成为一个研究热点. 然而，由于每种方法均有其适用范围，存在一系列关键难题需要进一步解决，目前的软件错误定位方法均处于理论与实验研究阶段，尚没有成熟的自动化软件错误定位工具广泛应用于工业界. 本文接下来对这些问题进行重点分析.

5　软件错误自动定位方法

本节论述了主流软件错误定位方法的基本思想，重点分析了各种方法的优缺点及其对关键科学问题的解决程度.

5.1　示例程序

使用图 3 中的示例程序 mid[20,74]对比分析代表性的软件错误定位方法. 程序 mid 的功能为：输入 3 个整数，然后输出这 3 个整数的中间值.

//Program mid. c #include〈stdio.h〉 int main(int *argc*, char **argv*[]) {	缺陷示例 P_1：s_9 修改为 $m=y$；						缺陷示例 P_2：s_5 修改为 if($y<z-1$) s_9 修改为 $m=y$；					
	t_1	t_2	t_3	t_4	t_5	t_6	t_1	t_2	t_3	t_4	t_5	t_6
int x, y, z, m;	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3	3,3,5	1,2,3	3,2,1	5,5,5	5,3,4	2,1,3
s_1: x=atoi($argv$[1]);	1	1	1	1	1	1	1	1	1	1	1	1
s_2: y=atoi($argv$[2]);	1	1	1	1	1	1	1	1	1	1	1	1
s_3: z=atoi($argv$[3]);	1	1	1	1	1	1	1	1	1	1	1	1
s_4: $m=z$;	1	1	1	1	1	1	1	1	1	1	1	1
s_5: if ($y<z$){	1	1	1	1	1	1	1	1	1	1	1	1
s_6:　if ($x<y$)	1	1			1	1	1					1
s_7:　　$m=y$;		1										
s_8:　else if ($x<z$)	1				1	1	1					1
s_9:　　$m=x$; }	1					1	1					1
s_{10}: else{			1	1				1	1	1	1	
s_{11}:　if ($x>y$)			1	1				1	1	1	1	
s_{12}:　　$m=y$;			1						1		1	
s_{13}:　else if ($x>z$)				1				1		1		
s_{14}:　　$m=x$;}												
s_{15}: printf("%d\n",m);}	1	1	1	1	1	1	1	1	1	1	1	1
成功/失效状态	p	p	p	p	p	f	p	f	p	p	f	f

图 3　示例程序 mid

本文在 mid 的基础上给出了两个含有缺陷的示例程序.其中,示例程序 P_1 包含一个赋值缺陷,将语句 s_9"$m=x$"修改为"$m=y$".示例程序 P_2 包含两个缺陷,在 P_1 的基础上又增加了一个谓词缺陷,将语句 s_5"if($y<z$)"修改为"if($y<z-1$)".

测试用例集 $T=\{t_1,t_2,t_3,t_4,t_5,t_6\}$,测试用例的成功/失效状态 p 表示测试用例执行成功,f 表示执行失效.对于 P_1 而言,成功测试用例集 $T_p=\{t_1,t_2,t_3,t_4,t_5\}$,失效测试用例集 $T_f=\{t_6\}$.对于 P_2 而言,$T_p=\{t_1,t_3,t_4\}$,$T_f=\{t_2,t_5,t_6\}$.

表格中的数字表示测试用例集 T 执行程序 P_1 和 P_2 的运行时信息 Γ,是语句覆盖信息,即如果执行测试用例 t_k 时,语句 s_i 被执行,则 s_i 和 t_k 对应的单元格数值为 1;否则为空白.

5.2 覆盖分析方法

覆盖分析方法通过对比失效执行和成功执行的程序元素(如语句、谓词、分支、基本块)的覆盖信息,定位可疑代码.主要思想是如果某个程序元素被较多的失效执行覆盖,却很少被成功执行覆盖,则该程序元素很可能含有缺陷.

这种方法通常包含如下步骤:

(1)执行测试用例,收集测试用例执行过程中的覆盖信息(即哪些程序元素被执行),以及测试用例的执行结果(成功/失效).

(2)统计分析失效执行和成功执行的程序元素覆盖信息,采用预先定义的度量公式,计算各个语句的可疑度,并将语句按照可疑程度由高到低排序,可疑度越高、排序越靠前则越可能是缺陷语句.

各种覆盖分析方法一般都遵从上述步骤,主要区别在于采用的度量公式不同.Naish 等人[32]用模型和实验评价了 36 种采用不同度量公式的覆盖分析方法.结果表明 Optimal 方法与其他方法相比,可以获得较优的错误定位有效性.证明尽管有些方法度量公式不同,但是对于 IF-ELSE2 型程序在可疑语句排序目的上是等价的.

表 1 列出了影响广泛的基于覆盖分析的软件错误定位技术,它们被 Naish 等人[32]证明对于 IF-ELSE2 型程序和多个方法等价,比如 Tarantula[4] 方法和 CBI[9]方法等价,Wong 方法[15]和其他 9 种方法等价.

这些方法应用于示例程序获得的基于语句覆盖的错误定位结果如表 2 所示.表格中的数值为语句的可疑度.对于示例程序 P_1,3 种方法均将缺陷语句 s_9 赋以最高可疑度,因此可以有效定位缺陷.但是对于示例程序 P_2 中的缺陷,各方法均没能较好地区分缺陷语句和非缺陷语句,需人工审查多条语句才能定位到缺陷语句.

表 1 影响广泛的基于覆盖分析的软件错误定位技术

错误定位技术	语句可疑度度量公式	等价方法
Tarantula	$\frac{\frac{a_{ef}}{a_{ef}+a_{nf}}}{\frac{a_{ef}}{a_{ef}+a_{nf}}+\frac{a_{ep}}{a_{ep}+a_{np}}}$	CBI
Wong	$a_{ef}-a_{ep}$	Rogers, SimpleMatching, Hamann, Sokal, M1, Euclid, Hamming, Manhattan, Lee
Optimal	$\begin{cases}-1, & a_{nf}>0\\ a_{np}, & \text{其他}\end{cases}$	O^p

a_{ef} 语句被测试用例集合中失效测试用例执行的次数.
a_{nf} 语句没有被测试用例集合中失效测试用例执行的次数.
a_{ep} 语句被测试用例集合中成功测试用例执行的次数.
a_{np} 语句没有被测试用例集合中成功测试用例执行的次数.

表 2 基于覆盖分析的方法分析示例程序的结果

语句	示例程序 P_1			示例程序 P_2		
	Tarantula	Wong	Optimal	Tarantula	Wong	Optimal
s_1	0.50	−4.00	0.00	0.50	0.00	0.00
s_2	0.50	−4.00	0.00	0.50	0.00	0.00
s_3	0.50	−4.00	0.00	0.50	0.00	0.00
s_4	0.50	−4.00	0.00	0.50	0.00	0.00
s_5	0.50	−4.00	0.00	0.50	0.00	0.00
s_6	0.63	−2.00	2.00	0.50	0.00	−1.00
s_7	0.00	−1.00	−1.00	0.00	0.00	−1.00
s_8	0.71	−1.00	3.00	0.50	0.00	−1.00
s_9	0.83	0.00	4.00	0.50	0.00	−1.00
s_{10}	0.00	−2.00	−1.00	0.50	0.00	−1.00
s_{11}	0.00	−2.00	−1.00	0.50	0.00	−1.00
s_{12}	0.00	−1.00	−1.00	0.50	0.00	−1.00
s_{13}	0.00	−1.00	−1.00	0.50	0.00	−1.00
s_{14}	0.00	0.00	−1.00	0.00	0.00	−1.00
s_{15}	0.50	−4.00	0.00	0.50	0.00	0.00

覆盖分析方法的优点是提供语句的可疑程度描述,并且由于不需要形式化建模,计算复杂度低,适于分析大规模程序.存在的主要问题是:

(1)通常只能揭示统计关联(变量的联合分布),而不能充分分析程序元素间的相互影响.当软件错误涉及多个程序元素间复杂交互的情况时,可能定位不到错误语句.例如,对于 P_2 中的缺陷语句 s_5,由于只统计语句覆盖,而语句 s_1、s_2、s_3、s_4、s_5 被相同的测试用例覆盖,因此,这些语句具有相同的可疑度值,导致无法准确区分缺陷语句和非缺陷语句.

(2)通常只检测可疑程序语句或谓词,缺少对错误行为状态的描述,需要由开发人员进一步判定是否存在错误.由于错误可能起源于失效点之前的任何位置,因此仅通过孤立的可疑语句或谓词来理解软件错误的产生原因是很困难的.

