(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 105975388 A
(43)申请公布日 2016.09.28

(21)申请号 201610184439.3

(22)申请日 2016.03.28

(71)申请人 南京邮电大学
地址 210003 江苏省南京市鼓楼区新模范马路66号

(72)发明人 王子元 张晓红 张卫丰 周国强 张迎周

(74)专利代理机构 南京知识律师事务所 32207
代理人 汪旭东

(51)Int.Cl.
G06F 11/36(2006.01)

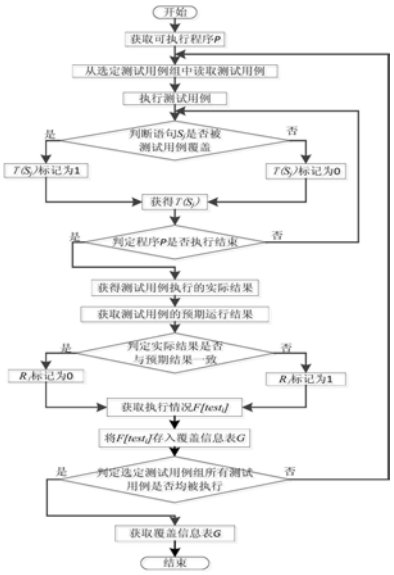
权利要求书4页 说明书9页 附图6页

(54)发明名称

基于频谱的增量式缺陷定位方法

(57)摘要

本发明公开了基于频谱的增量式缺陷定位方法，该方法采用增量式逐步迭代的方法，利用程序信息和测试信息找出程序缺陷语句或者预测缺陷语句可能存在的范围。该方法首先通过运行测试用例，收集测试用例在程序中执行的覆盖信息即频谱信息以及运行结果信息生成覆盖信息表；再对覆盖信息表进行统计分析，计算程序语句可疑度，根据可疑度对程序语句进行排序获得缺陷定位序列；根据定位序列中语句排列逐个进行排错，直到找到引发程序异常的语句。本发明从多角度采取优化策略提高软件缺陷定位的效率，提高了测试用例的覆盖率，很好地减少了收集频谱的开销，提高了可疑度算法的精确性。



CN 105975388 A

1.一种频谱信息去冗余优化的软件缺陷方法,其特征在于,所述方法包含如下步骤:

获取初始定位序列;包括:

获取程序语句编号序列S:S_1,S_2,S_3,…,S_j,…,S_n;

获取语句序列S:S_1,S_2,S_3,…,S_j,…,S_n相应语句的可疑度值序列Z_{rough}:Z_1,Z_2,Z_3,…,Z_j,…,Z_n,其中Z_j表示第j条语句的可疑度值;

根据Z_{rough}中可疑度值进行降序排列;

根据已排序的Z_{rough}序列,匹配对应的程序语句编号,对S:S_1,S_2,S_3,…,S_j,…,S_n进行排序,排序后的S即为缺陷定位序列,记为S_{rough};

测试用例增量执行;包括:

增加一组测试用例并执行;

选取下一组测试用例;

可疑度修正;

获取执行完上一组测试用例获得语句S_j的<A_{ef},A_{ep},A_{nf},A_{np}>;

将ef,ep,nf,np统称为ev,利用公式(2)分别计算ef,ep,nf,np再次发生的概率p(ef),p(ep),p(nf),p(np);

$$p(ev)=\frac{A_{ev}}{A_{ef}+A_{ep}+A_{nf}+A_{np}}$$ 公式(2)

根据公式3分别计算ef,ep,nf,np再次发生后所提供的信息量H(ef),H(ep),H(nf),H(np);

$$H(ev)=\log_2\frac{1}{p(ev)}$$ 公式3

根据公式4分别计算ef,ep,nf,np的增量权重α_{ef},α_{ep},α_{nf},α_{np};

$$\alpha_{ev}=\frac{\frac{1}{k}+H(ev)}{1+H(ev)}$$ 公式4

其中,k为测试用例组数,1/k即为选定执行的测试用例条数占测试用例总条数的百分比;

获取执行完本选定组测试用例获得的语句S_j的<a_{ef},a_{ep},a_{nf},a_{np}>,利用公式5,分别计算A_{ef},A_{ep},A_{nf},A_{np},

$A_{ev}=A_{ev}+\alpha_{ev}\times a_{ev}$ 公式5

根据公式1重新计算语句可疑度f_{S_j};

