文章编号:1671-4598(2019)05-0001-04 DOI:10.16526/j.cnki.11—4762/tp.2019.05.001 中图分类号:TP311.53 文献标识码:A

软件故障定位技术研究综述

李 雷,陈朝晖,李 轶,李经松
(北京控制工程研究所,北京 100081)

摘要:软件故障定位旨在利用程序信息以及测试用例找到导致程序出现故障的语句,以提高程序的安全性与健壮性;首先介绍了常见的单故障定位技术和现有的多故障定位技术,并对比分析了两者之间的差异,然后介绍了常用的测试用例集合,并列举了一些用来测试评价故障定位技术效率的方法,最后对现有的故障定位技术进行总结并提出未来研究方向的展望。

关键词:故障定位;程序分析;技术对比;评价方法

Overview of Software Fault Localization Technology

Li Lei, Chen Zhaohui, Li Yi, Li Jingsong
(Beijing Institute of Control Engineering, Beijing 100081, China)

Abstract: The purpose of software fault location is to use program information and test cases to find the statements that lead to the failure of the program, so as to improve the security and robustness of the program. Firstly introducing the common single fault location technology and the existing multi-fault location technology, and comparing the differences between them, then introducing the common set of test cases, and listing some methods to test and evaluate the efficiency of fault location technology. At last, the paper summarizes the existing fault localization technology and puts forward the prospect of the future research direction.

Keywords: fault localization; program analysis; technical comparison; evaluation method

0 引言

在软件调试过程中,故障定位是最耗时和费力的活动之一[1]。随着当前软件复杂度越来越高,迫切需要提高故障定位的效率,减少开发人员的工作量,降低开发过程的成本开销。软件故障定位是通过对源程序的语义和结构进行分析,结合程序执行情况和结果,帮助测试人员找到程序中的故障位置的过程。在故障定位的研究方法中依据是否需要运行程序的准则可以分为静态方法和动态方法。前者指的是不运行待测程序,利用程序语句之间的依赖关系以及类型约束来研究程序中的故障位置;后者则是结合测试用例运行待定位的程序,对程序执行过程中的信息进行分析,利用程序动态执行信息进行定位。

本文调研了近几年提出的一些代表性的故障定位技术,根据在进行定位时所利用的信息及定位方式进行分类并阐述其原理模型,并对不同方法的优缺点进行了对比。然后对测试用例程序集以及故障定位效率评价方法进行了介绍,最后对未来研究方向提出了展望。

1 故障定位技术

软件故障在IEEE标准中被定义为‘计算机程序中不正确的步骤、过程或者数据定义’,我们研究的故障定位就是要找到引起软件出错的故障所在位置[2]。我们通常将故障描述为由于客观因素或系统里包含一些系统问题使软件在运行时出现得到的实际结果和预期输出存在差异的情形。

在故障定位领域中,我们根据程序中包含的故障数目,将故障定位研究分成单故障定位和多故障定位。一般来说,多故障程序定位消耗的时间复杂度要高于单故障程序的时间复杂度,因此现在主要的研究重点大多是解决单故障程序的定位技术。本文首先介绍一些基于单故障程序定位的一些方法技术,然后列举一些多故障程序定位的算法技术,最后从几个方面对比单故障程序定位和多故障程序定位。

1.1 单故障定位技术

1.1.1 基于覆盖的故障定位技术

该类方法首先将待测程序进行插桩处理,然后执行源程序,得到程序语句在所有测试用例下的覆盖执行信息,最后利用各自的方法对执行信息进行分析计算并进行定位。Wong对语句和测试用例之间的关系提出了这样的假设[3]:一条语句包含故障的概率和它被成功用例执行频数成负相关,和它被失败用例执行次数成正相关。也就是说,某一语句在失败测试用例中出现的频率越高,而在成功执行的用例中出现的频数越少,则该语句含有故障的概率就越大。基于覆盖的故障定位技术的方法不足之处是对测试用例的要求和限制十分严格,如果测试集中的测试用例数目过少,则会导致语句覆盖不全面的情况;如果测试用例数目过多,以至于出现大量冗余用例,那么会使程序语句重复执行,会导致定位的精度降低[4]。另外大部分的错误语句都会既在失败执行用例又在成功执行用例中出现,当对程序语句进行并集或者交集操作时[5],错误语句往往会被去掉,导致得到的集合为空集或者不包含故障语句[6]。

