面向软件错误定位与理解的测试执行离散特征筛选

刘梦冷[1] 杨小双[1] 赵 磊[1,2] 王丽娜[1,2]
(武汉大学计算机学院 武汉 430072)[1]
(武汉大学空天信息安全与可信计算教育部重点实验室 武汉 430072)[2]

摘 要 软件错误定位与错误理解是软件调试过程中的重要步骤,然而调试人员利用基于覆盖分析的软件错误定位获取的可疑度,从高到低静态分析每条程序语句的检查方式,与实际软件调试过程并不相符。为了能够筛选更有助于理解错误根源的测试执行,尤其是致使程序失效的失效执行,帮助调试人员进行动态差异化分析,针对失效执行提出基于高可疑度覆盖率、揭示错误潜力和覆盖语句可疑度离散特征的3种优先级策略,针对成功执行提出加权余弦相似度匹配策略。通过将3种失效执行优先级策略与随机选择在常用错误定位技术中进行实验对比,验证了基于覆盖语句可疑度离散特征的失效执行筛选策略能够对筛选前后的错误理解工作量变化产生更强的积极影响和更弱的消极影响,并能够在相同工作量下理解更多的错误,进而更有助于将错误定位结果应用于错误根源的理解。

关键词 软件错误定位,错误理解,覆盖分析,测试执行优先级,离散特征
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.3.034

Discrete Characteristic-based Test Execution Selection for Software Fault Localization and Understanding

LIU Meng-leng[1] YANG Xiao-shuang[1] ZHAO Lei[1,2] WANG Li-na[1,2]
(School of Computer, Wuhan University, Wuhan 430072, China)[1]
(Key Laboratory of Aerospace Information Security and Trust Computing, Ministry of Education, Wuhan University, Wuhan 430072, China)[2]

Abstract Software fault localization and understanding are key steps in software debugging, while the way that debuggers use suspicious scores calculated by coverage-based fault localization from high to low to analysis every program statement does not correspond to real debugging. To select test executions which are more helpful for fault understanding, especially the failed executions which lead to program failure, and help debuggers conduct differentiation analysis dynamically, three prioritization strategies which are based on coverage rate of high suspicious scores, fault exposing potential and discrete characteristic of suspicious scores of covered statements were presented respectively for failed executions, and weighted cosine similarity matching was presented for passed executions. These failed test execution prioritization strategies and random sorting were compared in general fault localization techniques. And results show that the selection based on discrete characteristic of suspicious scores of statements covered by test executions has stronger positive influence and weaker negative influence on the change of fault understanding expense before and after selection, can understand more faults under same expense, and is more helpful for the application of fault localization results to fault understanding.

Keywords Software fault localization, Fault understanding, Coverage analysis, Test execution prioritization, Discrete characteristic

1 引言

软件是信息系统必不可少的组成部分,然而软件缺陷的存在严重影响信息系统的安全性与可靠性[1]。大部分软件缺陷是由于开发人员在软件设计开发过程中的逻辑不全面、设计不完整甚至编码失误所导致的软件源代码错误[2]。软件调试是定位并排除这些软件缺陷的常用手段,也是软件开发过程中繁琐且易出错的过程,具有很高的自动化需求[3]。它一般包括3个方面:(1)错误定位;(2)错误理解及修复;(3)修复后的回归测试[4]。目前已有大量工作[5]针对基于测试的软件调试自动化进行研究,其中基于覆盖分析的软件错误定位技术(coverage-based fault localization, CBFL)[6-8]按照测试用例执行后的状态,将所有原始测试用例划分为失效用例和成功用例,统计它们包含的大量覆盖信息,通过计算每条程序语句

到稿日期:2015-03-15 返修日期:2015-06-14 本文受国家自然科学基金(61303213,61373169),国家高技术研究发展计划(863计划)(2015AA016004),信息保障技术重点实验室开放基金(KJ-13-104)资助。
刘梦冷(1991-),男,硕士,主要研究方向为软件调试技术、软件安全测试,E-mail:coldreamlau@163.com;杨小双(1990-),女,硕士,主要研究方向为软件错误定位;赵 磊(1985-),男,博士,主要研究方向为软件调试技术、安全测试,E-mail:leizhao@whu.edu.cn;王丽娜(1964-),女,博士,教授,博士生导师,主要研究方向为网络安全、可信计算。

的可疑度(suspicious score),以此为指标降序排序,区分可能导致程序失效的危险程度,辅助调试人员优先审查更加可疑的代码语句,尽早发现错误存在的位置。研究表明,CBFL 技术不管在有效性还是可实施性方面都比其它技术具有更好的效果[9]。

然而,基于覆盖分析的软件错误定位仅仅返回了一份根据导致程序失效的可疑度大小降序排序的程序语句列表,并没有其他附加信息[10]。调试人员很难仅凭观察列表中静态的可疑代码便找出错误根源[11]。无法直接将错误可疑度应用于错误理解的主要原因,是没有考虑程序执行上下文的影响。Bettenburg[12]也指出,调试人员仅仅观察独立的代码语句,依靠自身编程经验很难理解错误触发和程序失效的原因,应当动态执行一些测试用例,观察和比较程序运行时状态和预期状态间的上下文差异。

为了得到有助于错误理解的执行上下文差异,需要对测试执行构建差异化模型[13-15],即通过对比失效执行和成功执行之间的特征差异,找到导致两次执行结果不同的关键差异状态。并且为了提高效率,应当采用最小差异化方法,缩小差异状态的比较范围,使得选择出的成功执行与失效执行有较高的相似度,仅有一小部分状态存在差异,进而帮助调试人员仅利用这一部分差异就能够对错误产生的原因进行分析。

同时,导致程序失效的失效执行是理解错误产生原因的关键因素。在随机调试过程中,如果筛选的失效执行能够优先覆盖利用错误定位方法获取的具有较高可疑度的程序语句,便可对这些语句进行诊断和分析,相比盲目的观察点选取,以错误定位为导向的调试会更加高效。为了弥补动态分析时执行上下文的缺失,以错误定位结果为导向,应当选择更适于理解错误根源的失效执行。其中结合测试执行的优先级排序(test case prioritization)[16]是常用策略,在软件调试中能够依据设定的优先级评价标准,衡量测试执行价值,按照相应的约简条件筛选满足某一特定目标的测试执行子集。

本文关注在有效利用基于覆盖分析的错误定位结果的情况下,以何种方式筛选更适合应用于理解错误根源的失效执行,以及能够与失效执行构成执行上下文最小差异的成功执行,进而在动态分析过程中为程序错误理解和失效场景判定弥补执行上下文的缺失。因此,本文提出的测试执行筛选策略主要包括两个部分:失效执行的优先级筛选和成功执行的相似度匹配。其中失效执行筛选是该策略的重中之重,提出 3 种面向错误定位与理解的启发式失效执行优先级策略,而在成功执行的匹配中提出了也能结合测试执行覆盖信息的加权余弦相似度算法。结合成功执行的匹配,通过将 3 种失效执行优先级策略与随机选择策略进行实验对比,力求发现更有助于理解错误根源的筛选策略。