将f_{S_j}记入z_{slight}中,z_{slight}表示修正后的可疑度值序列;

判断所有语句是否均获得修正后的可疑度值;

获得语句可疑度值序列z_{slight};

定位序列修正;

获取语句编号序列S:S_1,S_2,S_3,…,S_j,…,S_n;

获取语句序列S:S_1,S_2,S_3,…,S_j,…,S_n相应语句的可疑度值序列z_{slight}:z_1,z_2,z_3,…,z_j,…,z_n,其中z_j表示修正后的第j条语句的可疑度值;

根据z_{slight}中可疑度值进行降序排列;

根据已排序的z_{slight}序列，匹配对应的程序语句编号，对S:S_1,S_2,S_3,…,S_j,…,S_n进行排序，将排序后的S序列即为修正后定位序列，记为S_{slight}；

定位序列判定；

根据定位序列进行缺陷语句定位。

2.根据权利要求1所述的一种基于频谱信息的增增量式软件缺陷方法，其特征在于，所述方法包括：

步骤1:获取可执行的待测程序以及测试用例集，测试用例分组；

步骤1-1:获取待测的可执行程序P；

步骤1-2:将程序P的每条语句进行编号，按序编为S_1,S_2,S_3,…,S_j,…,S_n,将编号集合记为S,其中S_j表示程序P的第j条语句,n为程序语句的总条数；

步骤1-3:获取测试用例集，记为Listsuites；

步骤1-4:将测试用例集Listsuites中的测试用例进行等量随机分组，共分成k组，并对组进行编号，依次编为$Group_1$,$Group_2$,$Group_3$,...,$Group_k$。

3.根据权利要求1所述的一种基于频谱信息的增增量式软件缺陷方法，其特征在于，所述方法包括：

步骤2:运行测试用例，获取覆盖信息表；

步骤2-1:选取一组测试用例；

步骤2-2:对选择的测试用例集中的测试用例进行编号，按序编为$test_1$,$test_2$,$test_3$,…,$test_i$,…,$test_m$,其中$test_i$表示选定测试用例组中的第i条测试用例,m为选定测试用例组中测试用例的总条数；

步骤2-3:从选定的测试用例集按序读取测试用例$test_i$；

步骤2-4:运行当前读取的测试用例$test_i$；

步骤2-5:标记语句覆盖情况；

步骤2-5-1:判定程序P的语句S_j是否被测试用例覆盖，如果“是”,则转步骤2-5-2:,否则转步骤2-5-3；

步骤2-5-2:将语句标记为1；

步骤2-5-3:将语句标记为0；

步骤2-5-4:判定程序P是否执行结束，如果“是”,则转步骤2.5.5,否则转步骤2-5-1:,直到程序P的n条语句均被判定覆盖情况；

步骤2-5-5:收集执行完当前测试用例获得的频谱信息T(S_1),T(S_2),T(S_3),…,T(S_j),…,T(S_n),其中T(S_j)表示程序P的第j条语句被当前运行测试用例覆盖的情况；

步骤2-6:判定测试用例运行结果；

步骤2-6-1:获取当前测试用例$test_i$在程序中的实际运行结果；

步骤2-6-2:获取当前测试用例$test_i$的预期运行结果；

步骤2-6-3:判断$test_i$的实际运行结果与预期结果是否相同，如果“是”,转步骤2.6.4,否则转步骤2.6.5；

步骤2-6-4:运行结果记为0,记入R_i中,R_i表示测试用例运行结果情况；

步骤2-6-5:运行结果记为1,记入R_i中,R_i表示测试用例运行结果情况；

步骤2-7:收集测试用例$test_i$的运行情况，记为F[$test_i$]{T(S_1),T(S_2),T(S_3),…,T

$(S_j),\cdots,T(S_n),R_i\}$,

其中:$T(S_j)$表示语句S_j被当前测试用例覆盖的情况,即频谱信息;R_i表示当前测试用例的运行通过情况,即结果信息;

步骤2.8:将$F[test_i]\{T(S_1),T(S_2),T(S_3),\cdots,T(S_j),\cdots,T(S_n),R_i\}$按行存入如图1所示的覆盖信息表G中,G由$F[test_i]\{T(S_1),T(S_2),T(S_3),\cdots,T(S_j),\cdots,T(S_n),R_i\}$累积生成;