收稿日期:2018-11-01; **修回日期**:2018-11-26。
基金项目:国防基础科研资助项目(JCKY2016203B006)。
作者简介:李 雷(1994-),男,黑龙江哈尔滨人,硕士研究生,主要从事高可信软件技术研究,基于模型的软件设计方向的研究。

1.1.2 基于程序切片的故障定位技术

程序切片技术最初是 Weiser 提出的一种程序分解技术[7]，它是根据一些特定需求而截取的程序语句片段集合。我们把程序输出语句看成兴趣点，将输出结果变量看成兴趣变量。根据构成方式，我们可以将程序切片划分为静态切片和动态切片两种。前者是由包含和兴趣点相关的兴趣变量的所有语句组成，静态切片大多和整个程序的规模一样，因此利用静态切片处理故障定位的效率不高。而动态切片则是在某一具体输入的前提下包含对兴趣点有影响的兴趣变量的语句组成，和前者相比，切片规模更小，精度更高，定位效果有所提高。比较典型的基于程序切片的故障定位方法有下面几种：(1) 李必信[8]等人提出了一些基于程序切片的故障定位方法，并且得到了不错的定位效果。(2) Zhang[9]等人采用基于动态切片的方法实现故障定位技术，主要思路是构造数据切片、全切片以及相关切片 3 种程序切片，然后在这 3 种切片基础上缩小范围，找到程序的故障位置。基于切片的定位技术优点在于可以简化程序，将与故障无关的语句排除掉，降低了程序的复杂度，提高定位的效率。但是在大规模的程序中，我们得到的程序切片规模往往也很庞大，导致效率不能达到我们的预期；而且提取切片的过程有时相对比较繁琐，增加算法的复杂性。

1.1.3 基于程序谱的故障定位技术

程序谱描述的是程序中的语句在执行过程中包含的执行信息与特征，通常用二维数组来表示，程序谱最初是 Reps 等人[10]在研究千年虫问题时提出的。基于程序谱的故障定位技术由于其形式简单、易于操作实现、存储方便而且可以很清晰直白地将程序中所有语句在测试用例下的覆盖执行信息以及测试用例的执行结果表现出来，为统计语句的可疑度值提供了便利的条件。由于在单故障定位方面有着较高的效率，很多学者提出了大量基于程序谱的故障定位方法，比较有代表性的有以下几种方法：(1) 虞凯[11]使用多程序谱模型进行软件故障定位，在单程序谱的基础上增加了依赖对的频谱信息，构造两个程序频谱分别对应数据依赖和控制依赖对应的频谱，然后计算语句总的可疑度，以实现定位操作；(2) He 在其论文中使用构造层次谱的方法来进行故障定位[12]，将整个程序分为函数层和语句层，在不同层次间构造对应的频谱，然后进行定位；(3) Wong 在程序谱的基础上进一步提出构造语句的交叉表[13]，并利用 CrossTab 方法来统计每条语句的可疑度，利用统计学的思想来完成故障定位操作。

但是基于程序谱的定位方法也存在一定的问题。由于程序谱仅考虑了每条语句的执行情况，并没有将上下文语句之间包含的依赖关系进行分析，所以有时会出现定位到的结果并不是真正的错误语句，而是由于错误传递而产生非预期运行结果的相关语句。另外程序谱的构造需要大量测试用例，对测试用例的要求也相对较高。

1.1.4 基于模型的故障定位技术

基于模型的定位技术最初是 Reiter 提出的[14]。他使用了包含（SD，COMPS，OBS）的三元组结构来形式化的定义该方法。其中 SD 代表对源程序的描述，通常采用一阶逻辑语句描述程序内各部分之间的关系及行为；COMPS 代表了程序组件的集合；OBS 则定位为根据输入得到的观测值，是有限一阶语句的子集合[15]。Cleve 和 Zeller 提出的 Delta Debugging 方法[16]首先通过调试环境获取不同测试用例执行过程中的数据；在此基础上，构建各个测试用例执行过程中的程序状态图；接下来采用图挖掘算法识别成功和失败测试用例的程序状态图之间的差异；最后通过这些差异进行故障定位。DeMillo[17]提出了一种用于调试的软件故障分析模型。该方法在模型中定义了错误模式和错误类型，主要的思路是在程序运行过程中发现异常行为，就会根据对应的模式进行分类。根据已建立的故障模型的参考，就可以定位到程序中的故障位置。