(3)对测试用例的质量要求较高.如果测试用

例选择不当，会导致冗余测试用例具有相同或类似的覆盖信息，这些冗余的覆盖信息可能降低错误定位的精度和效率[93]. 此外，覆盖分析方法试图查找与失效相关的语句，因此无法识别频繁执行但是没有导致失效的错误语句，因此巧合正确的测试用例[23]（即覆盖了错误语句但未引发失效运行结果的测试用例）会导致错误语句具有较低的可疑度，降低该方法的有效性. 例如，t_1 为巧合正确的测试用例，它执行了缺陷语句 s_9，但没有产生失效，如果在测试用例集 T 中增加与 t_1 具有相同语句覆盖的测试用例，则会导致 Tarantula 和 Wong 等方法计算 s_9 可疑度的 a_{ep} 值的增加，进而降低其可疑度.

针对覆盖分析方法对不同数据集的定位有效性不稳定的问题，丁晖等人[78]提出一种基于事件信息量的错误定位方法，根据测试信息中不同事件的类型及其发生的概率，结合语句的执行信息，动态计算和调整错误定位的结果，使之能够适应不同类型的数据集，尽可能地降低用例集对定位效果的影响.

5.3　程序切片方法

程序切片方法主要包括静态切片和动态切片. 静态切片方法通过静态分析源程序，根据依赖信息识别可能影响程序中某个位置的变量值的语句集合. 动态切片方法仅对当前输入生成的程序执行路径进行分析，识别实际影响给定程序执行点的变量值的语句集合. 动态切片方法还可以分为反向动态切片和前向动态切片. 在某个程序点的变量的反向动态切片包括影响该变量值的所有执行语句. 在某个程序点的变量的前向动态切片，则包括被该变量值影响的所有执行语句.

例如，当用 t_6 测试示例程序 P_1，发现在语句 s_{15} 处输出错误的 m 值时，对其执行动态反向切片的结果为 $\text{Backward}(t_6, m@s_{15}) = \{s_{15}, s_9, s_8, s_5, s_1\}$. 对其第 1 个参数 $argv[1]$ 执行动态前向切片的结果为 $\text{Forward}(t_6, argv[1]) = \{s_1, s_9, s_{15}\}$.

当用 t_5 测试示例程序 P_2，发现在语句 s_{15} 处输出错误的 m 值时，对其 $\langle s_{15}, m\rangle$ 执行动态反向切片的结果为 $\text{Backward}(t_5, m@s_{15}) = \{s_{15}, s_{12}, s_{11}, s_{10}, s_2\}$. 对其第 2 个参数 $argv[2]$ 执行动态前向切片的结果为 $\text{Forward}(t_5, argv[2]) = \{s_2, s_5, s_{10}, s_{11}, s_{12}, s_{15}\}$.

切片方法的优点是描述了失效产生的上下文，并且所需测试用例较少，通常只需要提供失效测试用例. 例如 $\text{Backward}(t_6, m@s_{15})$ 仅通过分析失效测试用例 t_6 的执行信息，输出了与语句 s_{15} 处错误的 m 值存在控制依赖关系的语句（s_8，s_5）和数据依赖关系的语句（s_9，s_1）. 通过分析该切片可以发现 s_9 处的缺陷语句. 切片方法的缺点是不提供语句的可疑程度描述，并且切片规模仍然可能很大，导致在切片中观察程序行为的代价较大.

Zhang 与 Gupta 等人研究发现动态切片通常可以包含导致失效产生的错误代码，并且动态切片的规模与实际执行的代码数量相比要小的多，但是切片中实际的代码规模仍然可能很大，仍可能包含大量的不会生成错误值的代码. 因此先后提出失效输入的前向动态切片、砍片（Failure inducing chop）[45]、带有置信度的动态切片剪枝[46]进一步缩小切片规模，但仍需要开发人员审查大量的代码.

文万志和李必信等人，针对面向对象程序提出了一种基于层次切片谱的软件错误定位技术，利用层次切片技术提取各层次错误相关元素，减小程序谱的规模，从而提高语句搜索的效率[82]. 又针对程序谱分析方法会随着程序中错误数目的增多而效率下降，提出了一种基于条件执行切片谱的多错误定位技术，根据输入变量的谓词条件构建错误相关条件执行切片的谱矩阵，然后依次计算错误相关条件执行切片中的语句或语句块的可疑度，以提高多错误定位的效率[83]. 这两种方法结合了程序切片和程序谱方法的优点.

5.4　依赖分析方法

针对覆盖分析方法缺少对程序执行状态及其转移的分析，无法有效分析失效感染状态传播的问题，研究人员提出了依赖分析方法. 依赖分析方法的基本思想类似于覆盖分析方法，不同之处在于在统计分析成功执行和失效执行的语句状态时考虑程序实体间的依赖关系，例如控制依赖和数据依赖.

Zhang 等人[25]提出了 CP（Capture Propagation）方法. 认为错误的程序状态会沿着控制依赖向后传播，因此通过抽象控制流来传播基本块的感染状态. 首先，对程序进行控制依赖分析，建立控制流图. 然后，计算每条控制依赖边传播错误的可疑值. 最后，模仿拓扑排序的过程，利用后续基本块的可疑值逆向推导前驱基本块的可疑值.

Baah 等人[55]在程序依赖图的基础上使用概率图模型描述程序的状态，定义了 PPDG（Probabilistic Program Dependence Graph），并将其应用于错误定位和理解. 下面介绍该方法分析示例程序的过程及结果.

图 4 是示例程序 P_1 的程序依赖图，其中节点表示程序语句，实线边表示节点之间的控制依赖，虚线

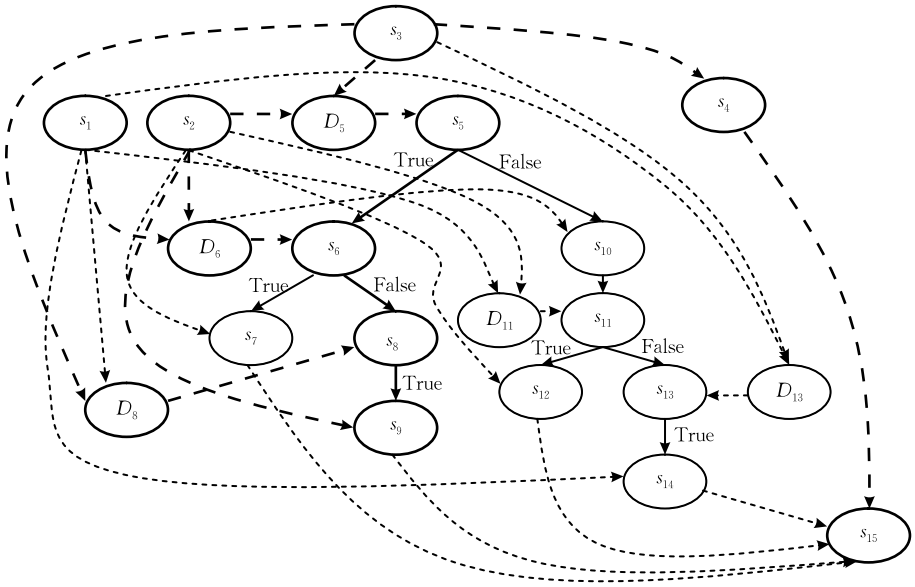

图 4 示例程序 P_1 的程序依赖图

边表示数据依赖. 控制依赖边上的 True 和 False 表示谓词状态. 此外 PPDG 还对传统的程序依赖图进行了改进,例如对谓词节点增加了数据依赖节点(表示为 D_i)等.

PPDG 中的每个节点都对应一个与程序执行相关的状态集合,并且每个节点状态都对应一个概率值. 示例程序 P_1 的节点状态及概率分布见表 3. 符号⊤表示节点被执行,⊥表示节点未被执行. 谓词节点状态表示为逻辑关系,数据流 $d_i(x)$ 表示节点 i 定义了变量 x. 例如节点 s_4 的状态集合为 $\{d_3(z),\perp\}$,表示 s_4 或者使用了节点 s_3 定义的变量 z,或者未被执行. 示例程序 P_2 和 P_1 具有相同的程序依赖图表示形式和节点状态,不同之处在于语句 s_5 处的谓词表达式不同以及程序的动态执行信息不同.

