ISSN 1673-9418 CODEN JKYTA8
Journal of Frontiers of Computer Science and Technology
1673-9418/2012/06(12)-1057-19
DOI: 10.3778/j.issn.1673-9418.2012.12.001

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

变异测试:原理、优化和应用*

陈 翔[1,2+],顾 庆[2]
1. 南通大学 计算机科学与技术学院,江苏 南通 226019
2. 南京大学 软件新技术国家重点实验室,南京 210093

Mutation Testing: Principal, Optimization and Application*

CHEN Xiang[1,2+], GU Qing[2]
1. School of Computer Science and Technology, Nantong University, Nantong, Jiangsu 226019, China
2. State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China
+ Corresponding author: E-mail: xchencs@ntu.edu.cn

CHEN Xiang, GU Qing. Mutation testing: principal, optimization and application. Journal of Frontiers of Computer Science and Technology, 2012, 6(12): 1057-1075.

Abstract: Mutation is a fault-based testing technique. This topic is widely researched for over 40 years. This paper summarizes previous research work into three modules: principal, optimization and application. In the principal module, this paper firstly introduces two fundamental hypotheses, secondly illustrates the traditional process of mutation analysis and gives definitions for the important concepts, lastly summarizes equivalent mutant detection techniques into static detection and dynamic detection categories. In the optimization module, this paper illustrates mutant selection optimization and mutant execution optimization. In the application module, this paper introduces three classical applications: test suite adequacy evaluation, test case generation and regression testing. Finally, this paper draws a conclusion and forecasts some potential future research work.

Key words: mutation testing; equivalent mutant; test suite adequacy; test case generation; regression testing

---

* The National Natural Science Foundation of China under Grant Nos. 61202006, 61021062 (国家自然科学基金); the Natural Science Research Project of Higher Education of Jiangsu Province under Grant No. 12KJB520014 (江苏省高校自然科学研究项目); the Application Research Plan of Nantong under Grant No. BK2012023 (南通市应用研究计划项目); the Open Project of State Key Laboratory for Novel Software Technology at Nanjing University under Grant No. KFKT2012B29 (南京大学计算机软件新技术国家重点实验室开放课题).
Received 2012-07, Accepted 2012-09.

摘 要: 变异测试是一种基于缺陷的软件测试技术,在近四十年得到国内外学者的广泛关注,并取得了一些研究成果。对已有的研究工作进行总结,将其分为变异测试原理、优化和应用三个模块。其中在变异测试原理模块中,给出变异测试的基本假设,对变异测试分析流程进行介绍,并对其中的重要概念依次给出定义,从静态检测和动态检测两个角度对等价变异体检测技术进行总结。在变异测试优化模块中,从变异体选择优化和变异体执行优化两个角度对已有研究工作进行总结。在变异测试应用模块中,选择了测试用例集充分性评估、测试用例生成和回归测试三个应用领域,对研究工作进行分类总结。最后对变异测试的未来研究方向进行了展望。

关键词: 变异测试;等价变异体;测试用例集充分性;测试用例生成;回归测试

文献标识码: A **中图分类号:** TP311.5

1 引言

软件测试是软件开发和维护过程中保障软件质量的重要手段,其中测试用例设计是软件测试中的核心问题。目前测试人员一般借助控制流或数据流分析来定义测试充分性准则[1],并用于指导随后的测试用例设计。本文的变异测试(mutation testing)则从另一个角度对测试充分性进行了度量,该测试技术可用于评估和改进测试用例集(test suite)的测试充分性。具体来说,测试人员首先根据被测程序特征设计变异算子(mutation operator),变异算子一般在符合语法前提下仅对被测程序作微小改动。然后对被测程序应用变异算子可生成大量变异体(mutant),在识别出等价变异体(equivalent mutant)后,若已有测试用例不能杀除所有非等价变异体,则需要额外设计新的测试用例,并添加到测试用例集中,以提高测试充分性。除了可用于测试用例集的充分性评估,变异测试也可以通过采用变异缺陷来模拟被测软件的真实缺陷,从而对研究人员提出的测试方法的有效性进行辅助评估。例如 Andrews 等人[2]和 Do 等人[3]均在他们的实证研究中,证实了变异算子生成的变异缺陷与真实缺陷在有效性评估中效果相似。上述变异测试分析有时又简称为变异分析(mutation analysis)。

变异测试作为一种基于缺陷的软件测试技术,在近四十年得到国内外学者的广泛关注。变异测试概念的提出最早可以追溯到 Hamlet[4]和 DeMillo 等人[5]在 20 世纪 70 年代早期的研究工作。随后研究人员对该领域展开了深入的研究,并发表了大量的论文。同时也开发出一些变异测试工具,例如 Mothra[6]、Proteum[7]、MuJava[8]、MuClipse[9]和 Javalanche[10]等。目前变异测试不仅成功应用于不同编程语言,例如 Fortran 语言、Ada 语言、C 语言、Java 语言、C# 语言、SQL 语言和 AspectJ 等,而且还成功应用于程序规约(specification)和程序模型上,例如有限状态自动机、状态图、Petri 网、网络协议和安全策略等[11]。

通过对已有文献的整理和分析,提出图 1 所示的研究框架。该框架将变异测试目前已有的研究成果分别纳入到原理、优化和应用三个模块中。在变异测试原理模块中,首先给出变异测试的两个基本假

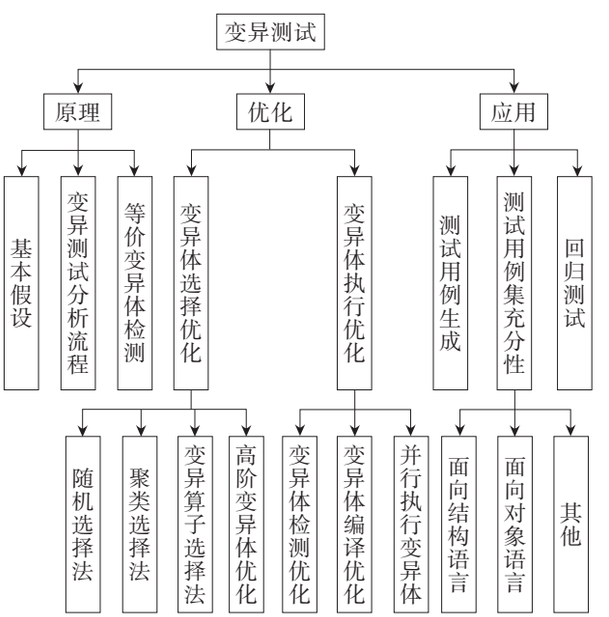

Fig.1 The research framework of mutation testing

图 1 变异测试研究框架

设,随后总结变异测试分析流程,并结合该流程对变异测试中的重要概念依次进行定义,最后从静态检测和动态检测两个角度对等价变异体检测技术进行分类和总结,具体内容见第2章。在变异测试优化模块中,主要从变异体选择优化和变异体执行优化两个角度对已有工作进行分类和总结。变异体选择优化主要包括随机选择法、聚类选择法、变异算子选择法和高阶变异体优化法四类方法。而变异体执行优化主要考虑变异体检测优化、变异体编译优化和并行执行变异体,具体内容见第3章。在变异测试应用模块中,主要考虑了测试用例集充分性评估、测试用例生成和回归测试三个典型应用,具体内容分别见第4章、第5章和第6章。最后对全文进行总结,并对未来可能研究方向进行了展望。

与Jia和Harman在TSE上发表的论文[11]不同,本文首次从三个维度(即原理、优化和应用)出发,对变异测试的已有研究成果进行了归类、分析和比较。同时补充了大量国内外研究人员近三年(截止到2012年9月份)在相关会议和期刊上的最新研究成果。除此之外,特别在变异测试原理分析、面向变异测试的测试用例生成技术以及回归测试等内容上进行了更加深入的总结,可以作为文献[11]的有效补充。

2　变异测试原理

2.1　基本假设

让变异测试生成代表被测程序所有可能缺陷的变异体的策略并不可行,传统变异测试一般通过生成与原有程序差异极小的变异体来充分模拟被测软件的所有可能缺陷。其可行性基于两个重要的假设:熟练程序员假设和耦合效应假设。

假设1(熟练程序员假设) DeMillo等人在1978年首先提出该假设[5]。即假设熟练程序员因编程经验较为丰富,编写出的有缺陷代码与正确代码非常接近,仅需作小幅度代码修改就可以完成缺陷的移除。基于该假设,变异测试仅需通过对被测程序作小幅度代码修改就可以模拟熟练程序员的实际编程行为。

假设2(耦合效应假设) 与假设1关注熟练程序员的编程行为不同,假设2关注的是软件缺陷类型。该假设同样由DeMillo等人首先提出[5]。他们认为,若测试用例可以检测出简单缺陷,则该测试用例也易于检测到更为复杂的缺陷。Offutt随后对简单缺陷和复杂缺陷进行了定义[12],即简单缺陷是仅在原有程序上执行单一语法修改形成的缺陷,而复杂缺陷是在原有程序上依次执行多次单一语法修改形成的缺陷。根据上述定义可以进一步将变异体细分为简单变异体和复杂变异体,同时在假设2基础上提出变异耦合效应。复杂变异体与简单变异体间存在变异耦合效应是指若测试用例集可以检测出所有简单变异体,则该测试用例集也可以检测出绝大部分的复杂变异体。该假设为变异测试分析中仅考虑简单变异体提供了重要的理论依据。研究人员进一步通过实证研究对假设2的合理性进行了验证。例如,Offutt通过对三个程序MID、TRITYP和FIND的考察发现,检测出所有简单变异体(即一阶变异体)的测试用例集可以检测出超过99%的复杂变异体(即二阶变异体和三阶变异体)[12]。