该类方法通过代码结构和上下文依赖关系构造故障模型，对程序的动态信息利用的比较充分，定位效果较高[18]。但是该方法也有一定的缺陷，首先建立故障模型需要大量的数学知识，同时模型建立以及推导过程也需要很多的时间。

1.2 多故障定位技术

在实际应用中，程序中往往不止包含一处故障，我们在使用针对单故障程序定位技术处理多故障问题时，会出现定位效果下降的问题。因此我们需要对多故障程序定位进行深入的研究是很有意义的。多故障定位技术的大致思想是将程序中的错误都“孤立”出来，使不同的错误都能够在测试用例中有所体现，然后基于不同的方法找到暴露错误的语句，使这些语句能够很快的被找到，从而实现多故障定位。很多学者提出不同的改进方法来实现多故障定位的需求。

李博[19]在虞凯提出的多程序谱的基础上进行改进，使之可以处理多故障程序定位问题，改进的方向是在得到数据依赖和控制依赖谱后，进行计算时对得到的可疑度不是简单的线性操作，而是先进行预处理后在求出总的可疑度值进行排序定位。而于全江在多故障环境下对程序谱故障定位技术的各个风险评估函数的定位效果进行研究[20]，并提出对部分参数进行修改权重的方法来探究最优参数权重，从而实现多故障定位中良好的定位效果。相比于一般的基于程序谱的方法，文万志在其论文中提出一种基于条件切片谱的方法来实现多故障程序定位技术[21]，其定位精度和效果有所提高。除了利用程序谱的方法之外，一些研究人员提出可以结合数据挖掘的思想来解决多故障定位问题，并且取得了一定的进展。张泽林在其论文[22]中提出利用聚类分析的思想来处理测试用例，将不同故障进行隔离，然后分别进行定位。Cao 采用基于 Chameleon 技术来实现软件多故障定位的要求[23]。该技术先是将测试用例进行约简，然后在对测试用例进行聚类操作，最后将分离开的所有故障单独的进行定位。

多故障程序定位的难点在于程序中多个故障之间存在互相干扰叠加的问题，现有的传统的处理单故障定位技术不

表 1　单故障程序与多故障程序区别

	单故障程序	多故障程序	产生的影响
故障数目	唯一确定，只包含一处	初始时未知，数目不定。	故障数目决定定位迭代的次数。
定位复杂度	找到一处故障后即可完成定位操作，定位次数为一次。	找到一处故障并改正后还需查找其他故障位置，定位次数未知	故障定位消耗的总时间有明显的差异，循环迭代地总次数也有不同。
测试用例分布	所有失败用例均指向相同的故障，故障语句的统计参数值和失败用例数目一致。而其他语句的统计值则小于等于失败用例数目 成功执行用例与失败执行用例数目分布是固定的。	失败用例都有各自对应的故障，每一条故障语句的统计参数值一般都小于所有失败用例数目。而非故障语句的统计值则可能会大于真正故障语句的统计值。 成功执行用例与失败执行用例的分布受程序中存在的故障数目影响。	在多故障环境下，根据统计公式得到的所有语句的可疑度结果中，非故障语句的值可能会比真正故障语句的结果要大，从而导致其排名靠前，先被校验，导致定位效率降低。 当在单故障程序中注入新的故障后，测试用例的分布会发生变化，一部分成功执行的测试用例会产生错误的结果而变成失败用例，这样会导致一些语句的可疑度值发生变化，从而使定效率降低。

能很好的解决这个现象，因此导致定位效果降低。很多学者提出的上述方法的本质目的都是希望和单故障定位一样，可以单独的处理每一个错误，从而提高多故障定位技术的效率。

1.3　单故障与多故障技术对比

单故障程序定位和多故障程序定位技术的差异在下表 1 中已经归纳总结出。现阶段主要的研究重点集中在单故障程序定位，而后者由于其复杂度要比单故障定位高很多，因此提出的相应的定位技术方法比较少。