本文第 2 节对提出的面向错误定位与理解的 3 种失效执行优先级筛选和加权余弦相似度的成功执行匹配进行介绍;第 3 节给出基于提出的策略进行的对比实验和结果分析;第 4 节介绍目前针对错误理解和诊断的相关工作;最后总结全文。

2 面向软件错误定位与理解的测试执行筛选策略

利用有效错误定位的结果,从所有测试执行中筛选有利于错误理解的失效和成功执行,可以在调试过程中产生执行上下文差异,帮助调试人员更好地理解错误根源和程序失效场景。本文设计的面向软件错误定位与理解的测试执行筛选策略总体流程如图 1 所示。在将原始测试用例经过测试、区分失效和成功用例,并获得错误定位结果后,首先依据错误定位结果对失效执行进行优先级筛选,然后为筛选后的失效执行匹配执行路径相似的成功执行,构成失效-成功执行对集合。最终希望以这样的执行对集合帮助调试人员理解错误。

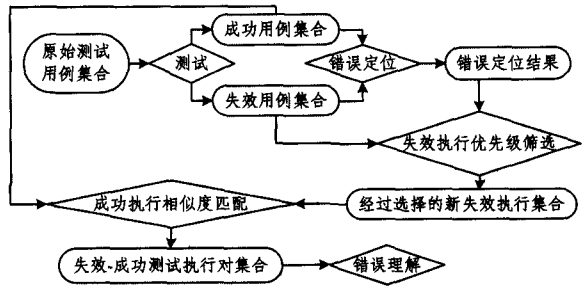

图 1 测试执行筛选策略总体流程

对于失效执行的优先级筛选,借鉴回归测试[17-19]的测试用例优先级策略,依据高可疑度程序语句的覆盖情况和测试执行揭示错误的潜力提出两种启发式失效执行优先级策略。然后依据测试执行覆盖程序语句可疑度的离散分布情况,提出一种基于语句可疑度离散特征的失效执行优先级策略。对于成功执行相似度匹配,为能充分利用错误定位结果,提出将语句可疑度视为权值的加权余弦相似度匹配策略。

2.1 失效执行的优先级筛选

尽管回归测试位于软件演化过程,目标是提高错误检测的效率,以尽可能少的用例触发更多的错误[20],而软件错误定位与理解中的执行优先级筛选应用于执行结果之后,目标是利用测试执行理解和修复错误;且回归测试通常忽视成功用例[21],而软件调试中的错误理解在很多情况下需要借助路径相似的成功执行与失效执行进行差异比较[22-24],但是回归测试的测试用例优先级策略却可以给软件错误定位与理解一些启发。

2.1.1 基于高可疑度覆盖率的失效执行优先级筛选

Rothermel 等人[17]在研究回归测试的测试用例筛选时,对测试用例优先级排序的定义为:

已知:测试用例集 T;PT 是 T 的一个全排列集;f 为从 PT 映射到实数的排序目标函数。

问题:找出 $T' \in PT$,使得 $(\forall T'')(T'' \in PT)(T'' \neq T')[f(T') \geqslant f(T'')]$。

他们认为,测试用例优先级排序包含多种启发式策略,其中最常见的一种是依据测试用例覆盖程序实体的总数进行排序,并认为覆盖的数目越多,测试用例的优先级越高,越值得被挑选出来进行回归测试。

而对于基于覆盖分析的软件错误定位与理解,静态的测试用例已经按照最终执行状态划分为失效用例和成功用例,常用错误定位算法(如 Tarantula[25])能够利用程序实体在所有失效用例和成功用例中的覆盖情况计算可疑度,以此作为区分其影响失效严重程度的重要依据,并且认为可疑度更高的程序实体导致程序失效的可能性更大。这种情况下,以程序语句作为研究的对象实体,考虑动态的测试执行对高可疑度程序语句的覆盖情况更具有实际意义。

据此,本文提出一种基于高可疑度语句覆盖率的失效执

行优先级策略(其中高可疑度设定为按照错误可疑度降序排列后,其排序在所有互不相同的语句可疑度总数前20%的可疑度)。定义高可疑度覆盖率 $high\_cover$ 为测试执行 t 覆盖高可疑度语句的数量与程序所有可执行语句总数的比值,如式(1)所示:

$$high\_cover(t)=\frac{number\ of\ high\ suspicious\ statements}{number\ of\ executable\ statements} \tag{1}$$

获取每个失效执行的高可疑度覆盖率后,按照覆盖率大小对失效执行降序排序,覆盖率越高则优先级越高。这种策略着重关注了高可疑度语句在程序失效过程中可能产生的重要影响,以最直观的数量统计方式比较哪条失效执行包含的高可疑度语句数目更多,并认为包含高可疑度语句数目越多的失效执行越应当被优先挑选出来用于错误根源的理解。

2.1.2 基于揭示错误潜力的失效执行优先级筛选

Elbaum 等人[18]在研究回归测试时发现,基于程序实体覆盖情况的优先级策略仅仅考虑了测试用例是否覆盖程序中的语句或分支,然而测试用例揭示错误存在的能力不仅依赖于它是否覆盖了这条错误语句,还依赖于错误语句由于这一测试用例而导致失效结果的概率。因此他们利用 PIE 模型[26]和变异分析近似计算测试用例揭示错误的潜力(fault-exposing-potential,FEP),以 FEP 作为优先级衡量指标,对所有测试用例降序排序。以语句作为研究对象,给定程序 P 和测试集合 T,对于每一个测试用例 $t\in T$,定义程序语句 $s\in P$ 的变异值 ms(mutation score)如式(2)所示:

$$ms(s,t)=\frac{exposed(s,t)}{mutants(s)} \tag{2}$$

其中,$exposed(s,t)$ 表示测试用例 t 揭示的语句 s 的变异体数,$mutants(s)$ 表示语句 s 的变异体总数。

之后对于每一个测试用例 $t\in T$,其 FEP 为所有语句 ms 的累加和,如式(3)所示:

$$FEP(t)=\sum_{s\in P} ms(s,t) \tag{3}$$

在计算测试用例的 FEP 并降序排序后,他们认为 FEP 值越大,测试用例优先级越高,从整体上来看其揭示错误的潜力越强,也越值得被挑选出来进行回归测试。

将这种衡量揭示错误潜力的策略借鉴到面向错误定位和理解的失效执行优先级排序中,每条程序语句的 ms 在错误定位中即对应按照某一错误定位方法计算所得的语句可疑度 $suspicious(s)$,失效执行 t 揭示错误的潜力 FEP 就等于所有程序语句可疑度的累加和,如式(4)所示:

$$FEP(t)=\sum_{s\in P} suspicious(s) \tag{4}$$

失效执行的 FEP 越大,优先级越高,越值得被挑选来进行错误定位和理解。这种策略综合了测试执行覆盖的所有语句可疑度对其引起程序失效的可能性。也可以理解为,在统计失效执行覆盖所有程序语句总数的基础上,将可疑度作为一种权值,赋权给每条程序语句。如此程序语句因可疑度大小不同被区分开来,不再具有同等地位,进而帮助调试人员进行更加有效的错误定位和理解。