2.2　变异测试分析流程

传统变异测试分析流程如图2所示。

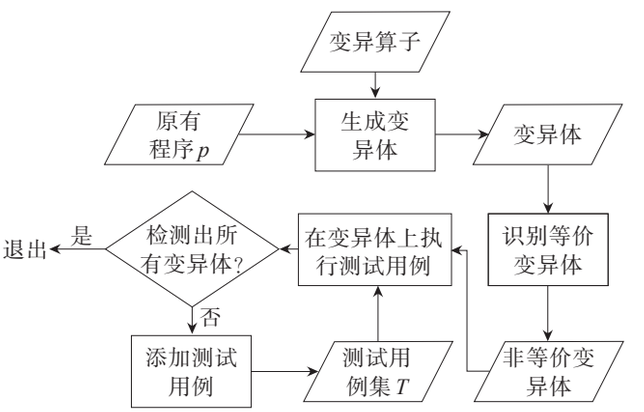

Fig.2　The traditional process of mutation testing analysis

图2　传统变异测试分析流程

给定被测程序 p 和测试用例集 T,首先根据被测程序特征设定一系列变异算子;随后通过在原有程序 p 上执行变异算子生成大量变异体;接着从大量变异体中识别出等价变异体;然后在剩余的非等价变异体上执行测试用例集 T 中的测试用例,若可

以检测出所有非等价变异体,则变异测试分析结束,否则对未检测出的变异体,需要额外设计新的测试用例,并添加到测试用例集 T 中。

基于上述传统变异测试分析流程,对其中的基本概念依次定义如下。

定义1(变异算子) 在符合语法规则前提下,变异算子定义了从原有程序生成差别极小程序(即变异体)的转换规则。

表1给出了一个典型的变异算子,该变异算子将“+”操作符变异为“-”操作符。选择被测程序 p 中的条件表达式 $a+b>c$ 执行该变异算子,将得到条件表达式 $a-b>c$,并生成变异体 p'。

Table 1 A typical mutation operator

表1 一个典型的变异算子

程序 p	变异体 p'
…	…
if $(a+b>c)$	if $(a-b>c)$
return true;	return true;
…	…

Offutt和King在已有研究工作的基础上,于1987年针对Fortran77首次定义了22种变异算子,这些变异算子的简称和描述如表2所示[6]。这22种变异算子的设定为随后其他编程语言变异算子的设定提供了重要的指导依据。

在完成变异算子设计后,通过在原有被测程序上执行变异算子可以生成大量变异体 M,在变异测试中,变异体一般被视为含缺陷程序。根据执行变异算子的次数,可以将变异体分为一阶变异体和高阶变异体,并分别定义如下。

定义2(一阶变异体) 在原有程序 p 上执行单一变异算子并形成变异体 p' ,则称 p' 为 p 的一阶变异体。

定义3(高阶变异体) 在原有程序 p 上依次执行多次变异算子并形成变异体 p' ,则称 p' 为 p 的高阶变异体。若在 p 上依次执行 k 次变异算子并形成变异体 p' ,则称 p' 为 p 的 k 阶变异体。

根据定义2和定义3易知,高阶变异体和一阶变异体间存在包含关系。根据假设2可知,相对于一阶变异体,高阶变异体更容易被测试用例检测到。但在高阶变异体中也存在一小部分变异体,其比对应的一阶变异体更难以被测试用例检测到。Jia和Harman等人对这类高阶变异体进行了深入研究[13-15]。以图3为例,图中左侧是一段被测代码,右侧有两个一阶变异体,其中通过将第一行代码变异为 $z=++x$ 以生成变异体1,将第二行代码变异为 $z=z+--y$ 以生成变异体2。右下角的两个测试用例均可以检测到变异体1或变异体2,但若将变异体1和变异体2合成为一个二阶变异体,则仅有一个测试用例可以检测出该二阶变异体。这类变异体一般比较稀有,利用价值较高,并可以更好地模拟实际缺陷。

Table 2 22 mutation operators designed for Fortran77

表2 针对Fortran77定义的22种变异算子

序号	变异算子	描述
1	AAR	用一数组引用替代另一数组引用
2	ABS	插入绝对值符号
3	ACR	用数组引用替代常量
4	AOR	算术运算符替代
5	ASR	用数组引用替代变量
6	CAR	用常量替代数组引用
7	CNR	数组名替代
8	CRP	常量替代
9	CSR	用常量替代变量
10	DER	DO语句修改
11	DSA	DATA语句修改
12	GLR	GOTO标签替代
13	LCR	逻辑运算符替代
14	ROR	关系运算符替代
15	RSR	RETURN语句替代
16	SAN	语句分析
17	SAR	用变量替代数组引用
18	SCR	用变量替代常量
19	SDL	语句删除
20	SRC	源常量替代
21	SVR	变量替代
22	UOI	插入一元操作符

当测试用例集 T 中的所有测试用例在生成的变异体上执行结束后,所有的变异体可以被划分为可

```
输入：a, x, y
1. z=x;
2. z=z+y;
3. if (a>0)
4.    return z;
5. else
6.    return 2*x+z;
```

测试用例	$a=1$	$a=-1$
原有程序	$x+y$	$3x+y$
变异体1	$x+y+1$	$3x+y+3$
变异体2	$x+y-1$	$3x+y-1$
变异体12	$x+y$	$3x+y+2$

其中：
变异体1(一阶变异体)：
将第一行变异为 $z=++x$
变异体2(一阶变异体)：
将第二行变异为 $z=z+--y$
变异体12(二阶变异体)：
合并变异体1和变异体2
两个测试用例：
(1)a=1　(2)a=−1

Fig.3　An instance of high order mutant

图3　高阶变异体实例

杀除变异体(killed mutants)和可存活变异体(survived mutants)，并依次定义如下。

定义4(可杀除变异体) 若存在测试用例 $t(t\in T)$，在变异体 p' 和原有程序 p 上的执行结果不一致，则称该变异体 p' 相对于测试用例集 T 是可杀除变异体。

定义5(可存活变异体) 若不存在任何测试用例 $t(t\in T)$，在变异体 p' 和原有程序 p 上的执行结果不一致，则称该变异体 p' 相对于测试用例集 T 是可存活变异体。

一部分可存活变异体通过设计新的测试用例可以转化成可杀除变异体，剩余的可存活变异体则可能是等价变异体。本文对等价变异体定义如下。

定义6(等价变异体) 若变异体 p' 与原有程序 p 在语法上存在差异，但在语义上与 p 保持一致，则称 p' 是 p 的等价变异体。

表3给出了一个典型的等价变异体。变异算子将被测程序 p 中for结构中的“<”操作符变异为“!=”操作符，如果循环体中不存在修改变量 i 的语句，则程序 p' 是程序 p 的等价变异体。

根据定义6易知，等价变异体不可能被任意测试用例检测到，因此在变异测试分析中需要排除这类变异体。但等价变异体的检测是一个不可判定问题[16]，是变异测试得到进一步使用和推广的主要障碍。第2.3节将从静态检测和动态检测两个角度对已有的等价变异体检测方法进行总结。

Table 3　A typical equivalent mutant

表3　一个典型的等价变异体

程序 p	变异体 p'
for (int $i=0;i<10;i++$){	for (int $i=0;i!=10;i++$){
$sum+=a[i]$;	$sum+=a[i]$;
}	}

变异测试分析最终通过输出变异评分(mutation score)来评估测试用例集的缺陷检测能力，变异评分 $MS(M,T)$ 可通过式(1)计算获得：

$$MS(M,T)=\frac{killed(M,T)}{|M|-eqv(M)} \tag{1}$$

其中，$killed(M,T)$ 函数返回测试用例集 T 可以检测出的变异体数量；$eqv(M)$ 函数返回等价变异体的数量。根据式(1)可以看出，MS 的取值范围介于0到1之间，MS 的取值越高，代表测试用例集的实际缺陷检测能力越强。变异测试分析的最优目标是希望 MS 的取值达到1.0，即测试用例集可以检测出所有的非等价变异体。

2.3　等价变异体检测

等价变异体检测是一个不可判定问题[16]，因此一般需要测试人员借助手工方式予以完成。第2.2节已经对等价变异体进行了定义，并给出了一个典型的等价变异体实例。虽然等价变异体与原有程序在语法层次上仅存在微小差别，但在语义层次上却保持一致。由于变异评分的计算建立在等价变异体识别完成之后，若不能检测出所有等价变异体，则变异评分将无法达到100%，导致测试人员很难对测试用例集的测试充分性进行准确判断。Offutt等人通过实证研究发现，在生成的大量变异体中，等价变异体所占比例一般介于10%到40%之间[16-17]。Grun等人对Jaxen XPath查询引擎生成的8个等价变异体进行分析[18]，给出了生成等价变异体的4种情况：针对无用代码的变异，抑制程序运行性能提高的变异，仅改变程序运行时内部状态但不影响程序输出结果的变异，以及变异语句不能被任意测试用例覆盖到的变异。但上述分析并不完备，仍存在一些难以归入

