基于数据流准则的测试用例生成方法

陈洁琼 姜淑娟 张争光
(中国矿业大学计算机科学与技术学院 徐州 221116)

摘 要 针对基于控制流的测试充分性准则易错失面向对象程序中的状态依赖关系,提出一种基于数据流准则的测试用例自动生成方法。通过数据流分析获取待覆盖的目标定义使用对,利用遗传算法自动生成覆盖定义使用对的测试用例,根据适应度函数指导测试用例的进化。将该方法与基于分支覆盖和语句覆盖的方法相比较。实验结果表明,与其他方法相比,该方法可以检测出更多的变异体,适应度函数的设计降低了进化代数。

关键词 面向对象程序,数据流准则,测试用例生成,适应度函数
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.02.015

Approach for Test Case Generation Based on Data Flow Criterion

CHEN Jie-qiong JIANG Shu-juan ZHANG Zheng-guang
(School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China)

Abstract Control flow criterion may miss the state dependent relations in object oriented program easily. This paper presented an approach for automatic test case generation based on data flow criterion, using data flow analysis to get definition use pairs that test suite should cover, using genetic algorithm to generate test suite automatically and evolving the test cases according to fitness function. The experimental results indicate that the test cases generated by our approach can detect more mutants comparing with approaches based on branch and statement criterion, and fitness function designed in our approach makes the number of generations decreased.

Keywords Object oriented program, Data flow criterion, Test case generation, Fitness function

1 引言

软件测试是保障软件可靠性的重要手段。根据考虑的代码元素的不同,结构性测试准则可以分为基于控制流的测试充分性准则和基于数据流的测试充分性准则[1-2]。其中基于控制流的测试充分性准则,即控制流准则考虑如语句、分支、判定、条件等控制流元素;基于数据流的测试充分性准则,即数据流准则考虑的是方法内或方法间的数据流交互关系。

目前测试用例生成方法主要研究的是基于控制流的测试充分性准则,如语句、分支等。数据流准则的复杂度比简单控制流准则高,有关数据流准则的研究较少。但是,数据流信息可以很好地获取面向对象程序中的状态依赖关系,而基于控制流的准则易错失这些关系[3-5]。因此在面向对象的程序测试过程中,数据流准则是一个新的研究热点。

在面向对象的系统中,存在方法的执行依赖于实例状态的情况,需要在执行方法前调用任何改变这个实例状态的公有方法[6]。图1是一个实现简易饮料自动售货机的面向对象的程序。其中第一列是行号,第二列是基本块编号。这段代码展示了一个方法间的状态依赖关系。在类 DrinkMachine 中,方法 addMoney 定义了变量 *paid*(第16行),方法 getDrink 使用了该变量(第7行),这两个方法可能存在交互关系。经典的数据流覆盖准则需要一个测试用例来调用方法 addMoney,并最终调用方法 getDrink。后一个方法对变量 *paid* 的计算依赖于前一个方法对 *paid* 的赋值,如果方法 addMoney 赋给 *paid* 一个可能错误的值,这个值传播到方法 getDrink 中使用时,数据流覆盖准则会增加这个错误值对使用产生影响的机会,在该变量的使用中表现为一个错误。而语句覆盖或分支覆盖无法获取这种由状态依赖关系产生的方法间的交互。

```
1  1  public class DrinkMachine {
2  1      drink dr;
3  1      int paid;
4  1      int addnum;
5  2  public void getDrink() {
6  2      boolean cangetDrink=true;
7  3      if (paid < dr.price || dr.num <= 5) {
```