2.1.3 基于可疑度离散特征的失效执行优先级筛选

基于高可疑度覆盖率和揭示错误潜力的失效执行优先级策略,都利用了可疑度这一重要参数。依据软件失效的原理,发生功能性故障时,失效执行的路径一定经过了程序中与该故障相关的错误语句[27]。高可疑度语句可被认为是更可能与软件功能性故障相关的语句,反之低可疑度语句引起程序功能性故障的可能性会较低。这时,一条失效执行包含的程序语句可疑度变化幅度越大,说明其覆盖的可疑语句波动情况越复杂,进而突显出高可疑度语句和低可疑度语句的相互作用及对测试执行可疑性的重要影响。因此依据每条程序语句的可疑度离散分布情况,统计出的失效执行的可疑度离散特征应当能够成为一种区分不同失效执行的关键因素。

1)离散度

统计学中,数据的样本标准差通常用于描述该组数据的离散程度,能够从一定程度上反映这组数据中不同数值的分布差异。样本标准差越大,说明这组数据数值分布波动性越大。标准差 σ 是总体各单位标志值与算术平均数的离差平方的算术平均数的平方根,又称"均方差",如式(5)所示:

$$\sigma=\sqrt{\frac{\sum(x-\bar{x})^2}{N-1}} \tag{5}$$

其中,N 表示数据个数,x 为各单位标志值,$\bar{x}$ 为各单位标志值的算数平均值。

虽然一般情况下标准差能够反映一组数据的离散分布情况,却存在很大的局限性。标准差以算术平均值为中心反映一个总体全部标志值的离散程度,是一个绝对指标,当用其对同一总体的不同时期或不同总体进行对比时缺乏可比性。一般情况下,标准差只能用于比较不同总体同一标志(同一水平)的标志值变异程度。只有平均水平相同时,才能依据标准差判断数据分布的波动情况。而平均水平不同时,就需要采用一种新的指标衡量数据的离散分布,即离散度。

离散度,即变异系数[28](Coefficient of Variance,CV),也称为差异系数、离散系数。它是一组数据标准差与算术平均值的比值,如式(6)所示[29]:

$$CV=\frac{\sigma}{\bar{x}} \tag{6}$$

由于离散度是两个有相同计量单位的量 σ 和 $\bar{x}$ 的商,因此是一个无名数,表现为相对数形式。它消除了不同平均水平对于衡量数据离散程度过程产生的影响,不受数据平均值的条件约束,比标准差的应用更加广泛。一组数据数值分布的离散度越大,说明数值间的差异变化越大,波动性越强。

2)基于可疑度离散度的失效执行优先级筛选

由于每条失效执行覆盖程序语句的情况存在差异,分析覆盖语句可疑度的离散程度时,单纯采用可疑度标准差就会受到所有覆盖语句可疑度平均值不同而产生的影响。这种情况下,应当利用比标准差应用更为广泛的离散程度衡量指标,即覆盖语句可疑度的离散度,消除平均值不同产生的影响。本文中测试执行的离散度(dispersion),即为覆盖所有语句的可疑度标准差和平均值的比值。测试执行 t_i 覆盖的所有语句可疑度的平均值 $avg\_suspicious(s)$ 定义如下:

定义 1(测试执行的可疑度平均值) 给定程序 P,测试执行 t 覆盖所有程序 P 中语句 s_j($1\leqslant j\leqslant totalnum$)的可疑度平均值 $avg\_suspicious(s)$ 的计算如式(7)所示:

$$avg\_suspicious(s)=\frac{\sum_{j=1}^{totalnum} suspicious(s_j)}{totalnum} \tag{7}$$

其中,$totalnum$ 是程序 P 中所有可执行语句的总数,$suspi$-

$cious(s_j)$是测试执行 t 覆盖程序 P 中每条语句 s_j($1 \leqslant j \leqslant totalnum$)的可疑度。

定义 2(测试执行的可疑度标准差) 给定程序 P,测试执行 t 覆盖所有程序 P 中语句 s_j($1 \leqslant j \leqslant totalnum$)的可疑度标准差 $std_deviation(t)$的计算如式(8)所示:

$$std\_deviation(t)=\sqrt{\frac{\sum_{j=1}^{totalnum}(suspicious(s_j)-avg\_suspicious(s))^2}{totalnum-1}} \tag{8}$$

定义 3(测试执行的可疑度离散度) 给定程序 P,测试执行 t 覆盖所有程序 P 中语句 s_j($1 \leqslant j \leqslant totalnum$)的可疑度离散度 $dispersion(t)$的计算如式(9)所示:

$$dispersion(t)=\frac{std\_deviation(t)}{avg\_suspicious(s)} \tag{9}$$

失效执行的语句可疑度离散度越大,表明该执行覆盖语句的可疑度分布情况越复杂。这种语句可疑度波动情况越紊乱的失效执行就越值得调试人员怀疑,应当被选择出来进行更加有效的错误定位和错误理解。

2.2 成功执行的相似度匹配

程序执行上下文的差异化分析需要借助失效执行和成功执行的差异比较才能分析错误产生的可能原因。为了缩小比较范围,应当尽可能保证选择的成功执行与失效执行具有较高相似度,使得致使程序失效的差异状态最大化体现。

在众多相似度计算方法中,余弦相似度[30]保持了许多优越的性质。Lee[31]和 Omiecinski[32]等证明,余弦相似度不受零事务的影响,且余弦相似度算法能够应对海量高维度数据[33],因此在度量多维度数据相似性方面十分有效。

为了能够利用有效错误定位的结果,本文将每条失效执行和成功执行均视为不同的语句覆盖向量,把可疑度作为向量中每个分量的权值,利用加权余弦相似度算法(Weighted Cosine Similarity)为失效执行匹配成功执行。

定义 4(加权语句覆盖向量) 假设$\langle s_1,\cdots,s_n\rangle$是程序 P 的语句集,测试执行 t 的加权语句覆盖向量如式(10)所示:

$$cover\_w(t)=\langle c_1,\cdots,c_n\rangle \tag{10}$$

其中,

$$c_i=\begin{cases} suspicious(s_i), & t \text{ 的执行路径覆盖了语句 } s_i \text{ 的可疑度} \\ 0, & t \text{ 的执行路径没有覆盖语句 } s_i \end{cases}$$

定义 5(加权语句覆盖向量的余弦相似度) 给定失效执行 tf,成功执行 tp。两者间加权的向量余弦相似度为其加权语句覆盖向量间的夹角余弦,如式(11)所示:

$$Cos\_Sim(cover\_w(tf),cover\_w(tp))=\frac{\sum_{i=1}^{n}(c_{tf_i}\times c_{tp_i})}{\sqrt{\sum_{i=1}^{n}(c_{tf_i}\times c_{tf_i})}\times\sqrt{\sum_{i=1}^{n}(c_{tp_i}\times c_{tp_i})}} \tag{11}$$