上述4种情况的等价变异体，如表3给出的这个等价变异体。根据是否需要执行变异体，本文将已有研究工作分为两类：等价变异体静态检测法和等价变异体动态检测法。

2.3.1 等价变异体静态检测法

Baldwin和Sayward借助编译器优化技术来进行等价变异体检测[19]。该方法基于如下猜测：源代码在编译时借助优化规则可以生成语义等价代码。他们提出6条编译优化规则来辅助等价变异体的检测。Offutt和Craft针对上述优化规则设计了算法，并开发出Equalizer工具[17]，在考察的15个程序对象中，该工具可以自动检测出10%的等价变异体。

Offutt和Pan将等价变异体检测问题作为不可行路径问题的一个实例进行研究。该方法建立在对变异体路径条件（path condition）的求解上。当约束求解器能够表明随后所有状态均是等价的，则可以认为该变异体是等价变异体。在分析完11个源程序后，发现该方法可以自动检测出47.63%的等价变异体[16]。除此之外，Hierons等人提出相似方法[20]，借助程序切片技术来协助测试人员提高等价变异体的识别效率。

2.3.2 等价变异体动态检测法

Adamopoulos等人提出一种基于遗传算法的协作演化法（即测试用例和变异体同时进行演化）来检测可能的等价变异体[21]。他们通过设置合理的适应值函数，确保当变异体是等价变异体时，该函数可以返回一个很小的适应值。基于该适应值函数，群体在演化过程中可以有效淘汰部分等价变异体，同时将那些难以检测的变异体和检测能力强的测试用例均保留下来。

Grun等人[18]借助变异体对程序执行的影响来识别等价变异体。他们将变异体的影响定义为原有程序和变异体间存在的程序行为差异，并通过语句覆盖信息对影响程度进行度量。结果表明，变异体检测能力和对应影响力上存在强相关性，即影响力较高的变异体成为等价变异体的概率较小。Schuler等人使用动态不变量（dynamic invariants）来定义程序的正常行为。在未被已有测试用例检测到的变异体中，若违背的动态不变量越多，则它影响程序语义的可能性也越高，即成为等价变异体的概率越低[22]。在变异测试分析中，需要设计新的测试用例来检测出这类变异体。

3 变异测试优化技术

从学术界观点来看，变异测试是一种基于缺陷的成熟测试技术，但应用到工业界时却存在一个重要的技术难点，即变异测试分析过程中的计算开销较大。例如，Delamaro和Maldonado发现，针对西门子套件中最简单的程序tcas（仅包含137行非注释代码），Proteum系统仅应用108个变异算子就可以生成4 937个变异体[7]。因此大量变异体的生成使得变异测试和分析时的开销极为高昂。其主要开销分布于变异体的生成、编译和执行过程中。为了将变异测试技术从学术界研究转化到工业界应用，需要提出有效的优化方法来减小计算开销。目前，研究人员一般从变异体选择优化和变异体执行优化两个角度展开深入研究。

3.1 变异体选择优化

变异体选择优化策略主要关注如何从生成的大量变异体中选择出典型变异体。该问题可简要描述如下。

给定：变异体集 M，测试用例集 T，$MS(M,T)$ 函数返回测试用例集 T 在变异体集 M 上的变异评分。

问题：寻找 M 的子集 M'，使得 $MS(M',T)\approx MS(M,T)$。

已有的变异体选择优化方法可以简单分为4类：随机选择法、聚类选择法、变异算子选择法和高阶变异体优化法。

3.1.1 随机选择法

随机选择法尝试从生成的大量变异体中随机选择出部分变异体。具体来说，首先通过执行变异算子生成大量变异体 M；然后定义选择比例 x；最后从变异体 M 中随机选择出 $|M|\times x\%$ 的变异体，剩余未被选择的变异体则被丢弃。研究人员在实证研究中对选择比例 x 的取值进行了探讨。例如，Wong和Mathur对Mothra系统中的22种变异算子展开了研究，将 x 的取值分别设置为 $\{10,15,20,\ldots,40\}$[23-24]，

结果表明，若随机选择10%的变异体，相对于选择所有变异体，其变异评分仅减少16%。x的取值若超过10，随机选择法是一种行之有效的优化方法。Wong和Mathur提出的随机选择法可归为单轮随机选择法，张路等人提出一种双轮随机选择法[25]。其中随机选择一个变异体的步骤是，首先随机选择一种变异算子，随后从该变异算子生成的变异体中再随机选择一个变异体。实证研究结果表明，相对于变异算子选择法[26-28]，随机选择法可以选择出少量的变异体达到同样效果。

3.1.2　聚类选择法

与上述随机选择法不同，该方法选择聚类算法对变异体进行聚类分析。具体来说，首先对被测程序p应用变异算子生成所有的一阶变异体；然后选择某一聚类算法根据测试用例的检测能力对所有变异体进行聚类分析，使得每个聚类内的变异体可以被相似测试用例检测到；最后从每个聚类中选择出典型变异体，而其他变异体则被丢弃。Hussain[29]在他的实证研究中选择了两种聚类算法*K*-means和Agglomerative Clustering算法，并与随机选择策略和贪婪选择策略进行了比较。结果表明，聚类法在不牺牲变异评分的前提下选择出的变异体数量更少。陈振宇等人则进一步采用域约简（domain reduction）技术来缩减变异测试开销[30]。

3.1.3　变异算子选择法

与上述两类方法不同，这类方法从变异算子选择角度出发，希望在不影响变异评分的前提下，通过对变异算子进行约简来大规模缩小变异体数量，从而减小变异测试和分析开销。这是一类研究人员关注较多的方法。Mathur在1991年首次提出该类方法，并将之称为选择性变异（selective mutation）[31]。他们在变异测试和分析中发现，不同类型的变异算子可以生成的变异体数量存在较大差别。例如，在针对Mothra系统设计的22种变异算子中，ASR算子和SVR算子可以生成30%到40%的变异体[6]。Offutt等人建议在选择变异算子时，通过忽略ASR算子和SVR算子可以有效减少生成的变异体数量，并将该策略称为“2-selective Mutation”[32]。同时他们又提出另2种选择策略，分别是忽略4种变异算子的“4-selective Mutation”策略和忽略6种变异算子的“6-selective Mutation”策略。结果表明，应用“2-selective Mutation”策略后，其变异评分均值为99.99%，可减少24%的变异体数量。应用“4-selective Mutation”策略后，其变异评分均值为99.84%，可减少41%的变异体数量。而应用“6-selective Mutation”策略后，其变异评分均值为88.71%，可减少60%的变异体数量。

Wong和Mathur则根据测试有效性对变异算子进行选择，并将之称为约束变异（constrained mutation）[24]。他们建议在Mothra系统中选择两种变异算子，即ABS算子和RAR算子。选择ABS算子是因为可检测出这类变异体的测试用例，其输入需要来自不同的输入域；选择ROR算子是因为可检测出这类变异体的测试用例需要覆盖到变异后的谓词。结果表明，仅选择这两种变异算子可以有效减少80%的变异体，并且变异评分值仅降低5%。Offutt等人[27]在已有的研究基础上进一步分析，将Mothra系统的22种变异算子分为语句相关、操作数相关和表达式相关3类。依次对每一类里的变异算子进行分析，最终从Mothra系统的22种变异算子中选择出5种重要变异算子：ABS算子、UOI算子、LCR算子、AOR算子和ROR算子。从中不难看出，这5种变异算子包含了Wong和Mathur选择的两种变异算子[24]。结果表明，选择这5种变异算子其变异评分仅降低0.5%。

Mresa和Bottaci则尝试了另一种思路，在不大幅度影响测试有效性的前提下，同时考虑了测试用例生成和等价变异体检测的开销[33]。在他们的研究工作中，每个变异算子根据价值和开销均被赋予了不同评分。结果表明，该方法可以在确保测试有效性的前提下有效减少等价变异体的数量。Barbosa等人在已有研究基础上，设计了从所有可能变异算子中选择出充分变异算子子集的6条指导策略[26]。他们将该指导策略应用到包含77种变异算子的Proteum系统中，并从中选择了10种典型变异算子子集。结果表明，这10种变异算子可以有效减少65.02%的变异体，但变异评分仅降低0.4%。Namin等人[28]将变异算子选择问题看做统计问题，并采用线性统计方法

从C语言的108种变异算子中选择出28种变异算子。最终结果表明,这28种变异算子足够用于评估测试用例集的测试充分性,同时可以减少92%的变异体。

张路等人选择了西门子套件中的7个程序,在实证研究中对2种随机选择法和3种变异算子选择法[26-28]进行了比较。结果表明,变异算子选择法相对于随机选择法来说并不存在明显优势,随机选择法值得研究人员继续深入研究[25]。

3.1.4 高阶变异体优化法

高阶变异体优化法基于如下推测:(1)执行一个k阶变异体相当于一次执行k个一阶变异体;(2)高阶变异体中等价变异体的出现概率较小。例如,Offutt等人在他们的实证研究中发现,一阶变异体中包含约10%的等价变异体,而二阶变异体中则仅包含1%的等价变异体[6]。

Jia和Harman[13]引入包含(subsuming)高阶变异体概念,与耦合效应相反,这类变异体比对应的一阶变异体更难以被测试用例检测到。同时他们提出3种搜索方法(贪婪法、遗传算法和爬山法)来寻找这类变异体。Langdon等人应用多目标方法(multi-objective approach)来指导生成高阶变异体,该方法在实现时结合了遗传规划、遗传算法和蒙特卡罗抽样。实证研究表明,该方法可以生成一些高阶变异体,这类变异体比任何一阶变异体都难以被测试用例检测到[14]。Polo等人在实证研究中重点关注了二阶变异体,他们提出3种将一阶变异体组合成二阶变异体的构造算法。实证研究表明,采用二阶变异体可以有效减少50%的测试开销,但却不会显著降低测试的有效性[34]。