总的来说，两者的差异主要是体现在对测试用例尤其是失败执行用例的统计处理方面，故障数目的差异因而导致对应的测试用例也随之不同，而就是这种不同造成在多故障定位中出现互相干扰影响的现象，进而导致定位效率下降，定位开销增大。

2　测试用例程序集、评价方法

本章主要从以下两个方面进行论述，首先介绍的是在软件故障定位技术中使用的评测数据集，其中包含一些故障程序以及相应的测试用例，其中我们主要以 Siemens 程序集为例进行说明。其次我们论述的是在利用 Siemens 程序集的基础上，对使用的故障定位方法给出的测试评价方法的讨论，列出了一些常用的测试评价方法。

2.1　测试用例程序集

在当前故障定位技术研究领域中，应用广泛的程序集是 Siemens 程序集，该程序集由 7 个程序组成，程序结构多样性，通用性强，每个程序的代码行数不超过 600 行，主要是基于 C 语言程序定位，后续加入针对 Java、c＋＋等常用语言的程序定位。该程序集是由西门子公司开发的，全部程序集可以从相应的网站中下载[24]。大部分程序中只包含一个故障，每个版本都对应大量测试用例。表 2 是该程序集的概要信息，其中包含程序名称、故障版本数、代码行数、可执行代码行数以及测试用例数[25]。

表 1 中 Siemens 套件包含 7 个程序，每个程序都有正确源码和若干带有故障的源码，一共有 132 个故障版本。print _ tokens 和 print _ tokens2 是一些词法分析的程序，replace 是一些模式转换的程序，schedule 和 schedule2 是和调度功能相关的程序，tcas 实现高度区分功能，tot－info 实现信息估量功能。套件中程序的目录结构都是一致的，具体目录结构如下：

表 2　Siemens 测试套件概要

程序名称	故障版本数	代码行数	可执行代码行数	测试用例数
tcas	41	173	55	1608
schedule	10	412	121	2650
schedule2	32	307	112	2710
print_tokens	7	565	175	4130
print_tokens2	10	510	178	4115
replace	9	563	216	5542
tot_info	23	406	113	1052

（1）source 目录：存放的是程序正确版本的源代码。

（2）inputs 目录：存放的是输入数据，但不是所有程序的该文件夹都有内容。

（3）versions 目录：存放的是程序各个故障版本的源代码。

（4）scripts 目录：存放的是测试脚本，可以运行测试用例，得到的结果存放在对应输出目录中。

（5）outputs 目录：存放的是正确版本的输出结果。

（6）newoutputs 目录：存放的是故障版本的输出结果。

（7）testplans. alt 目录：存放的是所有的测试套件，其中包含 universe 文件，该文件包含一个测试用例集合，存放所有测试用例。

2.2　算法评价方法

软件故障定位技术的目的是尽可能通过自动化过程实现故障的发现，减少测试人员的工作量。软件定位结果的好坏可以通过发现故障过程需要检查的语句数量进行评价。需要检测的语句数越少，说明效果越好。

Jones 和 Harrold[26] 提出了一种度量软件故障定位有效性和效率的方法，使用发现故障所需要检查的语句数进行评价。评价标准如公式（1）所示：

$$score = 1 - \frac{|\ tested\_entries\ |}{|\ executed\_entries\ |} \tag{1}$$

score 中各个变量的含义为：$|tested\_entries|$ 表示找到软件故障需要检查的语句数目；$|executed\_entries|$ 表示执行测试用例中所有可执行语句数目。在定位过程中，不需要检查的语句数目占所有语句的比例越大，说明定位效果越好。

在一种基于分块切片的故障定位技术中，采用约减率 RR（reduction ratio）来度量分块切片的约减度[27]，其评价方法如公式（2）所示：

$$RR = \frac{|BlockBWSlice|}{|Program|} \tag{2}$$

上述公式中，*BlockBWSlice* 代表的是根据兴趣块 B 和兴趣变量集 V 逆向遍历系统依赖图而得到的静态切片。在定位过程中，*BlockBWSlice* 的规模越小，说明定位效果越好。