其中,c_{tf_i}是失效执行 tf 覆盖的每条程序语句的可疑度,c_{tp_i}是成功执行 tp 覆盖的每条程序语句的可疑度。

加权余弦相似度越高,表明成功执行和失效执行越相似,对比产生的差异越小。通过计算完整的成功执行集合中每条成功执行与之前筛选出的每条失效执行的相似度,选取相似度最高的成功执行,与对应的失效执行构成一条"失效-成功测试执行对"。然后利用这样的失效-成功测试执行对,理解错误根源和程序失效场景。

3 实验验证

3.1 实验设置

为了结合成功执行的相似度匹配,对提出的 3 种失效执行优先级策略与软件调试中常用的随机选择进行实验对比,我们将开发的原型应用于软件调试实验中广泛使用的 3 个 Unix 程序(flex,grep,gzip)和 SIR[34]提供的 Siemens 基准测试包,其样本信息如表 1 所列。

表 1 Unix 的 3 个程序及 Siemens 基准测试包程序样本信息表

实验样本		错误版本数	测试执行个数	样本描述
Unix	flex	43	567	lexical parser
	grep	19	809	text processor
	gzip	17	213	compressor
Siemens	print_tokens	7	4130	lexical analyzer
	print_tokens2	10	4115	lexical analyzer
	replace	29	5542	pattern replacement
	schedule	9	2650	priority scheduler
	schedule2	9	2650	priority scheduler
	tcas	40	1578	altitude separation
	tot_info	23	1054	information measure
	合计	206		

同时,本文实验将不同的失效执行优先级策略应用于软件错误定位领域影响较为广泛的 4 种基于覆盖分析的错误定位技术 Tarantula[25]、CBI[35]、Jaccard[36]和 Ochiai[36]中,其有关算法如表 2 所列。

表 2 4 种常用错误定位技术

错误定位技术	可疑度算法
Tarantula	$\frac{\frac{failed(s)}{totalfailed}}{\frac{failed(s)}{totalfailed}+\frac{passed(s)}{totalpassed}}$
CBI	$\frac{failed(s)}{passed(s)+failed(s)}$
Jaccard	$\frac{failed(s)}{totalfailed+passed(s)}$
Ochiai	$\frac{failed(s)}{\sqrt{totalfailed\times(failed(s)+passed(s))}}$

其中,s 代表语句,$failed(s)$代表执行该条语句的失效用例个数,$totalfailed$ 代表所有失效用例的总个数;$passed(s)$代表执行该条语句的成功用例个数,$totalpassed$ 代表所有成功用例的总个数。

3.2 评估指标

衡量测试执行筛选是否有助于错误根源理解,多数情况下依赖用户自身体验。为了将这种抽象标准具体量化,参考 Yu 等人[37]在研究错误定位有效性时对调试人员找到错误语句需要查询的程序语句工作量的定义,可以重新借助错误定位,反馈式地考察测试执行选择对调试人员有效理解错误根源所需工作量产生的影响。有效理解错误所需工作量(Expense)是错误语句在错误理解过程中的可疑度排序($rank$)和程序可执行语句行数的比值,如式(12)所示:

$$Expense=\frac{rank\ of\ faulty\ statements}{number\ of\ executable\ statements}\times 100\% \tag{12}$$

用 $Expense\_increase$ 定义测试执行选择对错误根源理解产生的工作量变化,它是分别应用选择后的测试执行集合和原测试执行集合在错误理解过程中的 $Expense$ 的差值。$Expense\_increase$ 的计算如式(13)所示:

$$Expense\_increase = Expense(reduced) - Expense(unreduced) \quad (13)$$

如果 *Expense_increase* 为负数，就代表对于错误语句，进行错误理解时使用筛选出的测试执行集合得到的排序，要比原有测试执行集合得到的错误语句排序靠前，即利用经过筛选的失效-成功测试执行集合，调试人员只需查找较少的代码就能找到并理解错误。因此选择出的测试执行提高了错误理解的效率，对错误理解产生了积极影响。并且负数的绝对值越大，调试人员需要花费的工作量减少得越多。如果 *Expense_increase* 为正数，就代表经过筛选的测试执行降低了错误理解的效率，对错误理解产生了消极影响。

同时 Yu 等人[37]也指出，测试用例筛选与错误定位所需工作量之间存在权衡关系。他们对基于语句和基于向量的两种不同测试用例筛选策略进行研究，发现基于语句的策略虽然能够较大程度筛选用例，却会增加错误定位所需工作量；而基于向量的策略尽管未能更大程度筛选测试用例集合，但对错误定位工作量的影响却可忽略不计。由于本文采用的错误理解工作量评估指标参考错误定位工作量，其与测试执行的筛选间也存在这样的权衡关系，因此较好的测试执行筛选，应当能够维持与错误理解工作量间的权衡，在弱化因筛选测试执行而对工作量产生消极影响的同时，还能比其他策略对错误理解工作量产生更加积极的影响。

此外，不同的失效执行优先级策略对错误根源理解的影响还体现在筛选测试执行集合后，对调试人员而言，进行错误理解时所需的工作量规模上。相同工作量下能够理解的错误数量越多，失效执行优先级策略越有效。我们统计了不同 *Expense(reduced)* 指标下，累计能够有效定位并理解错误根源的错误版本数量占错误版本总数的比例，进而研究不同的优先级策略在测试执行筛选后对错误理解结果的影响。

3.3 实验流程

将基于测试执行可疑度离散度、高可疑度覆盖率、测试执行揭示错误潜力的失效执行优先级策略和随机选择 4 种策略依次记为 DIS、HSC、FEP 和 RAN。实验流程如下：

1)用原始测试用例集合对程序进行测试，产生成功用例集合和失效用例集合；

2)利用 Tarantula、CBI、Jaccard 和 Ochiai 4 种错误定位技术，分别获取每条程序语句在每种错误定位方法下的可疑度和 *Expense(unreduced)*；

3)依次选取 4 种定位方法计算的可疑度，利用 DIS、HSC、FEP 和 RAN 4 种策略分别对失效用例集合进行筛选，产生经过筛选的新失效执行集合；

4)利用加权余弦相似度匹配为每条选择出的失效执行匹配成功执行，构成新的失效-成功执行对集合；

5)依次按照对应的 4 种错误定位方法计算新的语句可疑度和 *Expense(reduced)*，获取每个被测程序在 4 种失效执行优先级策略下分别利用 4 种错误定位技术产生的 *Expense_increase*。

3.4 实验结果

由于 3 个实际 Unix 程序与 Siemens 基准测试包中 7 个程序在程序来源和源代码规模上均不同，我们在对 10 个程序统一实验后，还针对 Unix 程序和 Siemens 基准测试包分别建立了实验。首先从纵向和横向两种角度对比分析 4 种失效执行优先级策略对错误根源理解产生的 *Expense_increase*，然后对比筛选后相同 *Expense(reduced)* 下的错误理解结果，最终分析 3 种策略存在差异的原因。