3.2 变异体执行优化

除了通过减少变异体数量来指导变异测试优化外,也有研究人员从优化变异体执行时间角度展开研究。已有研究工作可以简要分为3类:变异体检测优化、变异体编译优化和并行执行变异体。

3.2.1 变异体检测优化

在变异体执行优化时,可以通过提高变异体检测效率来提高变异测试分析效率。测试用例检测变异体的方式可以分为3类:强变异(strong mutation)检测、弱变异(weak mutation)检测和固定变异(firm mutation)检测。

定义7(强变异检测) 给定被测程序p和基于程序p生成的一个变异体m,若测试用例t在程序p和变异体m上的输出结果不一致,则称测试用例t可以强变异检测到变异体m。

传统变异测试分析一般采用的是强变异检测方式。该定义最早由DeMillo等人提出[5]。随后为了优化变异体检测效率,Howden提出了弱变异检测方式[35],并定义如下。

定义8(弱变异检测) 假设被测程序p由n个程序实体构成$S=\{s_1, s_2, \ldots, s_n\}$。通过对程序实体$s_m$执行变异算子生成变异体$m$,若测试用例$t$在程序$p$和变异体$m$中的程序实体$s_m$上执行后的程序状态出现不一致,则称测试用例$t$可以弱变异检测到变异体$m$。

根据定义7和定义8可知,测试用例采用弱变异检测方式时,并不需要执行完所有的程序实体,从而提高了测试用例的检测效率。若测试用例弱变异检测到变异体,则该测试用例的执行满足可达性和必要性条件。若测试用例强变异检测到变异体,则该测试用例的执行还需额外满足充分性条件。因此两者之间存在包含关系,即若测试用例可以强变异检测到变异体,则一定可以弱变异检测到该变异体,反之则不然。Howden考虑了5种程序实体类型:变量引用、变量赋值、算术表达式、关系表达式和布尔表达式[35]。随后Offutt和Lee也提出了4种程序实体类型[36]:表达式首次执行后的评价(EX-WEAK/1)、语句的首次执行(ST-WEAK/1)、基本语句块的首次执行(BB-WEAK/1)和循环语句中基本语句块的n次迭代(BB-WEAK/N)。

弱变异检测方式的优点是不需要完整执行整个变异体,即一旦相关程序实体被执行后,就立刻可以判断该变异体是否被检测到。但由于不同程序实体间存在依赖关系,若用于评估测试用例集充分性,弱变异检测方式要弱于强变异检测方式。因此弱变异检测方式通过牺牲变异评分的精确性来提高变异测试分析的效率。研究人员对这种折衷展开了研究。

例如,Girgis和Woodward针对Fortran77程序语言实现了一个弱变异分析系统[37]。该系统通过分析程序内部状态来判断变异体是否被测试用例检测到,并且在实证研究中考虑了Howden提出的5种实体类型[35]中的4种,结果也验证了弱变异检测在计算开销上要小于强变异检测。Horgan和Mathur则通过理论分析表明,在特定条件下,弱变异检测得到的变异评分与强变异检测得到的变异评分保持一致[38]。Offutt和Lee同样实现了一个弱变异分析系统Leonardo[36]。他们的实证研究选择了22种Mothra变异操作,最终结果表明,在多数情况下弱变异检测可以替代强变异检测。

介于强变异检测和弱变异检测之间,Woodward和Halewood提出固定变异检测方式。该方式同时考虑了变异体执行的中间状态和最终输出结果[39]。随后Jackson和Woodward利用Java编程语言中的线程开发出一种并行固定变异检测方法[40]。

3.2.2 变异体编译优化

早期变异测试中变异体编译优化一般基于解释器,变异体采用解释执行方式输出执行结果。该类方式开销主要由解释开销构成。为了进行优化,King和Offutt[6]首先将原有程序转化为中间代码,然后对该中间代码进行变异和解释执行。该类技术具有较高的灵活性,可很好地适用于小规模程序,但却存在扩展性差问题,即当程序规模急速增加时,该类方法的执行开销也会急速增加。

随后Delamaro等人提出基于编译器的优化技术[7],这类技术将每个变异体均编译为可执行程序,然后测试用例在可执行程序上直接运行。与基于解释器的优化技术相比,这类技术在变异体执行上存在优势,因为编译执行的效率要显著优于解释执行。但当变异测试分析中变异体的编译时间大于执行时间时,这类优化方法将很难适用。DeMillo等人对传统的基于编译器的优化技术进行拓展[41]。他们认为,每个变异体相对于原有程序来说在语法层上仅存在微小差异,因此存在冗余编译开销。他们对编译器进行插桩,插桩后的编译器在编译原有程序时输出两个对象:原有程序的目标代码和变异体对应的一系列补丁。通过应用这些补丁,可以高效地将原有目标代码转化为变异体目标代码。

Untch等人基于程序模式(program schemata)法将所有变异体编码为一个元程序,其编译运行的开销要优于基于解释器的技术[42]。Ma等人采用变异体模式生成(mutant schemata generation)和字节码翻译技术[8]。在字节码翻译过程中,变异体基于原有程序的字节码,而不是基于源代码。基于字节码的变异体可以直接执行,从而省去了编译时间,同样该方法对于一些无法获知源代码的程序也能得到很好的使用。目前他们已经将该技术成功应用于Java编程语言,并开发出MuJava工具。但该方法也存在一些不足,首先该方法并不适用于所有编程语言。其次就Java编程语言而言,也不是所有变异算子均可适用于字节码级别。Mateo和Usaola对Untch等人的工作[42]进行了改进,提出了MUSIC(mutant schema improved with extra code)技术[43]。

Bogacki和Walter提出一种面向方面的变异方法来缩减编译开销[44]。他们通过设计一个方面补丁(aspect patch)来及时捕获方法输出。该方法需要执行两次补丁:第一次执行以获知结果和原有程序的上下文信息;第二次执行以生成并执行变异体。由此可见,该方法并不需要编译每一个变异体。

3.2.3 并行执行变异体

除了上述编译优化技术,研究人员也尝试借助高级计算机体系结构,采用并行执行方式来进一步提高变异测试分析效率。一些经典的研究工作包括:Mathur和Krauser借助向量处理机提高变异测试分析效率[45];Krauser等人在SIMD(single instruction multiple data)计算机上提出一种可并发执行变异体的方法[46];Fleyshgakker和Weiss提出一种可以提高并行变异测试分析的有效方法[47];Choi和Mathur[48],以及Offutt等人[49]借助MIMD(multiple instruction multiple data)计算机提高变异测试分析效率。

4 面向测试用例集充分性评估的变异测试研究

自20世纪70年代研究人员首次提出变异测试概念[4-5]以来,变异测试在不同应用领域中均获得成

功。最初的一种典型应用是作为测试充分性准则来评估和提高测试用例集的充分性,将该准则称之为变异准则。接下来从面向结构编程语言、面向对象编程语言、面向数据库应用、面向Web服务和面向方面等角度对已有的变异测试研究工作进行总结。

4.1 面向结构编程语言的变异测试研究

早期的变异测试研究集中关注的是面向结构编程语言,采用的标准评测数据集均基于Fortran编程语言。Budd等人于1978年首次针对Fortran IV编程语言设定了变异算子,并开发出变异测试工具PIMS[50]。但直到1987年,Offutt和King才对之前的研究工作进行系统总结,并针对Fortran77首次定义了22种变异算子,同时开发出变异测试工具Mothra[6]。这些变异算子及其描述如表2所示。可以将它们简单分为3类:(1)与语句分析相关,包括DER、GLR等变异算子;(2)与谓词分析相关,包括ABS、CRP等变异算子;(3)与偶然正确性(coincidental correctness)相关,包括AAR、ACR等变异算子。

期间研究人员对变异准则与数据流覆盖准则的包含关系进行了实证研究。例如,Mathur和Wong[23]在实证研究中发现,满足变异准则的测试用例集一般可以满足“all-use”覆盖准则,而反之则不然。该结果表明,变异准则一般包含“all-use”覆盖准则。Offutt等人[51]也通过实证研究验证了Mathur和Wong的结论,同时还对两个准则的缺陷检测能力进行了分析,发现变异准则生成的测试用例集比“all-use”准则可以多检测16%的缺陷。

随后研究人员对另一种重要的面向结构编程语言C语言展开了研究。Agrawal等人在1989年针对ANSI C编程语言规范设计了77种变异算子[52],这些变异算子可分为4类:变量变异、操作符变异、常量变异和语句变异。Delamaro和Maldonado随后开发出Proteum变异测试工具[7],该工具在Agrawal等人的研究工作基础上[52]实现了108种变异算子。Delamaro等人将变异测试从单元测试阶段转移到集成测试阶段[53]。他们从Agrawal定义的77种变异算子中选择出10种变异算子对程序接口进行了测试,这些变异算子主要考虑对公共函数签名植入缺陷。除了对C语言程序的功能属性进行变异测试分析外,也有研究人员对非功能属性进行变异测试分析,主要关注的对象是程序漏洞。经典研究工作包括:Shahriar和Zulkernine针对格式串缺陷(format string bugs)设计了8种变异算子[54]。Ghosh等人则将变异测试应用于自适应漏洞分析来检测缓冲区溢出漏洞[55]。