表 3 示例程序 P_1 的节点状态及概率分布

节点	状态	条件概率	节点	状态	条件概率
s_1,s_2,s_3	$\top,\perp$	$P(s_1),P(s_2),P(s_3)$	s_{10}	$<,>,==,\perp$	$P(s_{10}\mid D_5,s_5)$
s_4	$d_3(z),\perp$	$P(s_4\mid s_3)$	D_{11}	$(d_1(x),d_2(y)),\perp$	$P(D_{11}\mid s_1,s_2)$
D_5	$(d_2(y),d_3(z)),\perp$	$P(D_5\mid s_2,s_3)$	s_{11}	$<,>,==,\perp$	$P(s_{11}\mid D_{11},s_{10})$
s_5	$<,>,==,\perp$	$P(s_5\mid D_5)$	s_{12}	$d_2(y),\perp$	$P(s_{12}\mid s_{11},s_2)$
D_6	$(d_1(x),d_2(y)),\perp$	$P(D_6\mid s_1,s_2)$	D_{13}	$(d_1(x),d_3(z)),\perp$	$P(D_{13}\mid s_1,s_3)$
s_6	$<,>,==,\perp$	$P(s_6\mid D_6,s_5)$	s_{13}	$<,>,==,\perp$	$P(s_{13}\mid D_{13},s_{11})$
s_7	$d_2(y),\perp$	$P(s_7\mid s_6,s_2)$	s_{14}	$d_1(x),\perp$	$P(s_{14}\mid s_{13},s_1)$
D_8	$(d_1(x),d_3(z)),\perp$	$P(D_8\mid s_1,s_3)$			
s_8	$<,>,==,\perp$	$P(s_8\mid D_8,s_6)$	s_{15}	$d_4(m),d_7(m),d_9(m),$ $d_{12}(m),d_{14}(m)$	$P(s_{15}\mid s_4,s_7,s_9,$ $s_{12},s_{14})$
s_9	$d_2(y),\perp$	$P(s_9\mid s_8,s_2)$			

由于节点的状态受其控制依赖和数据依赖前驱节点的影响,因此节点状态的概率定义为条件概率. 这些概率值是通过用多组成功测试用例执行程序代码,捕获节点状态轨迹,然后统计学习获得的. 如果节点没有前驱节点则其状态概率为该状态在成功执行中出现的次数与该节点执行次数的比值,否则为其前驱节点在当前状态下的条件概率. 具有较高概率的状态是期望的程序状态.

在错误定位过程中,对于失效执行的各个节点状态,查找其在 PPDG 中的概率,并按照由低到高的顺序排列,具有较低概率值的节点状态具有较大的可疑度.

例如,失效测试用例 t_6 执行示例程序 P_1 的状态轨迹如表 4 左侧部分所示(对应于图 4 中加粗部分). 这些节点状态中,当前驱节点状态 $s_2=\top$ 时,$s_9=d_2(y)$ 具有最低的概率值,即语句 s_9 和 s_2 间对于变量 y 存在数据依赖关系最可疑. 因此可以定位缺陷语句 s_9. 失效测试用例 t_2 执行示例程序 P_2 的状态轨迹如表 4 右侧部分所示. t_2 执行了缺陷语句 s_5,但没有执行缺陷语句 s_9. 状态 $s_5=(==)$ 的概率值排序为 3. 由于没有被成功测试用例覆盖,状态 $s_{11}=<$ 和 $s_{13}=<$ 的可疑度为 0.

表 4　基于 PPDG 方法分析示例程序 P_1 和 P_2 的结果

示例程序 P_1		示例程序 P_2	
失效测试用例 t_6 执行状态轨迹	用成功测试用例集合 $T_p=\{t_1,t_2,t_3,t_4,t_5\}$ 学习的条件概率	失效测试用例 t_2 执行状态轨迹	用成功测试用例集合 $T_p=\{t_1,t_3,t_4\}$ 学习的条件概率
$\{(s_1:\top)$,	$P(s_1=\top)=P(s_2=\top)=P(s_3=\top)=1$	$\{(s_1:\top)$,	$P(s_5=(==)\mid D_5=(d_2(y),d_3(z)))=1/3=0.33$
$(s_2:\top)$,	$P(s_4=d_3(z)\mid s_3=\top)=1$	$(s_2:\top)$,	
$(s_3:\top)$,	$P(D_5=(d_2(y),d_3(z))\mid s_2=\top)=1$	$(s_3:\top)$,	$P(s_{10}=(==)\mid D_5=(d_2(y),d_3(z)))=2/3=0.67$
$(s_4:d_3(z))$,	$P(D_5=(d_2(y),d_3(z))\mid s_3=\top)=1$	$(s_4:d_3(z))$,	
$(D_5:(d_2(y),d_3(z)))$,	$P(s_5=<\mid D_5=(d_2(y),d_3(z)))=3/5=0.6$	$(D_5:(d_2(y),d_3(z)))$,	$P(s_{10}=(==)\mid s_5=(==)))=1/1=1$
$(s_5:<)$,	$P(D_6=(d_1(x),d_2(y))\mid s_1=\top)=3/5=0.6$	$(s_5:==)$,	$P(D_{11}=(d_1(x),d_2(y))\mid s_2=\top)=2/3=0.67$
$(D_6:(d_1(x),d_2(y)))$,	$P(D_6=(d_1(x),d_2(y))\mid s_2=\top)=3/5=0.6$	$(s_{10}:==)$,	$P(s_{11}=<\mid D_{11}=(d_1(x),d_2(y)))=0/2=0$
$(s_6:>)$,	$P(s_6=>\mid D_6=(d_1(x),d_2(y)))=1/3=0.33$	$(D_{11}:(d_1(x),d_2(y)))$,	$P(D_{13}=(d_1(x),d_3(z))\mid s_1=\top)=1/3=0.33$
$(D_8:(d_1(x),d_3(z)))$,	$P(s_6=>\mid s_5=<)=1/3=0.33$	$(s_{11}:<)$,	$P(D_{13}=(d_1(x),d_3(z))\mid s_3=\top)=1/3=0.33$
$(s_8:<)$,	$P(D_8=(d_1(x),d_3(z))\mid s_1=\top)=2/5=0.4$	$(D_{13}:(d_1(x),d_3(z)))$,	$P(s_{13}=<\mid D_{13}=(d_1(x),d_3(z)))=0/1=0$
$(s_9:d_2(y))$,	$P(D_8=(d_1(x),d_3(z))\mid s_3=\top)=2/5=0.4$	$(s_{13}:<)$,	
$(s_{15}:d_4(m))\}$	$P(s_8=<\mid D_8=(d_1(x),d_3(z)))=1/2=0.5$	$(s_{15}:d_4(m))\}$	
	$P(s_8=<\mid s_6=>)=1/1=1$		
	$P(s_9=d_2(y)\mid s_8=<)=1/1=1$		
	$P(s_9=d_2(y)\mid s_2=\top)=1/5=0.2$		
	$P(s_{15}=d_4(m)\mid s_4=d_3(z))=2/5=0.4$		

依赖分析方法的优点是不仅给出程序元素的可疑值，描述其可疑程度，还考虑了程序感染状态的传播，并且控制依赖或者数据依赖信息有助于理解失效产生的上下文.缺点是动态统计分析每个节点的状态及依赖关系(特别是数据依赖)具有较高的计算复杂度.此外，该方法本质上也是统计方法，对测试用例的数量和质量要求较高.

何加浪等人[86]提出了一种基于传播感知的程序故障定位方法.为了降低计算复杂度，该方法首先利用节点在成功执行路径和失效执行路径中出现的频率不同确定可疑空间中的每个节点的初始可疑度，接下来通过引入边传播趋势的概念确定初始可疑度最大的节点是否具有故障传播现象，最后对感知的故障传播相关节点进行可疑度修正来确定节点的最终可疑度.

赵磊等人[87]针对覆盖分析方法中代码覆盖率的独立统计忽略了程序内存在的复杂控制依赖和数据依赖的问题，提出程序失效规则及基于覆盖向量的覆盖信息分析模型，并在此模型基础之上，指出高可疑代码与错误代码在执行路径上的覆盖一致性，进而提出用于挖掘与高可疑代码相关联的错误代码的频繁集求解方法.

5.5　状态变更方法

状态变更方法在程序执行时获取或修改程序的状态，找出对测试结果(成功/失效)产生影响的关键程序元素，进而定位可能导致失效的代码.

Zeller 等人[61]提出了一种基于程序状态变更的错误定位技术 Delta debugging，开发了工具 igor，自动缩小程序成功执行和失效执行过程之间的区别，辅助错误定位.首先通过对源程序进行静态分析得到其定义的所有变量以及它们的值集，以此建立程序状态图.然后比较成功执行对应的程序状态图和失效执行对应的程序状态图，求出它们的最小公共子图，从而得到它们的差异.最后，利用这些差异变量，跟踪它们的值，直到找到错误语句[62].

igor 分析示例程序 P_1 的结果如图 5 所示.由于 igor 输出行号是自然行号(如包括函数名、变量声明语句等)，因此输出结果 sample1.c:8 对应于图 3 中的可执行语句 s_4.igor 通过对比失效测试用例 t_6(2,1,3)和成功测试用例 t_2(1,2,3)的执行状态值，将 t_6 在语句 s_4 和 s_8 处的 y 值以及 s_{15} 处的 m 值输出为与失效相关的状态.该结果没有直接定位到缺陷语句，需要进一步分析 s_8 处的选择分支，才能发现 s_9 处的缺陷.

```
输入信息：示例程序 P1，
失效测试用例 t6(2,1,3)，成功测试用例 t2(1,2,3)
igor 的输出结果：
'sample1' FAILS   if it outputs (?i)(\s|\A)1(\s|\Z)
'sample1' PASSES if it outputs (?i)(\s|\A)2(\s|\Z)
This is what happens when 'sample1' is invoked as 'sample1 2 1 3':
At 'main' (sample1.c:8(s4)), y is 1
At 'main' (sample1.c:12(s8)), y is 1
At 'main' (sample1.c:19(s15)), m is 1
```

图 5　igor 工具分析示例程序 P_1 的结果

igor 分析示例程序 P_2 的结果如图 6 所示，没有直接定位到缺陷语句.用同一个失效测试用例 t_2(1,2,3)但不同的成功测试用例执行 P_2 时，igor 在语句 s_4 处输出了不同的差异变量.可见其输出结果

受所选择的测试用例的影响.