3.4.1 对错误理解工作量变化的影响

1)纵向对比分析

我们选取 4 种软件错误定位技术中具有一定代表性的 Tarantula，利用 4 种失效执行优先级策略 DIS、HSC、FEP 和 RAN 对 Unix 的 3 个程序和 Siemens 基准测试包的 7 个程序进行实验。由于 Yu 的研究[37]表明测试用例的选择对软件调试过程可能产生积极和消极影响，因此为了研究不同优先级策略对错误理解产生的综合影响、积极影响和消极影响，我们统计了所有程序在不同策略下产生的 *Expense_increase* 的平均值、积极影响平均值(只统计 *Expense_increase* 为负的情况)和消极影响平均值(只统计 *Expense_increase* 为正的情况)。同时为了保证实验的完整性和普适性，我们将失效执行经过优先级排序后筛选时的百分比设定为 10%、20%、30%、40%、50%、60%、70%、80%和 90% 9 种不同情况，计算 10 个程序在这 9 种筛选率下产生的 *Expense_increase* 的平均值、积极影响平均值和消极影响平均值，如表 3 所列。

表 3 10 个实验程序在 9 种不同筛选率下产生的 *Expense_increase* 平均值、积极影响平均值和消极影响平均值

筛选率	DIS			HSC			FEP			RAN		
	平均	积极平均	消极平均	平均	积极平均	消极平均	平均	积极平均	消极平均	平均	积极平均	消极平均
10%	7.081	−1.194	8.275	15.347	−0.445	15.793	16.082	−0.402	16.484	12.65	−0.721	13.372
20%	6.920	−1.053	7.973	13.837	−0.511	14.347	14.945	−0.387	15.332	11.531	−0.682	12.213
30%	6.953	−1.079	8.033	13.922	−0.419	14.340	14.918	−0.384	15.302	11.14	−0.789	11.929
40%	6.965	−1.193	8.159	13.455	−0.271	13.726	13.704	−0.419	14.122	11.328	−0.76	12.087
50%	7.403	−1.160	8.562	13.082	−0.276	13.359	13.472	−0.439	13.911	9.743	−0.953	10.696
60%	8.038	−1.067	9.105	12.307	−0.305	12.613	12.654	−0.464	13.118	9.344	−1.061	10.405
70%	8.538	−0.717	9.254	11.583	−0.368	11.951	11.786	−0.339	12.126	10.064	−0.569	10.633
80%	8.834	−0.630	9.464	11.405	−0.405	11.810	11.521	−0.414	11.935	10.004	−0.553	10.957
90%	8.952	−0.599	9.551	11.189	−0.432	11.621	11.231	−0.433	11.665	10.751	−0.544	11.395

表中第一列表明 10%到 90%的 9 种筛选率，之后每三列一组，给出 4 种失效执行优先级策略下对错误理解产生的 *Expense_increase* 的平均值、积极影响平均值和消极影响平均值。首先观察代表对错误理解过程产生积极影响的积极平均值。从表 3 可以看出，DIS 策略在 9 种筛选率下对 10 个被测程序积极影响平均值的绝对值全部比 HSC、FEP 和 RAN 策略更大。例如 10%的筛选率下，DIS 策略产生的 *Expense_increase* 积极平均值的绝对值为 1.194，而 HSC、FEP 和 RAN 3 种策略产生的绝对值分别为 0.445、0.402 和 0.721；80%的筛选率下，DIS 策略产生的 *Expense_increase* 积极平均值的绝对值为 0.630，而 HSC、FEP 和 RAN 3 种策略产生的绝对值分别为 0.405、0.414 和 0.553。这说明相对于其他 3 种策

略，从帮助调试人员进行错误理解的部分来看，DIS 策略能够减少更多的工作量，产生更大的积极影响。

对于产生消极影响的部分，从表 3 也不难发现，DIS 策略与其他 3 种策略相比，在 9 种筛选率下对所有程序产生的 *Expense_increase* 消极平均值全部相对较小。例如 30%的筛选率下，DIS 策略产生的 *Expense_increase* 消极平均值为 8.033，而另外 3 种策略依次产生的平均值为 14.340、15.302 和 11.929。这说明相对于其他 3 种策略，DIS 策略对错误理解过程产生了更小的消极影响。

从综合所有 *Expense_increase* 的平均值角度来看，尽管在 9 个筛选率下所有的 *Expense_increase* 平均值均为正，但是 DIS 策略与其他 3 种策略相比，在 10 个被测程序中产生的平均值全部相对较低。例如 20%的筛选率下，对于 10 个被测程序，DIS 策略产生的 *Expense_increase* 平均值为 6.920，而 HSC、FEP 和 RAN 3 种策略产生的平均值分别为 13.837、14.945 和 11.531。这说明从综合影响的角度来看，DIS 相较其他 3 种策略对错误理解过程的影响更优。

分别对 Unix 的 3 个程序和 Siemens 基准测试包的 7 个程序进行实验，获取的结果如表 4 和表 5 所列。它们的数据与表 3 中获取的结果情况类似，无论从 *Expense_increase* 综合平均值，还是积极影响平均值、消极影响平均值角度来看，DIS 策略都能在 9 种不同的筛选率下，从数据上体现出明显的优势。这充分说明 DIS、HSC、FEP 和 RAN 4 种策略中，在调试人员将 Tarantula 错误定位技术获取的结果应用于错误理解过程中时，DIS 策略起到了更好的作用。

表 4　Unix 的 3 个程序在 9 种不同筛选率下产生的 *Expense_increase* 平均值、积极影响平均值和消极影响平均值

筛选率	DIS			HSC			FEP			RAN		
	平均	积极平均	消极平均	平均	积极平均	消极平均	平均	积极平均	消极平均	平均	积极平均	消极平均
10%	0.586	−1.140	1.726	3.218	−0.551	3.769	3.386	−0.496	3.883	2.814	−0.627	3.442
20%	1.110	−0.776	1.886	2.729	−0.629	3.359	3.219	−0.460	3.679	2.823	−0.678	3.501
30%	1.274	−0.672	1.946	2.810	−0.480	3.289	3.185	−0.447	3.633	1.844	−0.565	2.409
40%	1.275	−0.745	2.020	2.679	−0.562	3.241	3.126	−0.470	3.596	1.660	−0.678	2.338
50%	1.355	−0.692	2.047	2.601	−0.562	3.162	2.909	−0.490	3.399	2.053	−0.516	2.569
60%	1.397	−0.666	2.063	2.473	−0.552	3.025	2.763	−0.491	3.254	1.592	−0.593	2.184
70%	1.367	−0.664	2.030	2.077	−0.553	2.630	2.255	−0.469	2.724	1.788	−0.568	2.357
80%	1.449	−0.633	2.082	2.072	−0.515	2.587	2.279	−0.521	2.801	1.699	−0.586	2.285
90%	1.531	−0.616	2.147	1.999	−0.490	2.489	2.023	−0.487	2.510	1.943	−0.510	2.453