4.2 面向对象编程语言的变异测试研究

相对于面向结构编程语言,面向对象编程语言额外引入了封装、继承、多态和异常处理等特性。这些新的特征使得传统的变异算子难以完全适用于面向对象编程语言。

Kim等人[56]首先针对Java编程语言设计出20种变异算子,并将这20种变异算子分为6类:类型/变量、名称、类和接口声明、语句块、表达式和其他。随后他们引入了类变异分析概念[57],类变异分析主要针对Java语言中的面向对象语言特征进行变异测试,并从原有的20种变异算子中选择出其中的3种。在2000年,Kim等人又为类变异分析添加了10种变异算子[58]。在2001年,Kim等人最终为类变异分析确定了15种变异算子[59],这15种变异算子可分为4类:多态类型、方法重载类型、信息隐藏类型和异常处理类型。

Ma等人认为变异算子的设计应该考虑变异体的实际检测效果,与Kim等人[56-59]的研究工作不同,他们基于面向对象缺陷模型,设计出24种Java变异算子[8]。这些变异算子可以分为6类:信息隐藏类、继承类、多态类、重载类、与Java特有特征相关类和常见编程错误类。他们在实证研究中发现[60]:符合传统变异测试充分性的测试用例集,虽然可以检测出一些类变异缺陷,但却很难检测出一些基于EOA类变异算子和EOC类变异算子生成的变异体。Smith和Williams在Ma等人开发的MuJava变异工具[8]基础上,开发出基于Eclipse插件的变异测试工具MuClipse[9]。

研究人员也从其他角度对Java编程语言的变异测试进行了研究。例如,Alexander等人针对Java公共库(主要是Java容器库和迭代器库)设计了一组变异算子[61]。Bradbury等人基于Java并发环境对传统的变异算子进行了扩充[62]。与Java编程语言相似,

Derezinska针对C#程序设计了一组变异算子,并开发出工具CREAM[63]。

4.3 面向其他语言的变异测试研究

接下来主要总结面向数据库应用、面向Web服务测试和面向方面编程(aspect oriented programming,AOP)的变异测试。

变异测试已成功应用于数据库应用程序,主要检测与SQL语句相关的缺陷。研究人员结合SQL语句特征设计了多种变异算子。例如,Chan等人基于增强关系实体模型,针对SQL语句设计了7种变异算子[64]。Tuya等人针对SQL查询语句设计了另一套变异算子,并实现了变异测试工具SQLMutation。这套变异算子可分为4类:针对SQL语句的变异、针对条件和表达式中操作符的变异、处理控制的变异和针对标识符的变异[65]。Shahriar和Zulkernine针对数据库操作设计了9种变异算子,并实现了变异测试工具MUSIC[66]。Zhou和Frank基于Java数据库应用程序对变异测试进行了研究[67]。

变异测试也逐渐成功应用于Web服务测试。Lee和Offutt在2001年[68]首次将变异测试应用到Web服务中。他们提出交互规约模型对Web组件间的交互关系进行形式化描述,并基于该模型,提出一组通用变异算子对XML(extensible markup language)数据进行变异。随后Xu等人[69]针对XML数据变异问题提出新的策略,该策略不直接对XML数据进行变异,而是通过对XML Schema进行搅动来生成无效XML数据,并提出7种变异算子。同样Li和Miller也设计了一组基于XML Schema的变异算子[70]。除此之外,也有研究人员从XML语言自身特征出发对变异测试进行研究。例如,Lee等人针对OWL-S(ontology Web language for services)规约语言[71]设计变异算子,并检测相关语义错误。Estero-Botaro等人针对WS-BPEL(Web service business process execution language)规约语言[72]进行变异测试分析。姜瑛等人针对基于合约式设计的Web服务,设计了5种合约变异算子,同时提出一种可满足合约变异充分准则的测试用例自动生成技术[73]。

面向方面编程是一种新的编程范式,通过预编译方式和运行期动态代理,可以实现在不修改源代码的情况下完成程序功能的动态添加。Ferrari等人针对AOP设计了26种变异算子,这些变异算子可以分为3类:Pointcut表达式、Aspect声明和Advice的定义和实现[74]。AspectJ通过扩展Java编程语言实现了面向方面编程框架。Baekken和Alexander[75]、Anbalagan和Xie[76]均对AspectJ的变异测试进行了研究。

5 面向变异测试的测试用例生成技术

面向变异测试的测试用例生成研究集中关注的是设计出高质量测试用例来有效检测所有非等价变异体。本文将已有的工作分为3类:基于符号执行法、基于SBST法和基于混合法。最后对这些研究工作进行系统比较。

5.1 基于符号执行法的测试用例生成技术

这类技术可以进一步细分为传统的符号执行法和动态符号执行法。

最早的一种方法是基于约束的测试用例生成法(constraint based testing,CBT),该方法主要基于控制流分析和符号执行。Offutt于1988年在他的博士论文中首次对CBT展开了研究[77]。给定测试用例 t,被测程序 p 和在语句 s 上执行变异算子生成的变异体 m,若测试用例 t 可以检测到变异体 m,必须满足以下3个条件。

(1)可达性(reachability)条件:测试用例 t 执行时需要覆盖到变异语句 s。因为变异体 m 内其他语句与 p 相同,如果测试用例 t 不能执行 m 中的 s 语句,则 t 在 p 和 m 上的运行结果必然一致,即 t 不能检测到变异体 m。

(2)必要性(necessity)条件:测试用例 t 在执行完 p 和 m 的语句 s 后程序状态不一致。

(3)充分性(sufficiency)条件:执行完语句 s 后的不一致的程序状态将导致程序 p 和 m 的输出不一致。

这3个条件与Voas随后提出的PIE模型中的3个条件(即可达性、感染性和传播性)[78]保持一致。根据上述分析可以发现,设计出满足充分性条件(即强变异充分性准则)的测试用例的难度较高。目前绝大部分研究关注于寻找满足可达性和必要性条件(即

弱变异充分性准则)的测试用例,虽然这些测试用例不能确保满足充分性条件,但在实证研究中也能取得较高的变异评分。

DeMillo和Offutt在原有Mothra系统上开发出基于CBT的工具Godzilla[79]。该工具基于控制流分析、符号评价和约束满足求解技术对每一个变异体生成约束集并进行求解。结果表明,对于大多数程序来说,使用CBT技术生成的测试用例可以检测出其中98%的变异体。但是该技术的实际执行效果受到静态符号执行技术本身的约束,例如存在难以处理数组、循环结构和嵌套表达式的问题。针对上述不足,Offutt等人提出动态域约简(dynamic domain reduction)方法来解决该问题[80]。通过结合符号评价与域约简,可以有效解决数组和表达式问题,通过引入一个复杂的回溯搜索过程来强化测试用例的生成。刘新忠等人从数据依赖分析出发,基于域约简方法,提出一种基于约束的变异测试用例生成方法DRD[81]。单锦辉和刘明浩等人同样基于动态域约简提出一种新的变异测试用例自动生成方法[82-83]。已有研究工作仅针对单一变异体设计测试用例,利用检测同一位置的多个变异体条件相近的特点,对这些条件进行组合,然后设计出可同时检测多个变异体的有效测试用例生成方法。

动态符号执行法(dynamic symbolic execution,DSE)[84-86]最早由Sen等人提出,通过引入测试用例的具体执行,可以有效解决传统符号执行法的不足。其执行流程可以简要描述如下:首先随机生成一个测试用例,执行该测试用例并搜集对应执行轨迹上的路径条件(path condition)。接着对该路径条件上最后一个谓词执行取反操作,并生成一个新的路径条件,如果该路径条件并未出现过,则调用约束求解器进行求解,并生成新的测试用例,否则对路径条件上倒数第二个谓词进行取反操作。当不能生成新的路径条件时,该迭代过程将终止和结束。虽然动态符号执行法在理论上可以生成一系列测试用例来遍历被测程序内的所有可行路径,但在实际执行过程中却受到约束求解器求解能力的约束。目前基于DSE的测试用例生成工具包括DART[85]、CUT[84]和PEX[86]。该方法可以有效构造出满足可达性和必要性条件的测试用例,并且已经成功应用于面向变异测试的测试用例生成技术中。例如,张路等人基于Pex[86]开发出PexMutator工具[87],Papadakis等人将程序转化和DSE相结合,提出Mutation-DSE方法[88-89]。

5.2 基于SBST的测试用例生成技术

基于搜索的软件测试方法(search based software testing,SBST)同样可以指导生成测试用例[90-92]。典型算法包括爬山法、模拟退火、遗传算法(genetic algorithms)和禁忌搜索等。以遗传算法为例,该算法是一种全局搜索算法,主要受生物界进化规律(即适者生存和优胜劣汰机制)的启发。针对具体问题,遗传算法需要对候选解进行染色体编码和适应值函数设定。针对测试用例生成问题,候选测试用例可以直接编码为染色体,适应值函数的设定可以反映出染色体满足指定测试覆盖目标的情况。在生成单个测试用例时,从初始种群(一般由随机生成的多个测试用例构成)开始,不停进行种群演化,直至覆盖到目标测试需求或达到指定迭代次数为止。在每轮种群演化时,依次执行选择操作、交叉操作和变异操作可以形成新的种群。