输入信息:示例程序 P_2, 失效测试用例 $t_2(1,2,3)$,成功测试用例 $t_4(5,5,5)$
igor 的输出结果: 'sample2' FAILS if it outputs (?i)(\s|\A)3(\s|\Z) 'sample2' PASSES if it outputs (?i)(\s|\A)5(\s|\Z) This is what happens when 'sample2' is invoked as 'sample2 5 3 4': At 'main' (sample2.c:8(s_4)), z is 4 At 'main' (sample2.c:15(s_{11})), y is 3 At 'main' (sample2.c:19(s_{15})), m is 3
输入信息:示例程序 P_2, 失效测试用例 $t_2(1,2,3)$,成功测试用例 $t_3(3,2,1)$
igor 的输出结果: 'sample3' FAILS if it outputs (?i)(\s|\A)3(\s|\Z) 'sample3' PASSES if it outputs (?i)(\s|\A)2(\s|\Z) This is what happens when 'sample3' is invoked as 'sample3 5 3 4': At 'main' (sample3.c:8(s_4)), y is 3 At 'main' (sample3.c:15(s_{11})), y is 3 At 'main' (sample3.c:19(s_{15})), m is 3

图 6 igor 工具分析示例程序 P_2 的结果

由于程序状态空间巨大,修改所有可能的状态是不现实的. Zhang 等人[63]研究发现,很多错误都会导致程序谓词状态的改变,从而影响分支的选择. 基于此,提出了 Predicate Switching 方法. 它通过强制修改谓词状态,使得分支选择发生改变. 如果某个失效执行因为某个谓词状态的强制改变而变成成功执行的话,称此谓词为关键谓词. 通过查找关键谓词定位软件缺陷.

状态变更方法的优点是通过识别对运行时的程序状态做哪些修改有助于使失效执行转变为成功执行,可以为开发人员理解和修正软件错误提供有效参考. 缺点是须确保内存修改的语义一致性,并且对于复杂系统而言由于程序状态数目庞大,算法计算复杂度高,时间开销大.

5.6 模型检验方法

模型检验方法推导期望的程序行为模型,并通过检测失效执行对期望行为的违背情况识别错误行为.

Jose 等人开发了 BugAssist 工具[69],将错误定位问题转换为"最大可满足"问题. BugAssist 的输入是一个源程序,需要人工为该程序插桩断言,用以规格说明程序恒定满足的正确行为. 首先使用边界模型检验获得测试执行的边界,并将程序的边界语义编码为布尔表达式. 然后对失效执行创建一个不可满足表达式. 最后使用"最大可满足"规则,查找可以同时满足该表达式的最大短语集合,将其输出为可能的失效根源.

例如,对示例程序 P_1 和 P_2 分别插入断言 assert ($m==y\&\&m>=z\&\&m<=x \| m==x\&\&m>=z\&\&m<=y \| m==z\&\&m>=y\&\&m<=x \| m==x\&\&m>=y\&\&m<=z \| m==z\&\&m>=x\&\&m<=y \| m==y\&\&m>=x\&\&m<=z$);然后使用 BugAssist 分析 P_1,程序跟踪窗口中语句"$m=y$;"被蓝色加亮,表示该语句被识别为潜在的缺陷位置. 用 BugAssist 分析 P_2,正确识别了缺陷语句"if($y<z-1$)",但没有同时识别出缺陷语句"$m=y$;",需要在修正缺陷语句"if($y<z-1$)"之后再次检测. 模型检验方法的优点是结果准确,并能提供期望的行为状态或失效执行的违背情况描述,有助于理解和修正错误. 缺点是需要指定断言,而确定程序中恒定满足的属性分析过程比较复杂,且对于复杂系统而言,模型检验方法形式化模型逻辑推理的复杂度较高,开销很大.

5.7 不变量分析方法

不变量分析方法首先运行多个成功测试用例训练提取程序中的不变量(恒定满足的属性),然后运行失效测试用例,检测违背的不变量,将其作为导致失效的原因.

Hangal等人[71]开发了 DIDUCE(Dynamic Invariant Detection U Checking Engine),通过执行 Java 程序动态提取不变量,然后分析程序的执行行为,检测不变量违背,进而检测异常,并根据置信度值将异常排序. 如果表达式被执行多次,并且所观测的值很少违背,则具有较高的置信度. DIDUCE 可分析的程序属性包括某个变量的值是否总是相同、表达式的值是否总是正或者负、值的奇偶性、值的近似上界等. DIDUCE 包含两种工作模式:训练模式和检验模式. 在训练模式根据需要放宽不变量假设来学习不变量,在检验模式当放宽不变量时发出消息. 报告产生每个不变量违背的旧的不变量的置信度值和新的不变量的置信度值. 较大的置信度下降值是一个值得注意的不变量违背,对应的语句可能含有错误.

例如,分析值的范围不变量,对于示例程序 P_1 用成功测试用例集 $T_p=\{t_1,t_2,t_3,t_4,t_5\}$ 学习获得的各个赋值及输出变量的值的范围为 $s_1:x(1,5)$, $s_2:y(2,5)$, $s_3:z(1,5)$, $s_4:m(1,5)$, $s_7:m(2,2)$, $s_9:m(3,3)$, $s_{12}:m(2,2)$, $s_{15}:m(2,5)$. 失效测试用例 $t_6(2,1,3)$ 的执行对该值范围不变量的违背情况如下:$s_2:y(1)$, $s_9:m(1)$, $s_{15}:m(1)$. 通过分析该不变量违背可发现语句 s_9 处变量 m 赋值为 1 存在错误.

对示例程序 P_2 用成功测试用例集 $T_p=\{t_1,t_3,t_4\}$ 学习获得的各个赋值及输出变量的值的范围为 $s_1:x(3,5)$, $s_2:y(2,5)$, $s_3:z(1,5)$, $s_4:m(1,5)$,

$s_7:m(-\infty,+\infty)$，$s_9:m(3,3)$，$s_{12}:m(2,2)$，$s_{15}:m(2,5)$. 失效测试用例 $t_2(1,2,3)$的执行对该值范围不变量的违背情况如下：$s_1:x(1)$. 仅根据该结果难于定位缺陷位置.

不变量分析方法的优点是利用成功测试用例学习得到的不变量有助于分析软件的期望行为和属性. 存在的问题是对测试用例的质量要求较高. 当使用大量的任意测试用例(如软件测试中使用的所有测试用例)进行训练，提取不变量，可能导致程序不变量的范围太宽，产生大量误检，无法准确定位到失效的产生根源；而任意选择测试用例可能导致覆盖率低，训练不充分，很多语句没有被执行到，使得真正的失效产生原因没有被包含到候选集合中，导致漏检. 例如 P_2的值的范围不变量分析结果没能有效指出缺陷语句.

5.8　变异方法

基于变异的错误定位方法将变异技术应用于错误定位. 变异分析基于简单的语法规则转换程序语句，把原始程序转换成若干个变异(mutation)程序，从而向程序中植入缺陷. 变异测试通过用测试用例集执行变异程序，然后检查变异程序和原始程序版本间的行为差异. 对于给定的测试用例集，如果存在某个测试输入，使得变异程序产生的输出与原始程序的输出不同，则称该变异“被杀死”(killed)，否则称该变异是“活的”(lived).

基于变异的错误定位方法利用变异测试技术的错误植入能力，在特定程序语句上产生符合语法的程序错误，并利用测试结果变化识别可疑的变异并利用它们的位置来定位错误语句. 基于的假设是位于同一位置的错误语句及其变异具有相似的行为，因此被失效测试用例杀死的变异，很可能指出了错误位置.