表 5　Siemens 基准测试包中 7 个程序在 9 种不同筛选率下产生的 *Expense_increase* 平均值、积极影响平均值和消极影响平均值

筛选率	DIS			HSC			FEP			RAN		
	平均	积极平均	消极平均	平均	积极平均	消极平均	平均	积极平均	消极平均	平均	积极平均	消极平均
10%	11.500	−1.230	12.730	23.608	−0.373	23.981	24.729	−0.338	25.066	19.349	−0.786	20.135
20%	10.880	−1.242	12.120	21.401	−0.430	21.831	22.931	−0.338	23.268	17.461	−0.685	18.146
30%	10.820	−1.357	12.180	21.489	−0.377	21.866	22.908	−0.342	23.249	17.471	−0.942	18.412
40%	10.840	−1.499	12.340	20.794	−0.074	20.868	20.907	−0.384	21.291	17.912	−0.815	18.727
50%	11.520	−1.478	13.000	20.221	−0.081	20.303	20.666	−0.404	21.070	14.980	−1.250	16.231
60%	12.560	−1.340	13.900	19.005	−0.137	19.142	19.391	−0.445	19.836	14.623	−1.280	16.003
70%	13.420	−0.753	14.170	18.057	−0.242	18.299	18.277	−0.251	18.528	15.699	−0.570	16.269
80%	13.860	−0.628	14.490	17.762	−0.330	18.091	17.815	−0.340	18.155	15.660	−0.502	16.862
90%	14.010	−0.588	14.590	17.448	−0.393	17.841	17.503	−0.397	17.900	16.750	−0.535	17.484

2)横向对比分析

由于纵向分析中，在 10%到 90%的 9 种筛选率下，4 种策略在平均值、积极影响平均值和消极影响平均值上表现出的优劣形式并未因筛选率不同而大相径庭，且一般调试人员希望筛选较少的测试执行用于错误理解动态分析，因此我们选择筛选率为 20%进行后续实验。采用盒图形式表现 4 种失效执行优先级策略在 Tarantula、CBI、Jaccard 和 Ochiai 4 种错误定位技术上对错误理解所需工作量的影响，共会出现 16 种情况。将 Tarantula、CBI、Jaccard 和 Ochiai 依次简记为 T、C、J 和 O，10 个被测程序包含的 206 个错误版本在 16 种情况下产生的 *Expense_increase* 盒图分布如图 2 所示。

从左至右每 4 个盒图一组，经对比可以发现：HSC、FEP 和 RAN 策略产生的 *Expense_increase* 盒图分布上下边界均高于 DIS 策略，而 DIS 策略产生的盒图都分布在 *Expense_increase* 值相对更低的位置。这说明 HSC、FEP、RAN 的值分布范围相对 DIS 更大，产生了更大的 *Expense_increase*。由此表明 DIS 策略在 4 种错误定位技术上对筛选前后错误理解工作量变化的影响均优于其他 3 种策略。

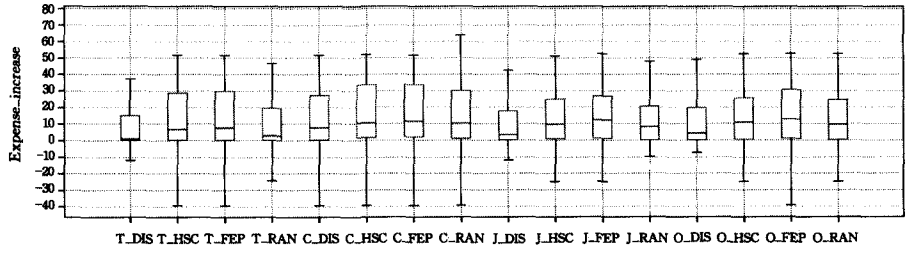

图 2　对 10 个实验程序将 4 种筛选策略应用于 4 种错误定位技术产生的 *Expense_increase* 盒图分布

同时我们也对 Unix 的 3 个程序 flex、grep 和 gzip，以及 Siemens 基准测试包的 7 个程序分别进行了盒图分布统计，结果如图 3 和图 4 所示。可以看出，在 Unix 的 3 个程序和 Siemens 的 7 个程序分别产生的 *Expense_increase* 分布中，

DIS策略的分布区域都相对较低，产生了更小的 *Expense_increase*。这再次佐证了基于覆盖语句可疑度离散度的失效执行优先级策略比其它3种策略更有助于调试人员将4种错误定位技术获取的结果应用于错误理解过程中。

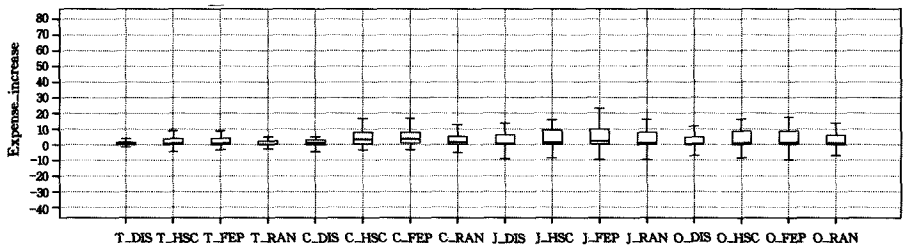

图3　对Unix的3个程序将4种筛选策略应用于4种错误定位技术产生的 *Expense_increase* 盒图分布

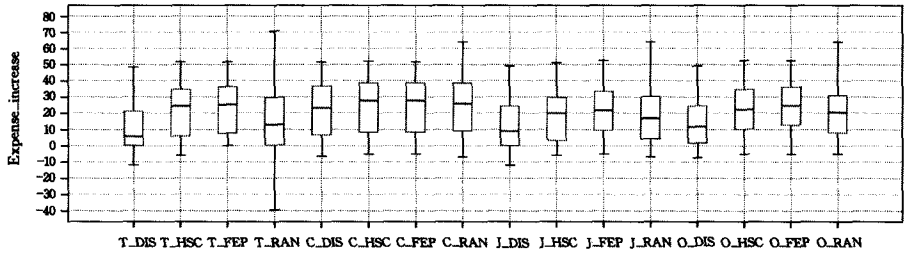

图4　对Siemens基准测试包中7个程序将4种筛选策略应用于4种错误定位技术产生的 *Expense_increase* 盒图分布

综合纵向和横向分析，在筛选更有助于将错误定位结果应用于错误理解执行上下文动态差异化分析的失效执行和成功执行时，由于该筛选与错误理解工作量间存在权衡，可能会造成积极或消极影响，因此需要找寻一种尽可能维系这种权衡，对错误理解工作量产生更强积极影响和更弱消极影响的筛选策略。而从大量实验数据可以看出，DIS策略无论在纵向还是横向的对比中，都比HSC、FEP和RAN策略对错误理解工作量产生了更强的积极影响和更弱的消极影响。

3.4.2　筛选后错误理解结果对比

为了对比4种失效执行优先级策略在4种错误定位技术下对原有测试执行集合筛选后，进行错误理解时所需的工作量，我们按照20%的筛选率进行失效执行优先级筛选，然后在0到100%的不同 *Expense(reduced)* 下，累计10个被测程序所有206个错误版本应用4种错误定位技术能够定位并有效理解错误根源的版本所占的比例，结果如图5所示，其中纵轴表示失效执行优先级策略能够定位并有效理解错误根源的比例。