Ayari等人提出面向变异测试的基于蚁群优化(ant colony optimization,ACO)的测试用例生成技术[93]。实证研究表明,该方法在变异评分和计算开销上要优于爬山法、遗传算法和随机搜索法。姚香娟和巩敦卫采用遗传算法提出一种基于路径比较的变异测试方法[94],该方法可以有效提高测试用例的生成效率。上述研究主要针对面向过程程序,Fraser和Zeller则针对面向对象程序展开深入研究[95],他们提出 μTEST方法,可以支持面向变异测试的测试用例和Oracle模块的构建。该方法选择了遗传算法,并结合面向对象程序的特征分别设定了染色体编码、适应值函数和变异、交叉和选择操作。

5.3 基于混合法的测试用例生成技术

已有研究工作仅针对弱变异充分性准则,Harman等人针对强变异充分性准则,提出一种基于DSE和SBST的混合测试用例生成方法SHOM[15]。并采用一阶变异体和高阶变异体对方法的有效性和执行效率

Table 4 Mutation-based test case generation

表4 面向变异测试的测试用例生成技术

作者和文献	发表年份	变异覆盖准则	测试用例生成技术	实验对象	代码最大行数/行	平均变异评分/(%)	变异体类型
DeMillo等[79]	1991	弱变异覆盖	静态域约简	5个Fortran程序	55	98	一阶
Offutt等[80]	1994	弱变异覆盖	动态域约简	12个Fortran程序	100	N/A	一阶
Zhang等[82-83]	2006	弱变异覆盖	动态域约简	5个C程序	21	95	一阶
刘新忠等[81]	2011	弱变异覆盖	动态域约简	5个Java程序	24	94	一阶
Zhang等[87]	2010	弱变异覆盖	DSE	5个C#程序	472	90	一阶
Papadakis等[88]	2010	弱变异覆盖	DSE	5个C程序	500	63	一阶
Papadakis等[89]	2010	弱变异覆盖	DSE	10个Java程序	100	90	一阶
Ayari等[93]	2007	弱变异覆盖	SBST	2个Java程序	72	88	一阶
Fraser等[95]	2012	弱变异覆盖	SBST	10个Java程序	388个类	75	一阶
姚香娟等[94]	2012	弱变异覆盖	SBST	8个C程序	173	N/A	一阶
Harman等[15]	2011	弱变异覆盖	DSE和SBST	18个C程序	9 564	69(一阶),71(二阶)	一阶和二阶

进行了评估。具体来说:SHOM方法首先采用DSE设计测试用例来满足弱变异测试覆盖准则,随后通过引入爬山法来尝试修改测试用例,使之满足测试充分性条件,即达到强变异测试覆盖准则。结果表明,相对于其他方法,SHOM生成的测试用例可以检测出更多的非等价变异体。

5.4 已有研究工作总结

该节对上述研究工作进行总结和比较,最终结果如表4所示。其中主要的比较因素包括:发表年份、变异覆盖准则、测试用例生成技术、实验对象个数和编程语言、实验对象中代码最大行数、平均变异评分和变异体类型等。其中N/A表示相关论文中并没有该项数据。从表中可以看出,目前DSE和SBST是主流的测试用例生成技术,同时实证研究中采用的实验对象逐渐从实验室小规模程序向工业界大规模程序过渡。

6 回归测试研究

变异测试除了可以评估和提高测试用例集的充分性,指导测试用例生成,还可以应用于回归测试。回归测试是软件开发和维护过程中保障软件质量的一种重要手段,用于保障代码修改的正确性,并避免代码修改对被测程序其他模块造成副作用。目前常见的回归测试技术包括:失效测试用例的识别和修复技术、测试用例选择技术、测试用例优先级排序技术、测试用例集约简技术和测试用例集扩充技术等。

当测试预算不足,以致不能执行完所有测试用例时,可以基于特定排序准则对测试用例的执行进行排序,这就是测试用例优先级问题。理想情况下,测试人员希望通过排序来最大化测试用例集的早期缺陷检测能力。Do和Rothermel将变异测试成功应用于该问题[3],在测试用例排序时考虑了测试用例对变异体的检测能力。结果表明,他们提出的排序策略可以有效提高测试用例集的缺陷检测率。

变异测试同样可以应用于测试用例集约简问题。该问题依据指定测试覆盖准则,可以进一步识别并移除冗余测试用例。Offutt等人提出一种Ping-Pong方法[96],其核心思想是首先针对某一测试覆盖准则生成变异体,然后从已有测试用例集中选择出拥有最高变异评分的测试用例子集。结果表明,该方法可以移除大约33%的测试用例,但并不影响其测试有效性。Usaola等人[97]提出一种基于贪婪法的测试用例集约简方法,将测试用例检测到的变异体信息作为测试用例的约简准则,该方法得到的约简后测试用例集与原有测试用例集拥有相同变异评分。

Zhang等人[98]针对回归测试场景,提出回归变异测试技术REMT来提高软件演化过程中的变异测试分析效率。该方法建立在回归测试用例选择技术基础上,基于演化前程序的变异测试结果,根据修改信息来增量计算演化后程序的变异测试结果。除此之外,他们还借助一个变异相关的测试用例优先级技术MTP来进一步提高效率。最终他们基于Javalanche[10]实现了REMT和MTP技术,并通过6个大规模开源软件,验证了该方法可以有效减小软件演化过程中的变异测试分析开销。

7 结束语

变异测试作为一种面向软件缺陷的测试技术,得到国内外研究人员的关注,并取得了大量研究成果。本文从变异测试原理、优化和应用3个角度对已有的研究工作进行了系统总结。

虽然目前针对变异测试的研究已经取得了大量的成果,但该领域仍存在很多研究点值得关注。可以考虑的研究点包括:

(1)变异测试中仍存在大量开放性问题。例如等价体变异检测问题,已有研究工作主要集中于从生成的变异体中识别出等价变异体。在将来的研究工作中可以考虑通过设计变异算子和分析被测程序特征,提出有效策略,在变异体生成过程中避免等价变异体的生成。

(2)在变异测试分析优化中,即变异体的生成、编译和执行过程中存在很多研究点。例如,在变异算子选择中可以考虑将变异算子按照变异体检测能力进行排序;借助硬件的发展和多核CPU的广泛使用,深入研究并行变异测试等。

(3)重点关注面向变异测试的测试用例生成技术。已有研究工作大部分关注于变异体的生成技术,但面向变异测试的测试用例生成研究还在起步阶段,并亟需成熟工具进行支持。可以进一步考虑将动态符号执行法和SBST进行有效组合,以提高测试用例的生成效率。

(4)进一步考虑变异测试研究成果与工业界测试流程的结合。目前的研究一般采用对照试验方式进行有效性评估,所得结论很难适用于软件企业的大型软件产品。其次研究人员提供的原型工具并未考虑与软件企业目前已有的开发和测试流程紧密结合,若要让企业接受学术界的研究成果,需要进一步提高测试工具的自动化程度,改善工具用户界面,并提高工具的鲁棒性。

References:

[1] Zhu H, Hall P, May J. Software unit test coverage and adequacy[J]. ACM Computing Survey, 1997, 29(4): 366-427.

[2] Andrews J H, Briand L C, Labiche Y. Is mutation an appropriate tool for testing experiments? [C]//Proceedings of the 27th International Conference on Software Engineering (ICSE '05), St Louis, Missouri, 2005. New York, NY, USA: ACM, 2005: 402-411.

[3] Do H, Rothermel G. On the use of mutation faults in empirical assessments of test case prioritization techniques[J]. IEEE Transactions on Software Engineering, 2006, 32(9): 733-752.

[4] Hamlet R G. Testing programs with the aid of a compiler[J]. IEEE Transactions on Software Engineering, 1977, 3(4): 279-290.

[5] DeMillo R A, Lipton R J, Sayward F G. Hints on test data selection: help for the practicing programmer[J]. Computer, 1978, 11(4): 34-41.

[6] King K N, Offutt A J. A Fortran language system for mutation based software testing[J]. Software: Practice and Experience, 1991, 21(7): 685-718.

[7] Delamaro M E, Maldonado J C. Proteum—a tool for the assessment of test adequacy for C programs[C]//Proceedings of the Conference on Performability in Computing Systems (PCS '96), New Brunswick, New Jersey, 1996: 79-95.

[8] Ma Y S, Offutt A J, Kwon Y R. MuJava: an automated class mutation system[J]. Software Testing, Verification and Reliability, 2005, 15(2): 97-133.

[9] Smith B, Williams L. On guiding the augmentation of an automated test suite via mutation analysis[J]. Empirical Software Engineering, 2009, 14(3): 341-369.

[10] Schuler D, Zeller A. Javalanche: efficient mutation testing for Java[C]//Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM

SIGSOFT Symposium on Foundations of Software Engineering (ESEC/FSE '09), Amsterdam, The Netherlands, 2009. New York, NY, USA: ACM, 2009: 297-298.

[11] Jia Yue, Harman M. An analysis and survey of the development of mutation testing[J]. IEEE Transactions on Software Engineering, 2011, 37(5): 649-678.

[12] Offutt A. Investigations of the software testing coupling effect[J]. ACM Transactions on Software Engineering and Methodology, 1992, 1(1): 5-20.

[13] Jia Yue, Harman M. Constructing subtle faults using higher order mutation testing[C]//Proceedings of the 8th IEEE International Working Conference on Source Code Analysis and Manipulation, Beijing, China, 2008: 249-258.

[14] Langdon W B, Harman M, Jia Yue. Efficient multi-objective higher order mutation testing with genetic programming[J]. Journal of Systems and Software, 2010, 83(12): 2416-2430.

[15] Harman M, Jia Yue, Landon W B. Strong higher order mutation-based test data generation[C]//Proceedings of the 19th ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE '11). New York, NY, USA: ACM, 2011: 212-222.