Papadakis 等人[74]提出了一种基于变异分析的错误定位方法 Metallaxis. 首先对程序中的每个可执行语句执行 1 阶变异转换(即执行一次语法修改)，生成若干个变异程序，并执行变异测试. 然后，将被杀死的变异看做被覆盖的元素，活的变异看做未被覆盖的元素，利用已有的覆盖分析方法(Ochiai[16])通过度量成功测试和失效测试用例杀死变异的数目，计算各个变异的可疑度值. 最后，根据变异的可疑度，给原始程序语句赋可疑度值. 对于同一语句的多个变异取其中的最大值作为该语句的可疑度值.

例如，对示例程序 P_2采用关系变异操作算子生成 35 个变异程序. 这些变异体中，语句 s_5的变异体 $<\rightarrow<=$，即将 if ($y<z-1$)修改为 if ($y<=z-1$)，被两个失效测试用例(t_2和 t_5)杀死，而未被任何成功测试用例杀死，由 Ochiai 公式计算得到可疑度值为 0.82，高于其他变异体的可疑度，因此所对应的语句 s_5被定位为可疑语句. 采用关系变异运算符可以定位关系运算相关的软件缺陷，但是无法定位 s_9那样的赋值语句缺陷.

如果对示例程序 P_2采用变量替换变异算子，语句 s_9的变异体 $y\rightarrow x$ 被 1 个失效测试用例 t_6杀死，未被任何成功测试用例杀死，由 Ochiai 公式计算得到可疑度值为 1.0，因此 s_9被定位为可疑语句. 采用变量替换变异算子可以有效定位这类变量引用错误的缺陷.

对于示例程序 P_1，可采用算数常量增 1 和减 1 算子生成 32 个变异程序，其中缺陷语句 s_9“$m=y$”的两个变异体 $y\rightarrow y+1$ 和 $y\rightarrow y-1$ 的可疑度为 0.71 排列在第一位，因此 s_9被识别为最可疑语句. 文献[74]中给出了详细分析结果.

基于变异的错误定位方法的优点是将变异测试过程与错误定位过程统一，并且由于其要求由变异引入的差异必须影响程序的输出，因此可以有效处理巧合正确的测试用例. 缺点是变异测试的复杂度较高，并且错误定位的有效性依赖于变异算子和测试用例的选择.

6　目前难点及未来可能的研究方向

已有软件错误定位方法为软件自动化调试奠定了良好的研究基础，然而从以上分析可以看出，目前的软件错误定位方法的研究大多致力于回答“什么导致了失效”，即指出可疑语句的位置，但通常缺少对“为什么程序会失效”的分析，不能为理解软件失效产生的上下文提供充分的信息. 定位结果的有效性，特别是分析多缺陷程序的有效性，还需要进一步提升. 此外，还缺少对“如何最大化地利用软件测试中提供的诊断信息，使其提高错误定位的有效性”的考虑. 因此，软件错误定位领域目前仍存在以下难点问题需要进一步分析与解决. 这些问题能否良好解决直接决定着错误定位方法的有效性及其能否被真正应用于实际的软件自动化调试过程中.

(1) 如何优化软件测试过程中测试用例的执行信息，提高错误定位的有效性？

测试过程中提供了大量的测试用例及其执行信

息,错误定位方法可以在这些信息的基础上进行分析,提取可疑程序元素.然而满足测试准则的测试用例集合,未必能为错误定位提供有效的信息.这是因为测试的目标是较高的代码覆盖程度,尽可能地揭示失效,而错误定位的目标是减小识别的覆盖代码集合,尽可能地确定含有缺陷的程序元素.Jiang 等人[93]通过实验验证测试的充分覆盖准则并不能充分确保错误定位的有效性.Gonzalez-Sanchez 等人[94]研究表明测试用例的选择是提高调试效率的关键.因此,测试用例的选择会对错误定位的有效性和效率产生显著的影响.

然而,已有关于测试用例选择的研究大多集中于如何提高测试和错误定位的效率,而缺少关于如何提高错误定位有效性的研究.Hao 等人[95]研究了基于语句覆盖的测试用例约简策略,将其应用于错误定位方法 Tarantula 上,结论是减少冗余测试用例有利于提高错误定位的有效性.Yu 等人[16]实验分析了测试用例约简对覆盖分析方法有效性的影响,得到的结论是与基于语句覆盖的测试用例约简方法相比,基于语句向量的测试用例约简方法删除了较少的测试用例,但约简后的测试用例更有利于错误定位,错误定位技术的有效性取决于所使用的约简策略.因此如何选择有利于提高错误定位方法有效性的测试用例值得进一步深入研究.

测试用例的巧合正确性也是影响错误定位(特别是覆盖分析方法)的有效性的一个重要因素[23].然而如何降低巧合正确性的影响却是个难点问题.Wang 等人[23]考虑了丢失语句情况下的巧合正确性问题,提出利用控制流和数据流模式(上下文模式)进一步精化代码覆盖信息,排除巧合正确的测试用例引入的错误覆盖信息.该方法的前提是开发人员预先知道错误类型,但在实际软件开发过程中该假设不一定成立.Masri 等人[96]提出启发式方法来区分巧合正确的测试用例,首先识别"出现在失效执行和少量(不为 0)成功执行中的程序元素"作为可能与巧合正确性相关的程序元素,然后分析测试用例对这些元素的覆盖情况,判定其是否是巧合正确的测试用例.该方法具有较高的误检率,导致无法显著提高错误定位的有效性.

此外,以上研究大多假设已有足够的测试用例信息,但有些情况下已有信息不足以有效执行错误定位,这就需要研究如何生成适用于错误定位的测试用例.

在回归测试中,测试和错误定位过程通常交替执行:开发人员测试程序,检测之前对程序所做的修改是否引入新的缺陷,如果产生失效,则执行错误定位和修正,并在修正了错误后再次执行回归测试.这就引入了测试用例优先排序问题"在发现失效后,按什么顺序选择执行剩余的测试用例才能有助于尽早、有效地定位到软件错误".Jiang 等人[97]实验分析了应用于软件测试中的测试用例优先排序技术对覆盖分析错误定位技术的有效性的影响,结果表明测试用例优先排序的策略和时间开销是影响错误定位有效性的关键因素.Gonzalez-Sanchez 等人[36]针对软件错误定位,提出使用贝叶斯理论计算错误定位的可疑度并对整个测试用例集进行排序,基于前次执行的成功/失效结果,动态选择测试用例,使每次测试具有较高的诊断性能.Yoo 等人针对覆盖分析错误定位方法,提出了基于信息熵理论排序测试用例[40],以提高错误定位的有效性.该方法适用于 Tarantula 等覆盖分析方法,而是否适合其他方法还需要进一步研究与分析.

综上所述,面向有效错误定位的测试用例优选、自动生成、优先排序技术是未来的重点研究方向.

(2)如何使软件错误定位方法在回答"什么导致了软件失效"时,既具有较低的计算复杂度,又能充分分析程序的执行状态传播及程序元素间的相互影响,从而有效地定位缺陷?

状态变更方法和模型检查方法提供了错误行为描述,但是这两种方法复杂度过高,难以应用于实际的软件调试过程中.覆盖分析方法计算复杂度低,但是通常只考虑单个程序元素的可疑程度,而忽略了程序感染状态的传播,当软件错误涉及多个程序元素间复杂交互的情况下,可能定位不到错误语句.

针对上述问题,Masri[56]提出基于信息流覆盖的错误定位方法,使用信息流捕获语句和程序变量间的交互,又进一步提出错误相关依赖链的概念[98];Baah 等人[57]提出基于因果图的覆盖分析方法.这 3 种方法考虑了程序的控制和数据依赖,为分析程序元素间的相互影响奠定了基础,但仍存在以下不足之处:① 缺少对期望执行状态及其转移的描述;② 信息流、依赖链、因果图的数目众多,动态跟踪大量测试用例执行时的动态数据依赖信息,计算复杂度高,有时会得到不准确的结果[56].

轻量级的、解析失效感染状态传播的错误定位方法,将有助于自动化软件错误定位工具在工业界的实际应用与推广,值得进一步研究与探讨.

(3)如何自动分析可疑程序点的执行状态上下文,定位失效的产生根源,为开发人员理解与修正软件缺陷提供充分的信息?

大多已有错误定位方法着重于缺陷定位,通常仅报告可疑代码行,而忽略了一个事实:理解一个失效的产生原因通常涉及复杂的活动. Parnin 和 Orso[99] 基于错误定位方法 Tarantula 进行实验,分析开发人员调试程序时的行为,得到的结论是很多情况下,开发人员在看完错误定位工具列出的可疑语句后,需要花费大量的时间来审查语句. 如果不理解为何失效,开发人员很难修正缺陷. 因此仅给出可疑语句是不够的,需要更多的关于失效产生的上下文信息来辅助开发人员理解软件错误.