Renieres 和 Reiss[28-29]两人提出了利用程序依赖图来度量算法效率的方法。他们首先给出了程序依赖图（program dependence graph，PDG）中任一节点的 k 阶依赖集（DS_k）的定义，其含义是与该节点的距离为 k 的所有节点集合。$DS_*(R)$ 表示可能包含故障的最小节点集合，$T=1-\frac{|DS_*(R)|}{|PDG|}$ 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；$|PDG|$ 是 *PDG* 中节点个数。该方法也被称为 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）中所有节点。由于故障集合 DS_*（R）中的故障的识别率有很大的差异，因此 T－Score 的结果可能会低于实际值。

Zhang 等人在衡量非参数检验的故障定位技术的有效性时，提出了基于谓词排名的评分标准，也称 P－score[30]。基于谓词可疑度降序排序给出故障报告，首先标记程序中和故障最接近的谓词为故障最相关谓词，然后定义错误定位技术的得分为检查到故障最相关谓词时，已检查的谓词占列表上所有谓词的百分比[31]。

以上这些衡量故障定位效果的方法大多在单故障程序定位中有着很好的效果，但是在多故障领域，这些衡量标准评价效果一般。因此，Jones 在论文[32]提出一种基于其提出的标准基础上的延伸公式，即：

$$Expense = \frac{rank\ of\ fault}{size\ of\ program} \tag{3}$$

$$D = \sum_{i=1}^{|faults|} Expense_i \tag{4}$$

针对多故障定位的分析，Jones 提出了 4 的评价方法，利用发现并修复软件中所有故障需要检查的语句数目占程序所有语句比例的总和来表示算法的效率，但是对于发现并修复软件的时间成本没有考虑。Jones 在 Score 的基础上扩展提出另外一种评价标准公式：

$$F = \sum_{i=1}^{|iterations|} \max\{Expense_f \mid f\ is\ fault\ subtask\ at\ iteration\ i\} \tag{5}$$

该标准利用发现并修复所有故障所需要的最小时间成本来表示定位的效率。

一般来说，大多数故障定位技术都是采用 Jones 提出的 score 法来评测方法本身的优劣，通过比较再找到所有故障前需要查找的语句数目来评价所使用的定位方法的有效性。而且评价的结果相对准确可靠，对提高定位方法的有效性提供了很大的帮助。

3 对未来研究方向的展望

针对软件故障定位问题的研究已经取得了一定的进展，研究者提出了不同的方法来实现故障定位。仍然存在一些问题需要进一步研究，未来的工作方向可以从下面几个角度入手：

1）大多数研究人员关注的重点是单故障程序，待定位的程序中只包含一处错误，而实际应用中，程序往往不只有一处故障，而且故障类型也有很大的差异，因此可以将关注点更多地聚焦在对多故障程序定位的研究上，对现有技术进行改进研究，来满足多故障程序定位的需求。

2）在故障定位的方法中，测试用例的选取也是十分重要，如果测试用例数目太少可能会导致不能覆盖到所有执行语句；而如果程序中有大量冗余的测试用例，则会导致可执行语句被大量重复执行，会对语句信息统计时造成干扰。因此在后续研究中，可以选择对测试用例进行优化，以提高故障定位的效率。

3）目前在故障定位领域中，都是以 Siemens 程序集作为实验的样本，并没有在其他平台进行实验，该程序集代码量相对较小，不能涵盖所有可能的故障。因此在后续研究中，可以选择更大的程序集进行实验研究，对故障种类也可以进行扩展补充。

4 结束语

本文首先分别从单故障和多故障两个角度综述了现有的故障定位技术，然后对这两个方向进行了对比，提出两者的差异。最后给出了测试用例程序集和算法评价方法，并对故障定位研究方向进行了展望。

参考文献：

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

[2] IEEE. IEEE standard glossary of software engineering terminology [S]. New York：IEEE，1990.

[3] Wong，Debroy V，Gao R，et al. The DStar method for effective software fault localization [J]. IEEE Transactions on Reliability，2014，63 (1)：290－308.

[4] Jiang B，Chan W K，Tsee T H. On practical adequate test suites for integrated test case prioritization and fault localization [A]. International Conference on Quality Software [C]. IEEE Computer Society，2011：21－30.

[5] Horgan J，London S，Agrawal，W. Wong. Fault localization using execution slices and dataflow tests [J]. Proceedings of IEEE Software Reliability Engineering. 1995：143－151.

（下转第 21 页）