[16] Offutt J, Pan Jie. Automatically detecting equivalent mutants and infeasible paths[J]. Software Testing, Verification and Reliability, 1997, 7(3): 165-192.

[17] Offutt A J, Craft W M. Using compiler optimization techniques to detect equivalent mutants[J]. Software Testing, Verification and Reliability, 1994, 4(3): 131-154.

[18] Grun B, Schuler D, Zeller A. The impact of equivalent mutants[C]//Proceedings of the 2009 IEEE International Conference on Software Testing, Verification, and Validation Workshops (ICSTW '09), Denver, Colorado, 2009: 192-199.

[19] Baldwin D, Sayward F G. Heuristics for determining equivalence of program mutations[R]. Yale University, 1979.

[20] Hierons R M, Harman M, Danicic S. Using program slicing to assist in the detection of equivalent mutants[J]. Software Testing, Verification and Reliability, 1999, 9(4): 233-262.

[21] Adamopoulos K, Harman M, Hierons R M. How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution[C]//Proceedings of the Genetic and Evolutionary Computation Conference, Seattle, Washington, 2004: 1338-1349.

[22] Schuler D, Dallmeier V, Zeller A. Efficient mutation testing by checking invariant violations[C]//Proceedings of the 8th International Symposium on Software Testing and Analysis (ISSTA '09). New York, NY, USA: ACM, 2009: 69-80.

[23] Mathur A P, Wong W E. An empirical comparison of data flow and mutation-based test adequacy criteria[J]. Software Testing, Verification and Reliability, 1994, 4(1): 9-31.

[24] Wong W E, Mathur A P. Reducing the cost of mutation testing: an empirical study[J]. Journal of Systems and Software, 1995, 31(3): 185-196.

[25] Zhang Lu, Hou Shanshan, Hu Junjue, et al. Is operator-based mutant selection superior to random mutant selection?[C]//Proceedings of the 32nd International Conference on Software Engineering (ICSE '10). New York, NY, USA: ACM, 2010: 435-444.

[26] Barbosa E F, Maldonado J C, Vincenzi A M R. Toward the determination of sufficient mutant operators for C[J]. Software Testing, Verification and Reliability, 2001, 11(2): 113-136.

[27] Offutt A J, Lee A, Rothermel G, et al. An experimental determination of sufficient mutant operators[J]. ACM Transactions on Software Engineering and Methodology, 1996, 5(2): 99-118.

[28] Namin A S, Andrews J H, Murdoch D J. Sufficient mutation operators for measuring test effectiveness[C]//Proceedings of the 30th International Conference on Software Engineering (ICSE '08). New York, NY, USA: ACM, 2008: 351-360.

[29] Hussain S. Mutation clustering[D]. London, UK: King's College, 2008.

[30] Ji Changbin, Chen Zhenyu, Xu Baowen, et al. A novel method of mutation clustering based on domain analysis[C]//Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE 2009), Boston, Massachusetts, 2009: 1-3.

[31] Mathur A P. Performance, effectiveness, and reliability issues in software testing[C]//Proceedings of the 15th Annual International Computer Software and Applications Conference (COMPSAC '91), Tokyo, Japan, 1991: 604-605.

[32] Offutt A J, Rothermel G, Zapf C. An experimental evaluation of selective mutation[C]//Proceedings of the 15th International Conference on Software Engineering (ICSE '93), Tokyo, Japan, 1993: 100-107.

[33] Mresa E S, Bottaci L. Efficiency of mutation operators and selective mutation strategies: an empirical study[J]. Software Testing, Verification and Reliability, 1999, 9(4): 205-232.

[34] Polo M, Piattini M, Garcia-Rodriguez I. Decreasing the cost of mutation testing with second-order mutants[J]. Software Testing, Verification and Reliability, 2009, 19(2): 111-131.

[35] Howden W E. Weak mutation testing and completeness of test sets[J]. IEEE Transactions on Software Engineering, 1982, 8(4): 371-379.

[36] Offutt A J, Lee S D. An empirical evaluation of weak mutation[J]. IEEE Transactions on Software Engineering, 1994, 20(5): 337-344.

[37] Girgis M R, Woodward M R. An integrated system for program testing using weak mutation and data flow analysis[C]// Proceedings of the 8th International Conference on Software Engineering (ICSE '85), London, England, 1985: 313-319.

[38] Horgan J R, Mathur A P. Weak mutation is probably strong mutation[R]. Purdue University, 1990.

[39] Woodward M R, Halewood K. From weak to strong, dead or alive? An analysis of some mutation testing issues[C]//Proceedings of the 2nd Workshop on Software Testing, Verification, and Analysis, Banff Albert, Canada, 1988: 152-158.

[40] Jackson D, Woodward M R. Parallel firm mutation of Java programs[C]//Proceedings of the 1st Workshop on Mutation Analysis, San Jose, California, 2001: 55-61.

[41] DeMillo R A, Krauser E W, Mathur A P. Compiler-integrated program mutation[C]//Proceedings of the 15th Annual International Computer Software and Applications Conference (COMPSAC '91), Tokyo, Japan, 1991: 351-356.

[42] Untch R H, Offutt A J, Harrold M J. Mutation analysis using mutant schemata[C]//Proceedings of the 1993 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '93), Cambridge, Massachusetts, 1993. New York, NY, USA: ACM, 1993: 139-148.

[43] Mateo P R, Usaola M P. Mutant execution cost reduction: through MUSIC (mutant schema improved with extra code)[C]// Proceedings of the 2012 IEEE 5th International Conference on Software Testing, Verification and Validation (ICST '12), Montreal, Canada, 2012. Washington, DC, USA: IEEE Computer Society, 2012: 664-672.

[44] Bogacki B, Walter B. Evaluation of test code quality with aspect-oriented mutations[C]//Proceedings of the 7th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP '06). Berlin, Heidelberg: Springer-Verlag, 2006: 202-204.

[45] Mathur A P, Krauser E W. Mutant unification for improved vectorization[R]. Purdue University, 1988.

[46] Krauser E W, Mathur A P, Rego V J. High performance software testing on SIMD machines[J]. IEEE Transactions on Software Engineering, 1991, 17(5): 403-423.

[47] Fleyshgakker V N, Weiss S N. Efficient mutation analysis: a new approach[C]//Proceedings of the 1994 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '94), Seattle, Washington, 1994. New York, NY, USA: ACM, 1994: 185-195.

[48] Choi B, Mathur A P. High-performance mutation testing[J]. Journal of Systems and Software, 1993, 20(2): 135-152.

[49] Offutt A J, Pargas R P, Fichter S V, et al. Mutation testing of software using a MIMD computer[C]//Proceedings of the 1992 International Conference on Parallel Processing (ICPP '92), Chicago, Illinois, 1992: 255-266.

[50] Budd T A, DeMillo R A, Lipton R J, et al. The design of a prototype mutation system for program testing[C]//Proceedings of the AFIPS National Computer Conference, Anaheim, New Jersey, 1978: 623-627.

[51] Offutt A J, Pan Jie, Tewary K, et al. An experimental evaluation of data flow and mutation testing[J]. Software: Practice and Experience, 1996, 26(2):165-176.

[52] Agrawal H, DeMillo R A, Hathaway B, et al. Design of mutant operators for the C programming language[R]. Purdue University, 1989.

[53] Delamaro M E, Maldonado J C, Mathur A P. Interface mutation: an approach for integration testing[J]. IEEE Transactions on Software Engineering, 2001, 27(3): 228-247.

[54] Shahriar H, Zulkernine M. Mutation-based testing of format string bugs[C]//Proceedings of the 11th IEEE High Assurance Systems Engineering Symposium (HASE '08), Nanjing, China, 2008. Washington, DC, USA: IEEE Computer Society, 2008: 229-238.

[55] Ghosh A K, Connor T, McGraw G. An automated approach for identifying potential vulnerabilities in software[C]//Proceedings of the 1998 IEEE Symposium on Security and Privacy, Oakland, California, 1998: 104-114.

[56] Kim S, Clark J A, McDermid J A. The rigorous generation of Java mutation operators using Hazop[C]//Proceedings of the 12th International Conference on Software and Systems

Engineering and Their Applications (ICSSEA ’99), Paris, France, 1999.

[57] Kim S, Clark J A, McDermid J A. Assessing test set adequacy for object oriented programs using class mutation[C]//Proceedings of the 3rd Symposium on Software Technology (SoST ’99), Buenos Aires, Argentina, 1999.

[58] Kim S, Clark J A, McDermid J A. Class mutation: mutation testing for object-oriented programs[C]//Proceedings of the Net Object Days Conference on Object-Oriented Software Systems, 2000.

[59] Kim S, Clark J A, McDermid J A. Investigating the effectiveness of object-oriented testing strategies using the mutation method[C]//Proceedings of the 1st Workshop on Mutation Analysis, San Jose, California, 2001: 207-225.

[60] Ma Y S, Harrold M J, Kwon Y R. Evaluation of mutation testing for object-oriented programs[C]//Proceedings of the 28th International Conference on Software Engineering (ICSE ’06), Annapolis, Maryland, 2006. New York, NY, USA: ACM, 2006: 869-872.

[61] Alexander R T, Bieman J M, Ghosh S, et al. Mutation of Java objects[C]//Proceedings of the 13th International Symposium on Software Reliability Engineering (ISSRE ’02), Annapolis, Maryland, 2002. Washington, DC, USA: IEEE Computer Society, 2002: 341-351.