到稿日期:2015-11-12 返修日期:2016-02-22 本文受国家自然科学基金(61502497),广西可信软件重点实验室研究课题(kx201530),南京大学计算机软件新技术国家重点实验室开放课题(KFKT2014B19)资助。
陈洁琼(1993—),女,硕士生,主要研究领域为软件测试、测试数据生成等,E-mail:JieqiongChen@cumt.edu.cn;姜淑娟(1966—),女,博士,教授,博士生导师,主要研究领域为编译技术、软件工程等,E-mail:shjjiang@cumt.edu.cn;张争光(1994—),男,硕士生,主要研究领域为软件测试、测试数据生成等。

```
8  4          cangetDrink=false;
9  4          dr. supply(addnum);
10 5      if (cangetDrink) {
11 6          paid -= dr. price;
12 6      }
13 6    }
14 6  }
15 7  public void addMoney(int money) {
16 7      paid += money;
17 7    }
18 7}
```

图 1 饮料自动售货机的部分代码

基于数据流准则的测试用例自动生成方法将定义使用对(definition-use pair,dup)作为测试目标[7],通过测试用例驱动程序的执行,对程序进行插桩并监控 dup 的覆盖情况[8],利用遗传算法实现测试用例的自动生成,其中适应度函数的设计对指导测试用例的进化起着关键作用。Girgis[9]和 Nayak[10]设计的适应度函数是已覆盖的 dup 路径的个数和所有 dup 路径的个数的比值,无法对测试用例的进化起到较好的指导作用。

针对以上问题,本文提出一种基于数据流准则的测试用例自动生成方法。首先生成测试程序的控制流图(CFG)以及过程间控制流图(ICFG)来识别程序中的测试目标 dup;然后利用遗传算法自动生成覆盖 dup 集合的测试用例。该方法利用数据流准则识别覆盖目标 dup 集合,适合获取面向对象程序中的状态依赖关系。其中适应度函数综合考虑了所有的 dup 来计算适应度值,一次覆盖所有的覆盖目标,降低了进化代数。

2 背景知识

定义 1(控制流图(CFG))[8] CFG 是一个有向图,其中每个节点表示代码的一个基本块,每条可能带有标签的边表示块之间的控制流,一个 CFG 有一个入口节点和一个出口节点分别表示程序的入口和出口。图 2 是一个求最大、最小值的程序,图中第一列数字表示行号,第二列表示基本块编号。图 3 是该程序的 CFG。

```
1  1  public class MaxMin {
2  1    int max=0;
3  1    int min=0;
4  2  public void MaxMin(int a,int b,int c) {
5  2    if(a>b){
6  3      max=a;min=b;}
7  4    else{
8  4      max=b;min=a;}
9  5    if(max<c){
10 6      max=c;}
11 7    if(min>c){
12 8    min=c;}}}
```

图 2 计算最大、最小值的代码

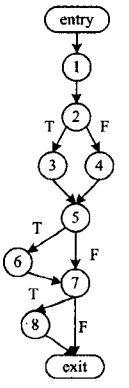

图 3 MaxMin 的控制流图

定义 2(定义使用对(dup))[6] dup 由相同变量的一个定义和一个使用组成。假设有一个变量 v,如果一条语句是对 v 的赋值,则称这条语句是 v 的一个定义;如果一条语句的执行需要 v 的值,则称这条语句是 v 的一个使用。变量的使用包含两种类型:1)计算使用(c-use),v 的值被用于计算语句或者输出语句;2)谓词使用(p-use),v 的值被用于一个谓词语句。从 v 的一个定义节点到 v 的一个使用节点的路径上,如果存在一个节点重新定义 v,就称这个节点为 v 的一个 kill 节点。如果一条程序路径遍历了 v 的一个定义并到达 v 的一个使用,且该路径上没有 kill 节点,就构成 v 的一个 dup。本文用一个三元组来表示这种关系:$dup=(d,u,v)$,d 是包含 v 的一个定义的 CFG 节点,u 是包含 v 的一个使用的节点或者分支。如图 3 中(3,5T,max)就是变量 max 的一个 dup。

3 本文方法

为了获取面向对象程序中的状态依赖关系,实现测试用例的自动生成,提出一种基于数据流准则的测试用例自动生成方法。该方法主要分为两个阶段:

1)数据流分析。利用 CFG 和 ICFG 识别出 dup 集合,将该集合添加到测试用例需要覆盖的目标中。

2)适应度函数设计。针对所有覆盖目标 dup 集合设计一个适应度函数来计算适应度值并指导测试用例的进化。

3.1 数据流分析

对于每个测试类,生成该类中各个方法的 CFG,然后生成过程间控制流图(ICFG)[11]连接类中的方法。图 1 中程序的部分 ICFG 如图 4 所示。

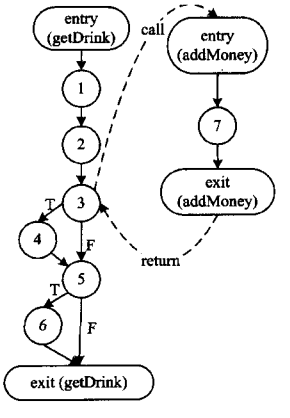

图 4 类 DrinkMachine 的部分 ICFG

本文利用经典的数据流方程式进行数据流的分析，如式(1)所示：

$$out[S]=gen[S]\cup(in[S]-kill[S]) \tag{1}$$

其中，$gen[S]$表示在句的末尾或者在句中得到的定义，$in[S]$表示进入语句 S 的开始点时携带的定义，$kill[S]$表示被语句 S 杀死的定义，$out[S]$表示可以到达语句 S 末尾的定义。

数据流分析考虑了局部变量、实例变量和静态变量。应用标准的数据流迭代算法[11-12]分析程序中的 dup。主要分为 3 个步骤：

1)利用标准的前向分析识别出可能会到达控制流图中的一个节点的变量的定义；

2)对于每个节点中的定义，识别出这个节点中相同变量的使用并创建相应的 dup；

3)将识别出的 dup 添加到测试用例需要覆盖的目标中。

3.2　适应度函数设计

通过前面的数据流分析得到一组覆盖目标，即 dup 集合。本文利用遗传算法自动生成覆盖这些目标的测试用例，根据适应度函数指导测试用例的进化。dup 可以表示为点到点的适应度函数[13]，用于衡量一个测试用例到 CFG 中的一个节点的距离[14]，如式(2)所示：

$$nodeDis=ap+v(dis) \tag{2}$$

其中，ap 表示接近程度(approach level)，如果测试用例执行的路径覆盖了这个节点，则 ap 的值为 0；反之，ap 的值为 1。dis 表示分支距离。例如，有一个分支谓词 $a>b$，那么其分支距离函数为 $b-a$。分支距离的计算如式(3)所示：

$$dis=\begin{cases}0, & b-a<0\\ |b-a|+1, & b-a\geqslant 0\end{cases} \tag{3}$$

当 $b-a\geqslant 0$ 时，分支距离的值加上一个常数 1 来处理边界值的情况。如果 $b=a$，不满足该分支条件，但是其分支距离为 0，所以本文加上一个常数 1 来处理这种情况。为了避免 dis 的数值过大使得 ap 值的指导作用变弱，对 dis 的值进行归一化的处理，如式(4)所示：

$$v(dis)=dis/(dis+1) \tag{4}$$

假设覆盖目标中有 n 个 dup，测试用例集 T 中有 m 个测试用例。$fitness(i,T)$表示测试用例集 T 覆盖第 i 个 dup 时的适应度函数值，$0<i\leqslant n$，如式(5)所示：

$$fitness(i,T)=\min(nodeDis_j(def_i)+nodeDis_j(use_i)+k) \tag{5}$$

其中，j 表示测试用例集 T 中第 j 个测试用例，$0<j\leqslant m$。$nodeDis_j(def_i)$表示第 j 个测试用例到第 i 个 dup 的定义节点的距离；$nodeDis_j(use_i)$表示第 j 个测试用例到第 i 个 dup 的使用节点的距离；k 是一个惩罚值，如果测试用例集 T 中的执行路径上存在第 i 个 dup 的 kill 节点，则设置 k 为 1；如果不存在 kill 节点，则设置 k 为 0。分别计算每个测试用例到第 i 个 dup 的适应度值，比较得出最小值作为测试用例集 T 在第 i 个 dup 上的适应度值。