程序不变量分析方法有助于检测失效的产生根源,但是缺少对错误传播的分析,并且存在大量的误检,错误定位的有效性存在局限性. Jiang 与 Su[53] 提出上下文提醒的统计分析调试方法,通过创建错误控制流路径将缺陷指示器(可能包含错误的谓词语句)连接在一起,辅助开发人员理解软件错误. Cheng 等人[100] 通过图挖掘算法识别失效产生的上下文. 然而,这两种方法只考虑了程序的控制结构,没有分析数据依赖信息,因此只能识别控制流相关的错误,无法有效识别数据流相关的错误;此外,只给出控制结构上下文,仍然无法充分辅助开发人员理解与修正错误.

综上,错误定位结果不但应给出可疑程序语句,还应为理解"可疑语句如何与软件错误相关,它们为什么会导致软件失效,以及如何修正这些错误"提供充分的信息. 如果能构建一个多维全景视图,自动分析推导失效相关的控制依赖及数据依赖、失效执行路径上的值以及可疑程序点的期望状态和表达式等信息,则可以有效辅助开发人员理解这些问题. 程序理解方法与软件错误定位方法的有机结合将有助于解决该难点问题.

(4)在存在多个失效测试用例的情况下,如何区分失效特征,避免多个错误间的相互干扰,并且深入考虑错误间的关联关系,有效定位多缺陷程序中的错误?

Wong 等人[27] 开发的工具 xDebug 通过对基于语句覆盖的错误定位方法进行实验得出结论:无论是对于成功测试用例集合还是失效测试用例集合,其中第 k 个测试用例比第 $k+1$ 个测试用例对错误定位的贡献大;成功测试用例集合的对错误定位的总贡献小于失效测试用例集合对错误定位的总贡献. 该研究给我们的启示是不同的测试用例对错误定位具有不同的贡献,并且在错误定位过程中尤其要充分考虑失效测试用例对错误定位的影响.

Debroy 与 Wong[101] 研究 siemens 基准测试程序组发现,程序中多个错误可能以多种方式相互作用,由一个错误导致失效的测试用例,在加入另一个错误时可能不会失效,因为它可能掩盖了第一个错误的失效因果. 此外,某个失效的产生可能会伴随着其他失效,这种错误间的相关性使得多缺陷程序的错误定位问题变的十分复杂. 然而 DiGiuseppe 等人[102] 研究错误数量对覆盖分析错误定位方法的影响的实验结果表明,有些错误的定位结果与其他错误是否出现无关. 通常即使存在多个错误,定位技术每次也能有效定位其中的一个错误. 然而如何自动区分多个错误间的关系到底属于以上哪种情况是一个难点问题,该问题的解决直接决定着多缺陷程序错误定位方法的有效性.

将错误关联、失效特征聚类、机器学习等技术应用于错误定位中将有助于解决该问题,值得深入开展相关研究.

7　结束语

软件自动错误定位技术是实现软件自动化调试、提高软件可靠性及开发和维护效率的关键,具有广阔的应用前景和发展空间. 本文深入调研了国内外软件工程方向的顶级期刊和会议,提出了"失效-错误定位-理解"模型,详细分析了软件错误自动定位领域的关键科学问题与发展趋势,展望了未来的研究方向.

参　考　文　献

[1] Shan Jin-Hui, Xu Ke-Jun, Wang Ji. A kind of software faults diagnosing framework. Chinese Journal of Computers, 2011, 34(2): 371-382(in Chinese)
(单锦辉, 徐克俊, 王戟. 一种软件故障诊断过程框架. 计算机学报, 2011, 34(2): 371-382)

[2] Yu Kai, Lin Meng-Xiang. Advances in automatic fault localization techniques. Chinese Journal of Computers, 2011, 34(8): 1411-1422(in Chinese)
(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(8): 1411-1422)

[3] Zeller A. Why Programs Fail: A Guide to Systematic Debugging. 2nd Edition. Amsterdam, Holland: Elsevier, 2009

[4] Jones J A, Harrold M J, Stasko J. Visualization of test information to assist fault localization//Proceedings of the 24th International Conference on Software Engineering. Florida, USA, 2002: 467-477

[5] Renieres M, Reiss S P. Fault localization with nearest neighbor queries//Proceedings of 18th IEEE International Conference on the Automated Software Engineering. Montreal, Canada, 2003: 30-39

[6] Liblit B, Aiken A, Zheng A X, et al. Bug isolation via remote program sampling//Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation. San Diego, USA, 2003: 141-154

[7] Jones J A. Fault localization using visualization of test information//Proceedings of the 26th IEEE International Conference on Software Engineering. Scotland, UK, 2004: 54-56

[8] Liu C, Yan X, Fei L, et al. SOBER: Statistical model-based bug localization//Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Lisbon, Portugal, 2005: 286-295

[9] Liblit B, Naik M, Zheng A X, et al. Scalable statistical bug isolation//Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation. Chicago, USA, 2005: 15-26

[10] Jones J A, Harrold M J. Empirical evaluation of the tarantula automatic fault-localization technique//Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. Long Beach, USA, 2005: 273-282

[11] Liu C, Fei L, Yan X, et al. Statistical debugging: A hypothesis testing-based approach. IEEE Transactions on Software Engineering, 2006, 32(10): 831-848

[12] Baudry B, Fleurey F, Le Traon Y. Improving test suites for efficient fault localization//Proceedings of the 28th International Conference on Software Engineering. Shanghai, China, 2006: 82-91

[13] Cellier P. Formal concept analysis applied to fault localization //Proceedings of the 30th International Conference on Software Engineering. Leipzig, Germany, 2008: 991-994

[14] Yilmaz C, Paradkar A, Williams C. Time will tell: Fault localization using time spectra//Proceedings of the 30th International Conference on Software Engineering. Leipzig, Germany, 2008: 81-90

[15] Wong W E, Wei T, Qi Y, et al. A crosstab-based statistical method for effective fault localization//Proceedings of the 2008 1st International Conference on Software Testing, Verification, and Validation. Lillehammer, Norway, 2008: 42-51

[16] Yu Y, Jones J A, Harrold M J. An empirical study of the effects of test-suite reduction on fault localization//Proceedings of the 30th International Conference on Software Engineering. Leipzig, Germany, 2008: 201-210

[17] Abreu R, Zoeteweij P, Golsteijn R, et al. A practical evaluation of spectrum-based fault localization. Journal of Systems and Software, 2009, 82(11): 1780-1792

[18] Ali S, Andrews J H, Dhandapani T, et al. Evaluating the accuracy of fault localization techniques//Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering. Washington, USA, 2009: 76-87

[19] Chilimbi T M, Liblit B, Mehra K, et al. HOLMES: Effective statistical debugging via efficient path profiling// Proceedings of the IEEE 31st International Conference on Software Engineering. British Columbia, Canada, 2009: 34-44

[20] Santelices R, Jones J A, Yu Y, et al. Lightweight fault-localization using multiple coverage types//Proceedings of the IEEE 31st International Conference on Software Engineering. British Columbia, Canada, 2009: 56-66

[21] Janssen T, Abreu R, Gemund A J C. Zoltar: A toolset for automatic fault localization//Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering. Washington, USA, 2009: 662-664

[22] Weiglhofer M, Fraser G, Wotawa F. Using spectrum-based fault localization for test case grouping//Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering. Washington, USA, 2009: 630-634

[23] Wang X, Cheung S C, Chan W K, et al. Taming coincidental correctness: Coverage refinement with context patterns to improve fault localization//Proceedings of the 31st International Conference on Software Engineering. British Columbia, Canada, 2009: 45-55

[24] Hao Dan, Zhang Lingming, Zhang Lu, et al. VIDA: Visual interactive debugging//Proceedings of the 31st International Conference on Software Engineering. British Columbia, Canada, 2009: 583-586

[25] Zhang Z, Chan W K, Tse T H, et al. Capturing propagation of infected program states//Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. New York, USA, 2009: 43-52

[26] Abreu R, Mayer W, et al. Refining spectrum-based fault localization rankings//Proceedings of the 2009 ACM Symposium on Applied Computing. Hawaii, USA, 2009: 409-414

[27] Wong W E, 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

[28] Zhang Z, Jiang B, Chan W K, et al. Fault localization through evaluation sequences. Journal of Systems and Software, 2010, 83(2): 174-187

[29] Artzi S, Dolby J, Tip F, et al. Practical fault localization for dynamic web applications//Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering. Cape Town, South Africa, 2010: 265-274

[30] Arumuga Nainar P, Liblit B. Adaptive bug isolation//Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering. Cape Town, South Africa, 2010: 255-264

[31] Park S, Vuduc R W, Harrold M J. Falcon: Fault localization in concurrent programs//Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering. Cape Town, South Africa, 2010: 245-254