步骤2.9:判断选定的测试用例集中的所有的测试用例是否均被执行,如果“是”,转步骤2-10:,否则转步骤2-3;

步骤2-10:获取包含选定测试用例集中所有测试用例的频谱信息以及结果信息的覆盖信息表G。

4.根据权利要求1所述的一种基于频谱信息的增增量式软件缺陷方法,其特征在于,所述方法包括:

步骤3:可疑度计算;

步骤3-1:针对S_j语句,收集执行完选定测试用例集所提供的$\langle a_{ef},a_{ep},a_{nf},a_{np}\rangle$,其中,ef表示测试用例i执行语句$S_j$,且执行实际结果与预期结果不一致;

ep表示某测试用i例执行语句S_j,且执行实际结果与预期结果一致;

nf表示某测试用例i没有执行语句S_j,且执行实际结果与预期结果不一致;

np表示某测试用例i没有执行语句S_j,且执行实际结果与预期结果一致;

$a_{ef}=\sum_{i=1}^{m}G_{i,j}*R_i$,表示语句$S_j$被测试用例覆盖且测试用例失败的次数,$G_{i,j}$表示覆盖信息表G的第i行第j列的元素值,$R_i$表示覆盖信息G中R列的第i行元素值,即:

$a_{ep}=\sum_{i=1}^{m}G_{i,j}*(1-R_i)$,表示语句$S_j$被测试用例覆盖且测试用例通过的次数;

$a_{nf}=\sum_{i=1}^{m}(1-G_{i,j})*R_i$,表示语句$S_j$没有被测试用例覆盖且测试用例失败的次数;

$a_{np}=\sum_{i=1}^{m}(1-G_{i,j})*(1-R_i)$,表示语句$S_j$没有被测试用例覆盖且测试用例通过的次数;

步骤3-2:令$\langle A_{ef},A_{ep},A_{nf},A_{np}\rangle=\langle a_{ef},a_{ep},a_{nf},a_{np}\rangle$,将$\langle A_{ef},A_{ep},A_{nf},A_{np}\rangle$初始化为$\langle a_{ef},a_{ep},a_{nf},a_{np}\rangle$的当前值;

步骤3-3:根据公式1计算语句S_j的可疑度f_{S_j};

$$f_{S_j}(A_{ef},A_{ep},A_{nf},A_{np})=\frac{\frac{A_{ef}}{A_{ef}+A_{nf}}}{\frac{A_{ef}}{A_{ef}+A_{nf}}+\frac{A_{ep}}{A_{ep}+A_{np}}}\qquad \text{公式 1}$$

步骤3-4:将$f_{S_j}(A_{ef},A_{ep},A_{nf},A_{np})$记入$Z_{rough}$中,$Z_{rough}$表示可疑度值序列;

步骤3-5:判断所有语句是否均获得可疑度值,如果“是”,转步骤3-6,否则转步骤3-1;

步骤3-6:获得语句可疑度值序列Z_{rough}。

5.根据权利要求1所述的一种基于频谱信息的增增量式软件缺陷方法,其特征在于,所述定位序列判定包括:

步骤6-1:检验所有测试用例是否均执行完毕,如果“是”,转步骤6-8;

步骤6-2:获取定位序列S_{rough};

步骤6-3:获取定位序列S_{slight};

步骤6-4:将S_{slight}与S_{rough}进行比较,检验S_{slight}相对于S_{rough}是否稳定,如果“是”,转步骤6-8:,否则转步骤6-5;

步骤6-5:令$S_{rough}=S_{slight}$;

步骤6-6:S_{slight}清空;

步骤6-7:转步骤5-1;

步骤6-8:S_{slight}判定为最终定位序列。

6.根据权利要求1所述的一种基于频谱信息的增增量式软件缺陷方法,其特征在于,根据定位序列进行缺陷语句定位包括:

步骤7-1:获取最终定位序列S_{slight};

步骤7-2:读取S_{slight}中的编号S_j;

步骤7-3:查看程序P的语句S_j是否含有缺陷,如果“是”,转步骤7-4:,否则转步骤7-2;

步骤7-4:确定程序语句S_j为缺陷语句;

步骤7-5:检验程序P所有缺陷是否均被定位,如果“是”,转步骤7-6:,否则转步骤7-2;