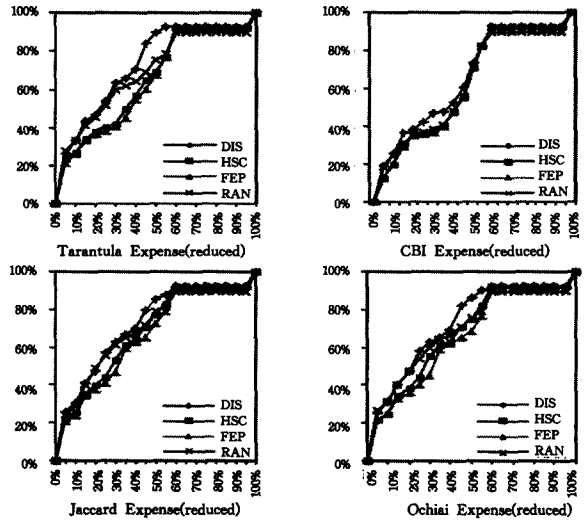

图5　10个被测程序实验结果总体对比

相同 *Expense(reduced)* 下定位并有效理解错误的比例越高，则优先级筛选策略越有效。从图5可以看出，无论采用何种错误定位技术，4种优先级策略中DIS策略有效理解错误的比例都要高于其它3种。比如 *Expense(reduced)* 为30%时，应用Tarantula技术，DIS、HSC、FEP、RAN 4种失效执行优先级策略能有效理解错误的比例分别为63.6%、42.6%、40.5%和59.5%；应用CBI算法，相对应的比例分别为46.7%、38.5%、36.4%和38.5%；应用Jaccard算法，相对应的能够有效理解错误的比例分别为63.1%、53.3%、46.7%和61.0%；应用Ochiai算法，相对应的比例分别为62.6%、55.4%、45.1%和60.0%。这说明筛选过后，DIS策略在相同的工作量下，相比其他3种策略能够帮助调试人员有效理解更多的错误。

3.5　原因分析

FEP策略将失效执行覆盖的所有程序语句的可疑度相加求和，虽然综合了执行中所有覆盖语句的可疑度，但是弱化了不同语句可疑度的高低分布情况对测试执行可疑性的影响。由于程序语句的个数一定，换个角度来看，等同于将可疑度相加之和除以程序语句个数获取平均值，按照测试执行的可疑度平均值进行排序。而这种求平均值的策略无法体现可疑度的高低差异，在区分测试执行的不同重要地位时是一种较为中庸的做法。

HSC策略统计每个测试执行覆盖的高可疑度语句数量。这种方法只着眼于高可疑度语句，且仅仅独立统计它们被覆盖的个数，没有考虑它们相互之间的分布差异。尤其是如果高可疑度语句集中分布于程序的某一段内，或分散分布在程序的不同段中时，这种单纯统计个数的方法可能会获得同样的结果，无法更好地区分测试执行。

相比之下，DIS策略总体关注了所有不同数值的语句可疑度在失效执行中的分布情况，既涉及失效执行覆盖所有语句可疑度的平均值，并消除其产生的负面影响，又能够充分体现高低不同的可疑度分布对失效执行的差异产生的作用，因此在实验中体现出较大的优势。

4 相关工作

Jiang 和 Su[38]针对基于覆盖分析的错误定位技术获取的谓词无法与错误理解直接关联的问题，结合特征筛选、聚类和控制流图遍历的方法，提出一种能够自动产生包含错误位置信息的控制流路径的技术，用来理解定位结果。

Hsu 等人[39]提出一种能够识别导致程序失效的语句序列的错误定位技术，能够通过分析与失效用例有关的部分执行轨迹，递增地识别这些轨迹中的公共序列，进而提供一种更为直观的方法帮助调试人员理解错误。

Cheng 等人[40]在方法和基本块两个粒度上建立软件执行模型，利用判别图识别相关上下文，理解错误根源。图中节点表示方法或基本块，边表示相应方法调用、返回或转移。

Rößler 等人[41]为了完全理解程序失效，提出一种收集失效用例中捕捉程序输入、结构和运行时行为等重要方面的各种"事实"的方法，通过产生附加的相似执行序列检测这些"事实"与程序失效间的相互关系。

针对大多数现有自动化调试技术存在的两个问题：1)需要大量的失效用例和成功用例；2)仅反馈可能的错误代码语句而没有对此进行解释，Zuddas 等人[10]提出了一种能够仅仅利用单独的失效用例中的故障数据发现错误根源的 MIMIC 拟态技术。它可以将多个成功和失效用例整合为与观测到的错误相似的执行，并利用这些执行探测那些或许能够解释错误根源的行为异常。

结束语 软件调试中，基于覆盖分析的错误定位依赖于原始测试用例集合按照执行结果状态划分的失效用例和成功用例。为了以错误定位结果为导向，更好地理解错误根源和程序失效场景，调试人员应将静态的测试用例作为动态分析过程中的测试执行，并筛选部分失效执行和成功执行，对程序运行时状态和预期状态的执行上下文进行差异化分析。

以基于覆盖分析的错误定位结果为导向，本文关注用何种策略筛选更适于理解错误根源的失效执行，以及能与失效执行构成执行上下文最小差异的成功执行。提出的失效执行筛选策略包括基于测试执行高可疑度覆盖率、揭示错误潜力和覆盖语句可疑度离散度等 3 种面向错误定位与理解的启发式优先级策略，针对成功执行匹配提出结合测试执行覆盖信息的加权余弦相似度算法。将错误理解过程的工作量变化和筛选后的错误理解结果作为具体量化影响错误理解能力的指标，在相同的成功执行加权余弦相似度匹配策略下，通过将 3 种失效执行优先级策略和随机选择策略应用于 4 种常用错误定位技术的实验对比，可以得到如下两个结论：

1)基于覆盖语句可疑度离散度的失效执行优先级策略相较基于高可疑度覆盖率、测试执行揭示错误潜力和随机选择的策略，对筛选前后错误理解工作量变化产生了更强的积极影响和更弱的消极影响，总体影响也优于另外 3 种策略；

2)在相同的错误理解工作量下，基于覆盖语句可疑度离散度的失效执行优先级策略能够有效理解更多的错误。

因此，基于覆盖语句可疑度离散度的测试执行筛选策略对于筛选前后错误理解工作量变化和筛选后错误理解结果的影响均优于其他 3 种策略，能够更好地结合错误定位结果，帮助调试人员筛选更有助于后续错误根源理解和修复的失效执行和成功执行。

参 考 文 献

[1] Shen Chang-xiang, Zhang Huan-guo, Feng Deng-guo, et al. Surver of information security [J]. Science China(E), 2007, 32(2): 129-150(in Chinese)
沈昌祥，张焕国，冯登国，等. 信息安全综述[J]. 中国科学 E 辑：信息科学，2007，32(2)：129-150

[2] Duraes A, Madeira H, Duraes J. Emulation of software faults: a field data study and a practical approach [J]. IEEE Transaction on Software Engineering, 2006, 32(11): 849-867

[3] Zhao Lei, Wang Li-na, Gao Dong-ming, et al. Mining associations to improve the effectiveness of fault localization[J]. Chinese Journal of Computers, 2012, 35(12): 2528-2540(in Chinese)
赵磊，王丽娜，高东明，等. 基于关联挖掘的软件错误定位方法[J]. 计算机学报，2012，35(12)：2528-2540