[62] Bradbury J S, Cordy J R, Dingel J. Mutation operators for concurrent Java (J2SE 5.0)[C]//Proceedings of the 2nd Workshop on Mutation Analysis, Raleigh, North Carolina, 2006: 83-92.

[63] Derezinska A. Quality assessment of mutation operators dedicated for C# programs[C]//Proceedings of the 6th International Conference on Quality Software (QSIC ’06), Beijing, China, 2006. Washington, DC, USA: IEEE Computer Society, 2006: 227-234.

[64] Chan W K, Cheung S C, Tse T H. Fault-based testing of database application programs with conceptual data model[C]// Proceedings of the 5th International Conference on Quality Software (QSIC ’05), Washington, USA, 2005. Washington, DC, USA: IEEE Computer Society, 2005: 187-196.

[65] Tuya J, Cabal M, Riva C. Mutating database queries[J]. Information and Software Technology, 2007, 49(4): 398-417.

[66] Shahriar H, Zulkernine M. MUSIC: mutation-based SQL injection vulnerability checking[C]//Proceedings of the 8th International Conference on Quality Software (QSIC ’08), Oxford, UK, 2008. Washington, DC, USA: IEEE Computer Society, 2008: 77-86.

[67] Zhou Chixiang, Frank P. Mutation testing for Java database applications[C]//Proceedings of the 2009 International Conference on Software Testing, Verification and Validation (ICST ’09). Washington, DC, USA: IEEE Computer Society, 2009: 396-405.

[68] Lee S, Offutt J. Generating test cases for XML-based Web component interactions using mutation analysis[C]//Proceedings of the 12th International Symposium on Software Reliability Engineering (ISSRE ’01). Washington, DC, USA: IEEE Computer Society, 2001: 200-209.

[69] Xu Wuzhi, Offutt J, Luo Juan. Testing Web services by XML perturbation[C]//Proceedings of the 16th International Symposium on Software Reliability Engineering (ISSRE ’05), Chicago, Illinois, 2005. Washington, DC, USA: IEEE Computer Society, 2005: 257-266.

[70] Li Jianbing, Miller J. Testing the semantics of W3C XML schema[C]//Proceedings of the Annual International Computer Software and Applications Conference, Turku, Finland, 2005: 443-448.

[71] Lee Shufang, Bai Xiaoying, Chen Yinong. Automatic mutation testing and simulation on OWL-S specified Web services[C]//Proceedings of the 41st Annual Simulation Symposium (ANSS ’08), Ottawa, Canada, 2008. Washington, DC, USA: IEEE Computer Society, 2008: 149-156.

[72] Estero-Botaro A, Palomo-Lozano F, Medina-Bulo I. Mutation operators for WS-BPEL 2.0[C]//Proceedings of the 21st International Conference on Software and Systems Engineering and Their Applications (ICSSEA ’08), Paris, France, 2008.

[73] Jiang Ying, Xin Guomao, Shan Jinhui, et al. A method of automated test data generation for Web service[J]. Chinese Journal of Computers, 2005, 28(4): 568-577.

[74] Ferrari F C, Maldonado J C, Rashid A. Mutation testing for aspect-oriented programs[C]//Proceedings of the 2008 International Conference on Software Testing, Verification and Validation (ICST ’08), Monterey, California, 2008. Washington, DC, USA: IEEE Computer Society, 2008: 52-61.

[75] Baekken J S, Alexander R T. A candidate fault model for AspectJ pointcuts[C]//Proceedings of the 17th International Symposium on Software Reliability Engineering (ISSRE ’06), Raleigh, North Carolina, 2006. Washington, DC, USA: IEEE Computer Society, 2006: 169-178.

[76] Anbalagan P, Xie Tao. Automated generation of pointcut mutants for testing pointcuts in AspectJ programs[C]//Proceedings of the 19th International Symposium on Software Reliability Engineering (ISSRE ’08), Redmond, Washington, 2008. Washington, DC, USA: IEEE Computer Society, 2008: 239-248.

[77] Offutt A J. Automatic test data generation[D]. Atlanta: Georgia Institute of Technology, 1988.

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

[79] DeMillo R A, Offutt A J. Constraint-based automatic test data generation[J]. IEEE Transactions on Software Engineering, 1991, 17(9): 900-910.

[80] Offutt A J, Jin Zhenyi, Pan Jie. The dynamic domain reduction procedure for test data generation[J]. Software: Practice and Experience, 1999, 29(2): 167-193.

[81] Liu Xinzhong, Xu Gaochao, Hu Liang, et al. An approach for constraint-based test data generation in mutation testing[J]. Journal of Computer Research and Development, 2011, 48(4): 617-627.

[82] Liu Minghao, Gao Youfeng, Shan Jinhui, et al. An approach to test data generation for killing multiple mutants[C]//Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM ’06), Pennsylvania, USA, 2006. Washington, DC, USA: IEEE Computer Society, 2006: 113-122.

[83] Shan Jinhui, Gao Youfeng, Liu Minghao, et al. A new approach to automated test data generation in mutation testing[J]. Chinese Journal of Computers, 2008, 31(6): 1025-1034.

[84] Sen K, Marinov D, Agha G. CUTE: a concolic unit testing engine for C[C]//Proceedings of the 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE ’05). New York, NY, USA: ACM, 2005: 263-272.

[85] Godefroid P, Klarlund N, Sen K. DART: directed automated random testing[C]//Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI ’05). New York, NY, USA: ACM, 2005: 213-223.

[86] Tillmann N, Halleux J. Pex-white box test generation for .NET[C]//Proceedings of the 2nd International Conference on Tests and Proofs (TAP ’08). Berlin, Heidelberg: Springer-Verlag, 2008: 134-153.

[87] Zhang Lingming, Xie Tao, Zhang Lu, et al. Test generation via dynamic symbolic execution for mutation testing[C]//Proceedings of the 2010 IEEE International Conference on Software Maintenance (ICSM ’10), Timisoara, Romania, 2010: 1-10.

[88] Papadakis M, Malevris N. Automatic mutation test case generation via dynamic symbolic execution[C]//Proceedings of the 21st International Symposium on Software Reliability Engineering (ISSRE ’10). Washington, DC, USA: IEEE Computer Society, 2010: 121-130.

[89] Papadakis M, Malevris N, Kallia M. Towards automating the generation of mutation tests[C]//Proceedings of the 5th Workshop on Automation of Software Test (AST ’10), Cape Town, South Africa, 2010. New York, NY, USA: ACM, 2010: 111-118.

[90] Ali S, Briand L C, Hemmati H, et al. A systematic review of the application and empirical investigation of search-based test case generation[J]. IEEE Transactions on Software Engineering, 2009, 36(6): 742-762.

[91] Harman M, Jones B F. Search-based software engineering[J]. Information and Software Technology, 2001, 43(14): 833-839.

[92] McMinn P. Search-based software test data generation: a survey[J]. Software Testing, Verification, and Reliability, 2004, 14(2): 105-156.

[93] Ayari K, Bouktif S, Antoniol G. Automatic mutation test input data generation via ant colony[C]//Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation (GECCO ’07). New York, NY, USA: ACM, 2007: 1074-1081.

[94] Yao Xiangjuan, Gong Dunwei. Mutation testing based on comparison of paths[J]. Chinese Journal of Electronics, 2012, 40(1): 103-108.

[95] Fraser G, Zeller A. Mutation-driven generation of unit tests and Oracles[J]. IEEE Transactions on Software Engineering, 2012, 38(2): 278-292.

[96] Offutt A J, Pan Jie, Voas J M. Procedures for reducing the size of coverage-based test sets[C]//Proceedings of the 12th International Conference on Testing Computer Software, Washington, DC, USA, 1995: 111-123.

[97] Usaola M P, Mateo P R, Lamancha B P. Reduction of test suites using mutation[C]//Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering (FASE '12), Tallinn, Estonia, 2012. Berlin, Heidelberg: Springer-Verlag, 2012: 425-438.

[98] Zhang Lingming, Marinov D, Zhang Lu, et al. Regression mutation testing[C]//Proceedings of the 2012 International Symposium on Software Testing and Analysis (ISSTA '12), Minneapolis, MN, USA, 2012. New York, NY, USA: ACM, 2012: 331-341.

附中文参考文献：

[73] 姜瑛, 辛国茂, 单锦辉, 等. 一种Web服务的测试数据自动生成方法[J]. 计算机学报, 2005, 28(4): 568-577.

[81] 刘新忠, 徐高潮, 胡亮, 等. 一种基于约束的变异测试数据生成方法[J]. 计算机研究与发展, 2011, 48(4): 617-627.

[83] 单锦辉, 高友峰, 刘明浩, 等. 一种新的变异测试数据自动生成方法[J]. 计算机学报, 2008, 31(6): 1025-1034.

[94] 姚香娟, 巩敦卫. 基于路径比较的变异测试方法[J]. 电子学报, 2012, 40(1): 103-108.

CHEN Xiang was born in 1980. He received his Ph.D. degree from Nanjing University in 2011. Now he is a lecturer at Nantong University, and the member of CCF. His research interests include software testing and program analysis.
陈翔(1980—),男,江苏南通人,2011年于南京大学获得博士学位,现为南通大学计算机科学与技术学院讲师,CCF会员,主要研究领域为软件测试,程序分析。

GU Qing was born in 1972. He is a professor and Ph.D. supervisor at Nanjing University. His research interest is software testing.
顾庆(1972—),男,江苏常州人,南京大学教授、博士生导师,主要研究领域为软件测试。