本文综合考虑了所有识别出的 dup 集合，对每个 dup 上测试用例计算得出的适应度值进行求和，一次覆盖所有的目标。适应度函数如式(6)所示：

$$fit(T)=\sum_{1}^{n}fitness(i,T) \tag{6}$$

其中，n 为覆盖目标中 dup 的个数，i 为覆盖目标中第 i 个 dup，$0<i\leqslant n$。$fit(T)$为 0 时，表示该测试用例集 T 覆盖了所有的目标。

例如，假设图 2 中的程序的一个测试用例集 T 为{(1,4,3)(5,3,2)}，T_1(1,4,3)的执行路径为(2F,5F,7F)，T_2(5,3,2)的执行路径为(2T,5F,7T)，由于执行路径中不存在变量 max 的 kill 节点，因此 k 为 0。那么对于覆盖目标中的一个 dup_1(3,5T,max)，其适应度值为：

$$\begin{aligned}fitness(1,T_1)&=node_1(def_1)+node_1(use_1)+k\\&=1+(b-a+1)/((b-a+1)+1)+0+\\&\quad(max-c+1)/((max-c+1)+1)+0\\&=1+(4-1+1)/(4-1+1+1)+0+(4-\\&\quad3+1)/(4-3+1+1)+0\\&=2.47\end{aligned}$$

$$\begin{aligned}fitness(1,T_2)&=node_2(def_1)+node_2(use_1)+k\\&=0+((b-a)+1)/((b-a+1)+1)+0+\\&\quad(max-c+1)/((max-c+1)+1)+0\\&=0+0+0+(5-2+1)/(5-2+1+1)+0\\&=0.8\end{aligned}$$

比较选出最小值，$fit(1,T)=0.8$。

类似地，计算测试用例相对于覆盖目标中其他 dup 的适应度值，将所有的适应度值进行求和得出最终的适应度值 $fit(T)$。

4　实验

4.1　实验对象

本文利用 10 个测试程序对方法进行评估。其中 3 个是简单的面向过程的程序，TriangleType 用于判断三角形类型；MaxMin 用于计算最大值和最小值；NextDate 用于判断是否为闰年。还有 7 个面向对象的程序，其中 DrinkMachine 是图 1 中的程序；其他测试程序均来自文献[6]。

表 1 描述了 10 个测试程序的信息。从表 1 中可以看出，NextDate 中识别出的 dup 的个数少于分支个数，其他程序中识别出的 dup 的个数均多于分支个数和语句个数。

表 1　程序信息统计

编号	测试程序	dup	分支	语句
pro1	TriangleType	23	17	20
pro2	MaxMin	14	6	12
pro3	NextDate	24	41	10
pro4	DrinkMachine	21	9	18
pro5	ListChannel	43	17	32
pro6	DefaultDateSet	16	16	26
pro7	FoodList	98	29	53
pro8	ProductDetails	258	130	292
pro9	BattleStatistics	475	156	235
pro10	HandballModel	348	317	442

4.2　实验设计

本文采用变异分析[15]对测试用例进行评估。手动在程序中植入错误，如果存在一个断言可以区分原始版本的类和变异后的类的执行，就认为这个变异体被一个测试用例杀死了(比如测试用例在一个版本上的运行失败，但是在另一个版本的运行通过)，即测试用例检测出了这个变异体。本文采用