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

[5] Cao He-ling, Jiang Shu-juan, Ju Xiao-lin. Survey of software fault localization[J]. Computer Science, 2014, 41(2): 1-6(in Chinese)
曹鹤玲，姜淑娟，鞠小林. 软件错误定位研究综述[J]. 计算机科学，2014，41(2)：1-6

[6] Arumuga Nainar P, Chen T, Rosin J, et al. Statistical debugging using compound Boolean predicates[C]// Proceedings of the 2007 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2007). New York: ACM Press, 2007: 5-15

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

[8] Chilimbi T, Liblit B, Mehra K, et al. HOLMES: effective statistical debugging via efficient path profiling[C]// Proceedings of the 31st International Conference on Software Engineering (ICSE 2009). Vancouver, Canada: 2009: 34-44

[9] 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. ACM, 2005: 273-282

[10] Zuddas D, Wei Jin, Pastore F, et al. MIMIC: locating and understanding bugs by analyzing mimicked executions[C]// Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering. ACM, 2014: 815-826

[11] Sumner W N, Bao Tao, Zhang Xiang-yu. Selecting peers for execution comparison[C]// Proceedings of the 2011 International Symposium on Software Testing and Analysis. ACM, 2011: 309-319

[12] Bettenburg N, Just S, Schröter A, et al. What makes a good bug report? [C]// Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering. New York, USA: ACM Press, 2008: 308-318

[13] Brumley D, Caballero J, Liang Zhen-kai, et al. Towards Automatic Discovery of Deviations in Binary Implementations with Applications to Error Detection and Fingerprint Generation[C]// Proceedings of 16th USENIX Security Symposium on USENIX

Security Symposium. 2007

[14] Jeffrey D, Gupta N, Gupta R. Fault localization using value replacement[C]// Proceedings of the 2008 International Symposium on Software Testing and Analysis. New York, USA: ACM Press, 2008(1): 167-178

[15] Evans R, Savoia A. Differential testing: a new approach to change detection[C]// Proceedings of the the 6th joint meeting of the European Software Enginee-ring Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering: Companion Papers. ACM, 2007: 549-552

[16] Wong W, Horgan J, London S, et al. A study of effective regression testing in practice[C]// Proceedings of the International Symposium on Software Reliability Engineering. IEEE Press, 1997: 264-274

[17] Rothermel G, Untch R H, Chu Cheng-yun, et al. Test case prioritization: An empirical study[C]// IEEE International Conference on Software Maintenance, 1999(ICSM'99). IEEE, 1999: 179-188

[18] Elbaum S, Malishevsky A G, Rothermel G. Prioritizing test cases for regression testing[C]// Proceedings of the International Symposium on Software Testing and Analysis. ACM Press, 2000: 102-112

[19] Elbaum S, Malishevsky A G, Rothermel G. Test case prioritization: A family of empirical studies[J]. IEEE Transactions on Software Engineering, 2002, 28(2): 159-182

[20] Yan Sha-li, Chen Zhen-yu, Zhao Zhi-hong, et al. A dynamic test cluster sampling strategy by leveraging execution spectra information[C]// 2010 Third International Conference on Software Testing, Verification and Validation (ICST). IEEE, 2010: 147-154

[21] Taneja K, Xie Tao, Tillmann N, et al. Guided path exploration for regression test generation[C]// 31st International Conference on Software Engineering — Companion Volume, 2009. IEEE, 2009: 311-314

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

[23] Wang Tao, Roychoudhury A. Automated path generation for software fault localization[C]// Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering. ACM, 2005: 347-351

[24] Guo Liang, Roychoudhury A, Wang Tao. Accurately choosing execution runs for software fault localization[M]// Compiler Construction. Springer Berlin Heidelberg, 2006: 80-95

[25] 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. ACM, 2005: 273-282

[26] Voas J M. PIE: A dynamic failure-based technique [J]. IEEE Transactions on Software Engineering, 1992, 18(8): 717-727

[27] Li Jia-jing, Su Xiao-hong, Ma Pei-jun, et al. A test cases selection method oriented to fault localization requirement based on branch clustering [J]. Intelligent Computer and Application, 2012, 2(5): 16-19(in Chinese)
李佳婧, 苏小红, 马培军, 等. 面向错误定位的基于分支聚类的测试用例选择方法[J]. 智能计算机与应用, 2012, 2(5): 16-19

[28] He Xiao-cong. Coefficient of variation and its application to strength prediction of adhesively bonded joints[C]// Measuring Technology and Mechatronics Automation, 2009. ICMTMA'09. International Conference on. IEEE, 2009, 1: 602-605

[29] Xue Li-xiang, Qiu Bao-zhi. Boundary points detection algorithm based on coefficient of variation[J]. Pattern Recognition and Artificial Intelligence, 2009(5): 799-802(in Chinese)
薛丽香, 邱保志. 基于变异系数的边界点检测算法[J]. 模式识别与人工智能, 2009(5): 799-802

[30] Nierman A, Jagadish H V. Evaluating structural similarity in XML documents[C]// Proceedings of the WebDB Workshop. Madison Wisconsin, USA: EECS, 2002: 61-66

[31] Lee Y K, Kim W Y, Cai Y D, et al. CoMine: efficient mining of correlated patterns[C]// Proceedings of the Third IEEE International Conference on Data Mining. Melbourne, 2003: 581-584

[32] Omiecinski E R. Alternative interest measures for mining associations in databases[J]. IEEE Transactions on Knowledge and Data Engineering, 2003, 15(1): 57-69

[33] Anna K. A fast outlier detection strategy for distributed high-dimensional data sets with mixed attributes[J]. Data Mining and Knowledge Discovery, 2010, 20(2): 259-289

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

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

[36] Abreu R, Zoeteweij P, Van Gemund A J C. On the accuracy of spectrum-based fault localization[C]// Testing: Academic and Industrial Conference Practice and Research Techniques — MUTATION, 2007. IEEE, 2007: 89-98

[37] Yu Yan-bing, Jones J A, Harrold M J. An empirical study of the effects of test-suite reduction on fault localization[C]// Proceedings of the 30th International Conference on Software Engineering. ACM, 2008: 201-210

[38] Jiang Ling-xiao, Su Zhen-dong. Context-aware statistical debugging: from bug predictors to faulty control flow paths[C]// Proceedings of the 22nd IEEE International Conference on Automated Software Engineering (ASE 2007). Atlanta, USA, 2007: 184-193

[39] Hsu H-Y, Jones J A, Orso A. Rapid: Identifying bug signatures to support eebugging activities[C]// 2008 23rd IEEE/ACM International Conference on Automated Software Engineering. IEEE, 2008(2): 439-442

[40] Cheng Hong, Lo D, Zhou Yang, et al. Identifying bug signatures using discriminative graph mining[C]// Proceedings of the eighteenth International Symposium on Software Testing and Analysis-ISSTA '09. New York, USA: ACM Press, 2009: 141-150

[41] Rößler J, Fraser G, Zeller A, et al. Isolating failure causes through test case generation[C]// Proceedings of the 2012 International Symposium on Software Testing and Analysis. ACM, 2012: 309-319