步骤7-6:缺陷定位结束。

7.根据权利要求1所述的一种基于频谱信息的增增量式软件缺陷方法,其特征在于,所述方法是对采集的所有测试用例集进行等量随机分组,选择其中一组测试用例执行,收集本组测试用例的执行信息以及结果信息,生成覆盖信息表,对覆盖信息表进行统计分析,进行语句可疑度计算,根据可疑度值的大小获得定位序列,重新选择一组新的测试用例执行,原组测试用例的频谱信息与结果信息与新组测试用例执行信息进行动态加权对原可疑度进行修正,获得新的定位序列,新的定位序列与原定位序列进行比较,若序列趋于稳定或测试用例均执行完毕,则终止测试用例的执行,根据获得的最新定位序列逐个进行排错,直到找到引发程序异常的语句。

基于频谱的增量式缺陷定位方法

技术领域

[0001] 本发明涉及一种基于频谱的增量式缺陷定位方法，属于软件测试技术领域。

背景技术

[0002] 多年来，人们在缺陷定位的研究中提出了许多方法，主要通过程序的静态信息和动态信息来定位程序错误。但获得静态信息的开销较大，对于大型软件，全面的静态分析甚至是不现实的，而动态信息的收集主要是运行测试用例，并不会给测试带来过多的开销。同时，由于动态信息包含了程序运行时的信息，与利用静态信息的方法相比，可以提供更准确的结果。

[0003] 利用程序频谱信息进行缺陷定位，是目前比较切实有效的软件缺陷定位方法，程序频谱是一种表示程序运行时的覆盖信息，反应程序运行某一特征的代码剖面信息。程序频谱与程序行为之间存在着一定的关系，通过研究运行失败测试用例得到的频谱信息与运行通过测试用例得到的频谱信息之间的差异性可为软件缺陷定位提供帮助。对于程序的单条语句，被失败测试用例执行的越多，通过的测试用例执行的越少，语句含有错误的可能性就越大，发生错误的概率就越大。利用这种特征对程序语句被通过测试用例以及失败测试用例的覆盖情况进行统计分析，找出含有缺陷的程序语句。对基于频谱的程序定位方法，这里

发明内容

[0004] 本发明目的在于解决了上述现有技术的不足，提出了一种基于频谱的增量式缺陷定位方法，该方法采取优化策略产生一种新的缺陷定位方法，使缺陷定位的效率更高，本发明采取一种增量式的方法执行测试用例，并考虑了大量测试用例情况下的边际效应，对后续增加的测试用例对可疑度的贡献进行动态加权，对语句可疑度进行修正，根据修正后的定位序列进行缺陷定位。具体概述如下：对采集的所有测试用例集进行等量随机分组，选择其中一组测试用例执行，收集本组测试用例的执行信息(即频谱)以及结果信息，生成覆盖信息表，对覆盖信息表进行统计分析，进行语句可疑度计算，根据可疑度值的大小获得定位序列。重新选择一组新的测试用例执行，原组测试用例的频谱信息与结果信息与新组测试用例执行信息进行动态加权对原可疑度进行修正，获得新的定位序列，新的定位序列与原定位序列进行比较，若序列趋于稳定或测试用例均执行完毕，则终止测试用例的执行，根据获得的最新定位序列逐个进行排错，直到找到引发程序异常的语句。

[0005] 本发明解决其技术问题所采取的技术方案是：一种基于频谱的增量式缺陷定位方法，该方法包括如下步骤：

[0006] 步骤1：获取可执行的待测程序以及测试用例集，测试用例分组；

[0007] 步骤1-1：获取待测的可执行程序P；

[0008] 步骤1-2：将程序P的每条语句进行编号，按序编为S_1,S_2,S_3,…,S_j,…,S_n，将编号集合记为S，其中S_j表示程序P的第j条语句，n为程序语句的总条数；

[0009] 步骤1-3:获取测试用例集,记为Listsuites;

[0010] 步骤1-4:将测试用例集Listsuites中的测试用例进行等量随机分组,共分成k组,并对组进行编号,依次编为$Group_1$,$Group_2$,$Group_3$,...,$Group_k$;

[0011] 步骤2:运行测试用例,获取覆盖信息表;

[0012] 步骤2-1:选取一组测试用例;