变异率评估测试用例，其计算如式(7)所示：

$$变异率=\frac{m_{kill}}{m} \tag{7}$$

其中，m_{kill} 表示被测试用例杀死的变异体个数，m 表示总共的变异体个数。变异率越大，说明测试用例检测出的变异体越多，测试用例越好。

为了进一步研究本文方法，设计了 2 个问题：

问题 1 相比于其他方法，本文方法是否更适合测试面向对象的程序？

数据流准则更适合获取面向对象的程序中的状态依赖关系，控制流准则易错失这些依赖关系。为了验证本文方法是否更适合于测试面向对象的程序，设计了 4.3.1 节的实验。

问题 2 本文设计的适应度函数是否优于其他文献的适应度函数？

本文设计的适应度函数针对每个 dup 对测试用例进行评价，并综合考虑所有的 dup，对每个 dup 求出的适应度值进行求和。本文的适应度函数相比于其他文献的适应度函数是否能更好地指导测试用例的进化，为此本文设计了 4.3.2 节的实验。

4.3 实验结果

通过两个实验对本文方法进行进一步研究，并对实验结果作出讨论。

4.3.1 变异率的比较

图 5 中分别记录了分支、语句以及 dup 覆盖生成的测试用例的变异率。由于 dup 覆盖生成的测试用例数目多于语句覆盖和分支覆盖，因此本文利用随机测试(RT)为每个程序随机生成 100 个测试用例。结果显示，在 TriangleType 和 MaxMin 中，4 种类型的测试用例的变异率都为 100%；在 NextDate 程序中，分支覆盖、语句覆盖以及 dup 覆盖的变异率差别较小，RT 的变异率较低。对 7 个面向对象的测试程序计算所有程序变异率的平均值，得出 RT 的平均变异率为 41.5%，分支覆盖的平均变异率为 59.6%，语句覆盖的平均变异率为 58.8%，dup 覆盖的平均变异率为 68.1%。虽然 RT 生成的测试用例数目多于其他 3 种覆盖方式，但是 RT 的变异率普遍偏低，这说明测试用例的优劣和测试用例的数目之间的关联较小。

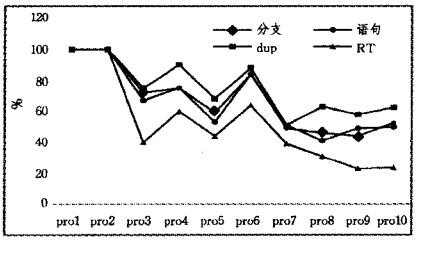

图 5 4 种测试用例的变异率

4.3.2 两种适应度函数的比较

本文实现了文献[10]中的适应度函数，如式(8)所示：

$$fit=\frac{一个测试用例已覆盖的定义使用路径的个数}{所有定义使用路径的个数} \tag{8}$$

分别利用式(6)和式(8)中的适应度函数指导测试用例的进化，并记录每个测试程序需要的进化代数。进化代数越小，表明适应度函数的指导作用越好。

本文将每个程序的测试用例生成时间设置为 2 分钟，超过 2 分钟算法终止，2 分钟内算法迭代的次数即进化代数。由于在面向对象程序中测试用例的适应度函数的值达到 0 较为困难，因此本文在统计程序的进化代数时，只统计每个程序的测试用例集合的适应度函数值达到最小值时的进化代数。结果如表 2 所列。

表 2 两种适应度函数进化代数的比较

测试程序	Fit1	Fit2
TriangleType	16	5
MaxMin	22	7
NextDate	35	7
DrinkMachine	457	292
ListChannel	66	11
DefaultDataSet	16	9
FoodList	147	78
ProductDetails	64	67
BattleStatistics	311	105
HandballModel	494	166

表 2 中第 1 列是测试程序的名称，第 2 列 Fit1 表示式(8)的适应度函数需要的进化代数，第 3 列 Fit2 表示式(6)即本文设计的适应度函数需要的进化代数。