[32] Naish L, Lee H J, Ramamohanarao K. A model for spectra-based software diagnosis. ACM Transactions on Software Engineering and Methodology, 2011, 20(3): Article No. 11

[33] Wang S, Lo D, Jiang L, et al. Search-based fault localization //Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering. Lawrence, USA, 2011: 556-559

[34] Zhang Z, Chan W K, Tse T H, et al. Non-parametric statistical fault localization. Journal of Systems and Software, 2011, 84(6): 885-905

[35] Bandyopadhyay A. Improving spectrum-based fault localization using proximity-based weighting of test cases//Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering. Lawrence, USA, 2011: 660-664

[36] Gonzalez-Sanchez A, Piel É, Abreu R, et al. Prioritizing tests for software fault diagnosis. Software: Practice and Experience, 2011, 41(10): 1105-1129

[37] Gopinath D, Zaeem R N, Khurshid S. Improving the effectiveness of spectra-based fault localization using specifications//Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. Essen, German, 2012: 40-49

[38] Artzi S, Dolby J, Tip F, et al. Fault localization for dynamic Web applications. IEEE Transactions on Software Engineering, 2012, 38(2): 314-335

[39] De Souza H A, Chaim M L. Adding context to fault localization with integration coverage//Proceedings of the IEEE/ACM 28th International Conference on Automated Software Engineering. Silicon Valley, USA, 2013: 628-633

[40] Yoo S, Harman M, CLark D. Fault localization prioritization: Comparing information-theoretic and coverage-based approaches. ACM Transactions on Software Engineering and Methodology, 2013, 22(3): Article No. 19

[41] Agrawal H, Horgan J R. Dynamic program slicing//Proceedings of the ACM Conference on Programming Language Design and Implementation. New York, USA, 1990: 246-256

[42] Agrawal H, Demillo R A, Spafford E H. Debugging with dynamic slicing and backtracking. Software: Practice and Experience, 1993, 23(6): 589-616

[43] Gyimóthy T, Beszédes Á, Forgács I. An efficient relevant slicing method for debugging//Proceedings of the 7th European Conference on Foundations of Software Engineering and 7th ACM SIGSOFT Symposium on the Foundations of Software Engineering. Berlin Heidelberg, Germany, 1999: 303-321

[44] Zhang X, Gupta R, Zhang Y. Efficient forward computation of dynamic slices using reduced ordered binary decision diagrams//Proceedings of the 26th International Conference on Software Engineering. Scotland, UK, 2004: 502-511

[45] Gupta N, He H, Zhang X, et al. Locating faulty code using failure-inducing chops//Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. California, USA, 2005: 263-272

[46] Zhang X, Gupta N, Gupta R. Pruning dynamic slices with confidence//Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation. Ontario, Canada, 2006: 169-180

[47] Wong W E, Qi Y. Effective program debugging based on execution slices and inter-block data dependency. Journal of Systems and Software, 2006, 79(7): 891-903

[48] Sterling C D, Olsson R A. Automated bug isolation via program chipping. Software: Practice and Experience, 2007, 37(10): 1061-1086

[49] Zhang X, Gupta N, Gupta R. Locating faulty code by multiple points slicing. Software: Practice and Experience, 2007, 37(9): 935-961

[50] Zhang C, Yan D, Zhao J, et al. BPGen: An automated breakpoint generator for debugging//Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering. Cape Town, South Africa, 2010: 271-274

[51] Alves E, Gligoric M, Jagannath V, et al. Fault-localization using dynamic slicing and change impact analysis//Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. Lawrence, USA, 2011: 520-523

[52] Santelices R, Zhang Y, Jiang S, et al. Quantitative program slicing: Separating statements by relevance//Proceedings of the 2013 International Conference on Software Engineering. Piscataway, USA, 2013: 1269-1272

[53] Jiang L, Su Z. Context-aware statistical debugging: From bug predictors to faulty control flow paths//Proceedings of the 22nd IEEE/ACM International Conference on Automated software Engineering. Atlanta, USA, 2007: 184-193

[54] Abreu R, Zoeteweij P, Van Gemund A J C. Spectrum-based multiple fault localization//Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering. Auckland, New Zealand, 2009: 88-99

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

[56] Masri W. Fault localization based on information flow coverage. Software Testing, Verification and Reliability, 2010, 20(2): 121-147

[57] Baah G K, Podgurski A, Harrold M J. Mitigating the confounding effects of program dependences for effective fault localization//Proceedings of the 19th ACM SIGSOFT Symposium on Foundations of Software Engineering and the

13th European Conference on Foundations of Software Engineering. Szeged, Hungary, 2011: 146-156

[58] Qi D, Roychoudhury A, Liang Z, et al. Darwin: An approach to debugging evolving programs. ACM Transactions on Software Engineering and Methodology, 2012, 21(3): Article No. 19

[59] Gore R, Reynolds Jr P F. Reducing confounding bias in predicate-level statistical debugging metrics//Proceedings of the International Conference on Software Engineering. Zürich, Switzerland, 2012: 463-473

[60] Campos J, Abreu R, Fraser G, et al. Entropy-based test generation for improved fault localization//Proceedings of the IEEE/ACM 28th International Conference on Automated Software Engineering. Silicon Valley, USA, 2013: 257-267

[61] Zeller A, Hildebrandt R. Simplifying and isolating failure-inducing input. IEEE Transactions on Software Engineering, 2002, 28(2): 183-200

[62] Cleve H, Zeller A. Locating causes of program failures// Proceedings of the 27th International Conference on Software Engineering. Missouri, USA, 2005: 342-351

[63] Zhang X, Gupta N, Gupta R. Locating faults through automated predicate switching//Proceedings of the 28th International Conference on Software Engineering. Shanghai, China, 2006: 272-281

[64] Dallmeier V, Zeller A, Meyer B. Generating fixes from object behavior anomalies//Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering. Auckland, New Zealand, 2009: 550-554

[65] Jeffrey D, Nagarajan V, Gupta R, et al. Execution suppression: An automated iterative technique for locating memory errors. ACM Transactions on Programming Languages and Systems, 2010, 32(5): Article No. 17

[66] Sumner W N, Zhang X. Memory indexing: Canonicalizing addresses across executions//Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering. Seoul, Korea, 2010: 217-226

[67] Yu K, Lin M, Chen J, et al. Practical isolation of failure-inducing changes for debugging regression faults//Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. Essen, Germany, 2012: 20-29

[68] Sumner W N, Zhang X. Comparative causality: Explaining the differences between executions//Proceedings of the 2013 International Conference on Software Engineering. San Francisco, USA, 2013: 272-281

[69] Jose M, Majumdar R. Cause clue clauses: Error localization using maximum satisfiability//Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation. San Jose, USA, 2011: 437-446

[70] Cho C Y, D'Silva V, Song D. BLITZ: Compositional bounded model checking for real-world programs//Proceedings of the IEEE/ACM 28th International Conference on Automated Software Engineering. Silicon Valley, USA, 2013: 136-146

[71] Hangal S, Lam M S. Tracking down software bugs using automatic anomaly detection//Proceedings of the 24th International Conference on Software Engineering. Vancouver, British, 2002: 291-301

[72] Sagdeo P, Ewalt N, Pal D, et al. Using automatically generated invariants for regression testing and bug localization //Proceedings of the IEEE/ACM 28th International Conference on Automated Software Engineering. Silicon Valley, USA, 2013: 634-639

[73] Murtaza S S, Madhavji N, Gittens M, et al. Diagnosing new faults using mutants and prior faults (NIER track)// Proceedings of the IEEE/ACM 28th International Conference on Software Engineering. Honolulu, USA, 2011: 960-963

[74] Papadakis M, Le Traon Y. Metallaxis-FL: Mutation-based fault localization. Software Testing, Verification and Reliability, 2013

[75] Weiser M. Programmers use slices when debugging. Communications of the ACM, 1982, 25(7): 446-452

[76] Tang De-Gui, Chen Lin, Wang Zi-Yuan, Ding Hui, Zhou Yu-Ming, Xu Bao-Wen. Spectra-based fault localization by increasing marginal weight. Chinese Journal of Computers, 2010, 33(12): 2335-2342(in Chinese)
(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)

[77] Hui Zhan-Wei, Huang Song, Ji Meng-Yu. Research on spectra-based integer bug localization. Chinese Journal of Computers, 2012, 35(10): 2204-2214(in Chinese)
(惠战伟, 黄松, 嵇孟雨. 基于程序特征谱整数溢出错误定位技术研究. 计算机学报, 2012, 35(10): 2204-2214)

[78] Ding Hui, Chen Lin, Qian Ju, Xu Lei, Xu Bao-Wen. Fault localization method using information quantity. Journal of Software, 2013, 24(7): 1484-1494(in Chinese)
(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)