从表 2 可以直观地看出，对于前 3 个面向过程的程序，Fit2 需要的进化代数少于 Fit1 需要的进化代数。对于 7 个面向对象的程序，ProductDetails 程序中两个适应度函数需要的进化代数相近(Fit2 比 Fit1 多了 3 代)。其他面向对象程序中 Fit2 需要的进化代数均少于 Fit1 需要的进化代数。式(8)中的适应度函数没有对每个 dup 进行评价，对测试用例的指导进化作用不明显。本文设计的适应度函数针对每个 dup 对测试用例进行评价，将评价得出的适应度值进行求和，减少了进化代数。

4.3.3 结果分析

通过前面两个实验，对提出的两个问题作出回答。

回答问题 1：相比于其他方法，本文方法是否更适合测试面向对象的程序？

针对简单的面向过程的程序，本文方法较其他方法的变异率没有太大差别；针对面向对象的程序来说，本文方法的变异率有明显提高，主要是由于数据流准则适合获取面向对象的程序中的状态依赖关系，基于控制流的准则易失去这些依赖关系。因此本文方法更适合测试面向对象的程序。

回答问题 2：本文设计的适应度函数是否优于其他文献的适应度函数？

本文设计的适应度函数分别对每个 dup 进行评价，并综合考虑所有的 dup，将评价得出的适应度值进行求和。在大部分测试程序上，本文方法需要的进化代数少于文献[10]中适应度函数所需的进化代数，对测试用例起到了很好的指导作用。

5 相关工作

Weyuker[16] 理论上研究了数据流准则的复杂性，利用测试用例的数目评估覆盖准则的复杂度。Denaro 等人[2] 量化了数据流测试的复杂度，并指出未覆盖的数据流元素不仅仅依赖于语句的可执行性。Santelices 等人[8] 提出了一种有效

监控 dup 覆盖的方法。静态阶段首先判断是否可以从分支中推断出 dup，如果可以则找出需要的分支并利用分支对程序插桩；否则对程序直接插桩。动态阶段监控插桩后的程序，并生成 dup 覆盖报告。

陈继峰等人[17]也提出了一种基于数据流覆盖准则的测试用例自动生成算法。该方法利用 Warshall 算法判断 dup 的可测性，并根据指定测试序列上的分支谓词构造约束求解系统生成测试用例。本文方法通过 CFG 和 ICFG 进行过程内和过程间的数据流分析，得到覆盖目标 dup 集合。利用遗传算法，根据适应度函数自动生成覆盖测试目标的测试用例。

Andrews 等人[15]利用变异分析对覆盖准则进行评估和比较，在一个工业程序上实证研究变异分析的可行性，结果表明变异分析在比较测试用例和覆盖准则的有效性上是有作用的，但考虑的是面向过程的程序中的过程内数据流关系。本文采用变异分析对不同覆盖方式生成的测试用例进行评估。

Nayak 等人[10]利用粒子群优化算法实现基于数据流的测试用例自动生成算法，但其适应度函数是测试用例已覆盖的定义使用路径和所有定义使用路径的比值，没有针对每个 dup 对测试用例进行评价。本文设计了一个适应度函数分别针对每个 dup 对测试用例进行评价，将得到的适应度值进行求和，针对所有覆盖目标进化测试用例。

结束语　本文通过对程序进行数据流分析得出待覆盖的测试目标 dup 集，利用遗传算法自动生成测试用例，根据适应度函数指导测试集的进化。通过 10 个测试程序对方法进行评估，实验证明本文方法取得了更高的变异率，更适合测试面向对象的程序。将本文设计的适应度函数和文献[10]中的适应度函数作比较，实验证明本文的适应度函数需要的进化代数更少，对测试用例起到更好的指导作用。

实验中观察到基于数据流的覆盖准则会生成更多的覆盖目标，其复杂度高于分支覆盖和语句覆盖。如何约减覆盖目标中 dup 的个数，降低数据流分析的复杂度是本文需要进一步研究的问题。

参 考 文 献

[1] YOUNG M. Software Testing and Analysis: Process, Principles, and Techniques[M]. John Wiley & Sons, 2008.

[2] DENARO G, PEZZE M, VIVANTI M. Quantifying the Complexity of Dataflow Testing[C]// 2013 8th International Workshop on Automation of Software Test (AST). IEEE, 2013: 132-138.

[3] HARROLD M J, ROTHERMEL G. Performing Tata Flow Testing on Classes [J]. ACM SIGSOFT Software Engineering Notes, ACM, 1994, 19(5): 154-163.

[4] DENARO G, GORLA A, PEZZÈ M. Contextual Integration Testing of Classes[M]// Fundamental Approaches to Software Engineering. Springer Berlin Heidelberg, 2008: 246-260.

[5] SOUTER A L, POLLOCK L L. The construction of contextual def-use associations for object-oriented systems [J]. IEEE Transactions on Software Engineering, 2003, 29 (11): 1005-1018.

[6] DENARO G, MARGARA A, PEZZE M, et al. Dynamic Data Flow Testing of Object Oriented Systems[C]// Proceedings of the 37th International Conference on Software Engineering-Volume 1. IEEE Press, 2015: 947-958.

[7] HERMAN P M. A data flow analysis approach to program testing[J]. Australian Computer Journal, 1976, 8(3): 92-96.

[8] SANTELICES R, HARROLD M J. Efficiently Monitoring Data-flow Test Coverage[C]// Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering. ACM, 2007: 343-352.

[9] GIRGIS M R. Automatic Test Data Generation for Data Flow Testing Using a Genetic Algorithm[J]. J. UCS, 2005, 11(6): 898-915.

[10] NAYAK N, MOHAPATRA D P. Automatic test data Generation for Data Flow Testing Using Particle Swarm Optimization [M]// *Contemporary Computing*. Springer Berlin Heidelberg, 2010: 1-12.

[11] PANDE H D, LANDI W A, RYDER B G. Interprocedural defuse associations for C systems with single level pointers[J]. IEEE Transactions on Software Engineering, 1994, 20(5): 385-403.

[12] HARROLD M J, SOFFA M L. Efficient computation of interprocedural definition-use chains[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1994, 16(2): 175-204.

[13] WEGENER J, BARESEL A, STHAMER H. Evolutionary test environment for automatic structural testing[J]. Information and Software Technology, 2001, 43(14): 841-854.

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

[15] ANDREWS J H, BRIAND L C, LABICHE Y, et al. Using mutation analysis for assessing and comparing testing coverage criteria[J]. IEEE Transactions on Software Engineering, 2006, 32 (8): 608-624.

[16] WEYUKER E J. The complexity of data flow criteria for test data selection[J]. Information Processing Letters, 1984, 19(2): 103-109.

[17] CHEN J F, SHEN J Y, WANG X J, et al. Automatic test data generation algorithm based on data flow rules[J]. Microelectrinics & Computer, 2007, 24(1): 5-8. (in Chinese)
陈继锋，沈钧毅，王欣峻，等. 一种基于数据流准则的测试数据自动生成算法[J]. 微电子学与计算机，2007，24(1)：5-8.

(上接第 87 页)

罗大晖，陈娟. 基于 HTML5 的 Web 离线应用研究与实现[J]. 计算机应用与软件，2012，29(12)：263-305.

[14] ZHENG Y. Research and Application of HTML5 Local Storage and Offline Caching Mechanisms[D]. Wuhan: Wuhan University of Technology, 2014. (in Chinese)
郑艳. HTML5 本地存储和离线缓存机制应用研究[D]. 武汉：武汉理工大学，2014.

[15] TIAN L, DU H C, TANG L. The discussion of cross-platform mobile application based on Phonegap[C]// Proceedings of 2013 IEEE 4th International Conference on Software Engineering and Service Science. 2013: 652-655.