[79] Ye Gang, Yu Dan, Li Zhong-Wen, et al. Fault localization based on Kolmogorov-Smirnov testing model. Journal of Computer Research and Development, 2013, 50(4): 686-699 (in Chinese)
(叶钢, 余丹, 李重文等. 一种基于 Kolmogorov-Smirnov 检验的缺陷定位方法. 计算机研究与发展, 2013, 50(4): 686-699)

[80] Li Wei, Zheng Zheng, Hao Peng, Gao Yi-Chao, Rao Pei-Feng, Gong Cheng. Predicate execution-sequence based fault localization algorithm. Chinese Journal of Computers, 2013, 36(12): 2406-2419(in Chinese)
(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)

[81] Tu Jing-Xuan, Chen Lin, Xu Lei, Lu Hong-Min, Xu Bao-Wen. Fault localization of object-oriented programs with considering class feature. Chinese Journal of Computers, 2013, 36(12): 2420-2428(in Chinese)
(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)

[82] Wen Wan-Zhi, Li Bi-Xin, Sun Xiao-Bing, Liu Cui-Cui. Technique of software fault localization based on hierarchical slicing spectrum. Journal of Software, 2013, 24(5): 977-992 (in Chinese)

(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)

[83] Wen Wan-Zhi, Li Bi-Xin, Sun Xiao-Bao, Qi Shan-Shan. A technique of multiple fault localization based on conditioned execution slicing spectrum. Journal of Computer Research and Development, 2013, 50(5): 1030-1043(in Chinese)
(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)

[84] Xiao Qing, Gong Yun-Zhan, Yang Zhao-Hong, Jin Da-Hai, Wang Ya-Wen. Path sensitive static defect detecting method. Journal of Software, 2010, 21(2): 209-217(in Chinese)
(肖庆，宫云战，杨朝红，金大海，王雅文. 一种路径敏感的静态缺陷检测方法. 软件学报，2010，21(2)：209-217)

[85] Liu Yong-Po, Wu Ji, Jin Mao-Hong, Yang Hai-Yan, Jia Xiao-Xia, Liu Xue-Mei. Experimentation study of BBN-based fault localization. Journal of Computer Research and Development, 2010, 47(4): 707-715(in Chinese)
(柳永坡，吴际，金茂忠，杨海燕，贾晓霞，刘雪梅. 基于贝叶斯统计推理的故障定位实验研究. 计算机研究与发展，2010，47(4)：707-715)

[86] He Jia-Lang, Meng Jin, Zhang Kun, Zhang Hong. A fault propagation-aware program fault location method. Journal of Electronics & Information Technology, 2011, 33(9): 2192-2198(in Chinese)
(何加浪，孟锦，张琨，张宏. 一种故障传播感知的程序故障定位方法. 电子与信息学报，2011，33(9)：2192-2198)

[87] Zhao Lei, Wang Li-Na, Gao Dong-Ming, Zhang Zhen-Yu, Xiong Zuo-Ting. Mining associations to improve the effectiveness of fault localization. Chinese Journal of Computers, 2012, 35(12): 2528-2540(in Chinese)
(赵磊，王丽娜，高东明，张震宇，熊作婷. 基于关联挖掘的软件错误定位方法. 计算机学报，2012，35(12)：2528-2540)

[88] He Tao, Wang Xin-Ming, Zhou Xiao-Cong, Li Wen-Jun, Zhang Zhen-Yu, Cheung Shing-Chi. A software fault localization technique based on program mutations. Chinese Journal of Computers, 2013, 36(11): 2236-2244(in Chinese)
(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)

[89] Xu Bao-Wen, Nie Chang-Hai, Shi Liang, Chen Huo-Wang. A software failure debugging method based on combinatorial design approach for testing. Chinese Journal of Computers, 2006, 29(1): 132-138(in Chinese)
(徐宝文，聂长海，史亮，陈火旺. 一种基于组合测试的软件故障调试方法. 计算机学报，2006，29(1)：132-138)

[90] Zhou Wu-Jie, Zhang De-Ping, Xu Bao-Wen. An adaptive algorithm of locating fault interactions based combinatorial testing. Chinese Journal of Computers, 2011, 34(8): 1509-1518(in Chinese)
(周吴杰，张德平，徐宝文. 基于组合测试的软件故障定位的自适应算法. 计算机学报，2011，34(8)：1509-1518)

[91] He Jia-Lang, Zhang Hong. Application of artificial neural network in software multi-faults location. Journal of Computer Research and Development, 2013, 50(3): 619-625(in Chinese)
(何加浪，张宏. 神经网络在软件多故障定位中的应用研究. 计算机研究与发展，2013，50(3)：619-625)

[92] Zhang Yun-Qian, Zheng Zheng, Ji Xiao-Hui, Zhang Wen-Bo, Zhang Zhen-Yu. Markov model-based effectiveness predicting for software fault localization. Chinese Journal of Computers, 2013, 36(2): 445-456(in Chinese)
(张云乾，郑征，季晓慧，张文博，张震宇. 基于马尔可夫模型的软件错误定位方法. 计算机学报，2013，36(2)：445-456)

[93] Jiang B, Chan W K, Tse T H. On practical adequate test suites for integrated test case prioritization and fault localization //Proceedings of the 11th International Conference on Quality Software. Madrid, Spain, 2011: 21-30

[94] Gonzalez-Sanchez A, Gross H G, van Gemund A J C. Modeling the diagnostic efficiency of regression test suites// Proceedings of the IEEE 4th International Conference on Software Testing, Verification and Validation Workshops (ICSTW). Berlin, Germany, 2011: 634-643

[95] Hao D, Xie T, Zhang L, et al. Test input reduction for result inspection to facilitate fault localization. Automated Software Engineering, 2010, 17(1): 5-31

[96] Masri W, Assi R A. Cleansing test suites from coincidental correctness to enhance fault-localization//Proceedings of the 3rd International Conference on Software Testing, Verification and Validation. Paris, France, 2010: 165-174

[97] Jiang B, Zhang Z, Chan W K, et al. How well does test case prioritization integrate with statistical fault localization? Information and Software Technology, 2012, 54(7): 739-758

[98] Assi R A, Masri W. Identifying failure-correlated dependence chains//Proceedings of the IEEE 4th International Conference on Software Testing, Verification and Validation. Berlin, Germany, 2011: 607-616

[99] Parnin C, Orso A. Are automated debugging techniques actually helping programmers?//Proceedings of the 2011 International Symposium on Software Testing and Analysis. Toronto, Canada, 2011: 199-209

[100] Cheng H, Lo D, Zhou Y, et al. Identifying bug signatures using discriminative graph mining//Proceedings of the 18th International Symposium on Software Testing and Analysis. New York, USA, 2009: 141-152

[101] Debroy V, Wong W E. Insights on fault interference for programs with multiple bugs//Proceedings of the 20th International Symposium on Software Reliability Engineering. Mysuru, Karnataka, 2009: 165-174

[102] DiGiuseppe N, Jones J A. On the influence of multiple faults on coverage-based fault localization//Proceedings of the 2011 International Symposium on Software Testing and Analysis. Toronto, Canada, 2011: 210-220

WANG Ke-Chao, born in 1980, Ph. D. candidate, lecturer. His research interest is software fault localization.

WANG Tian-Tian, born in 1980, Ph. D., associate professor. Her research interests include software debugging and computer aided education.

SU Xiao-Hong, born in 1966, Ph. D., professor. Her research interest is software bugs detection.

MA Pei-Jun, born in 1963, Ph. D., professor. His research interests include software engineering and information fusing.

Background

Despite the achievements made in software development, software faults are still pervasive. Software testing and debugging working together can promote reliable software: testing exposes software faults, and debugging eliminates those faults. To expedite debugging, researchers have proposed automatic fault localization techniques, which use the information provided by test cases executed in the testing phase to deduce a list of program elements that are highly suspected to be a fault, so as to narrow down the search domain of the faults. Automatic fault localization has been a hot spot issue in software development and maintenance in the past fewer years, and various fault localization methods have been studied. However, few practical fault localization tools have been applied in the software industry.

Our paper surveys the state-art of automatic software fault localization. The novelty and contributions of our research are as follows. (1) Our paper focuses on the analysis of the key scientific issues in software fault localization. A "failure-fault localization-comprehension" model has been proposed to introduce the scientific issues. Key scientific problems which need further researched have been analyzed in detail. (2) A thorough survey on the top journals and conferences has been conducted, and the development trend has been statistically analyzed.

This work is supported by the National Natural Science Foundation of China (Nos. 61202092, 61173021), the Research Fund for the Doctoral Program of Higher Education of China (No. 20112302120052), the Research Fund for the Innovative Scholars of Harbin (No. RC2013QN010001), the Youth Academic Backbone Project of Heilongjiang (No. 1254G037) and the Youth Special Project of Heilongjiang Education Science (No. GJD1214038).