[0013] 步骤2-2:对选择的测试用例集中的测试用例进行编号,按序编为

[0014] $test_1$,$test_2$,$test_3$,…,$test_i$,…,$test_m$,其中$test_i$表示选定测试用例组中的第i条测试用例,m为选定测试用例组中测试用例的总条数;

[0015] 步骤2-3:从选定的测试用例集按序读取测试用例$test_i$;

[0016] 步骤2-4:运行当前读取的测试用例$test_i$;

[0017] 步骤2-5:标记语句覆盖情况;

[0018] 步骤2-5-1:判定程序P的语句S_j是否被测试用例覆盖,如果“是”,则转步骤2-5-2:,否则转步骤2-5-3;

[0019] 步骤2-5-2:将语句标记为1;

[0020] 步骤2-5-3:将语句标记为0;

[0021] 步骤2-5-4:判定程序P是否执行结束,如果“是”,则转步骤2.5.5),否则转步骤2-5-1:,直到程序P的n条语句均被判定覆盖情况;

[0022] 步骤2-5-5:收集执行完当前测试用例获得的频谱信息

[0023] $T(S_1)$,$T(S_2)$,$T(S_3)$,…,$T(S_j)$,…,$T(S_n)$,其中$T(S_j)$表示程序P的第j条语句被当前运行测试用例覆盖的情况;

[0024] 步骤2-6:判定测试用例运行结果;

[0025] 步骤2-6-1:获取当前测试用例$test_i$在程序中的实际运行结果;

[0026] 步骤2-6-2:获取当前测试用例$test_i$的预期运行结果;

[0027] 步骤2-6-3:判断$test_i$的实际运行结果与预期结果是否相同,如果“是”,转步骤2.6.4),否则转步骤2.6.5);

[0028] 步骤2-6-4:运行结果记为0,记入R_i中,R_i表示测试用例运行结果情况;

[0029] 步骤2-6-5:运行结果记为1,记入R_i中,R_i表示测试用例运行结果情况;

[0030] 步骤2-7:收集测试用例$test_i$的运行情况,记为:

[0031] $F[test_i]\{T(S_1),T(S_2),T(S_3),\cdots,T(S_j),\cdots,T(S_n),R_i\}$,

[0032] 其中:$T(S_j)$表示语句S_j被当前测试用例覆盖的情况,即频谱信息;R_i表示当前测试用例的运行通过情况,即结果信息;

[0033] 步骤2.8:将$F[test_i]\{T(S_1),T(S_2),T(S_3),\cdots,T(S_j),\cdots,T(S_n),R_i\}$按行存入如图1所示的覆盖信息表G中,G由$F[test_i]\{T(S_1),T(S_2),T(S_3),\cdots,T(S_j),\cdots,T(S_n),R_i\}$累积生成;

[0034] 步骤2.9:判断选定的测试用例集中的所有的测试用例是否均被执行,如果“是”,转步骤2-10:,否则转步骤2-3;

[0035] 步骤2-10:获取包含选定测试用例集中所有测试用例的频谱信息以及结果信息的覆盖信息表G;

[0036] 步骤3:可疑度计算;

[0037] 步骤3-1:针对S_j语句,收集执行完选定测试用例集所提供的<a_{ef},a_{ep},a_{nf},a_{np}>,其中,ef表示测试用例i执行语句S_j,且执行实际结果与预期结果不一致;

[0038] ep表示某测试用i例执行语句S_j,且执行实际结果与预期结果一致;

[0039] nf表示某测试用例i没有执行语句S_j,且执行实际结果与预期结果不一致;

[0040] np表示某测试用例i没有执行语句S_j,且执行实际结果与预期结果一致;

[0041] $a_{ef}=\sum_{i=1}^{m}G_{i,j}*R_i$,表示语句$S_j$被测试用例覆盖且测试用例失败的次数,$G_{i,j}$表示覆盖信息表G的第i行第j列的元素值,$R_i$表示覆盖信息G中R列的第i行元素值;

[0042] $a_{ep}=\sum_{i=1}^{m}G_{i,j}*(1-R_i)$,表示语句$S_j$被测试用例覆盖且测试用例通过的次数;

[0043] $a_{nf}=\sum_{i=1}^{m}(1-G_{i,j})*R_i$,表示语句$S_j$没有被测试用例覆盖且测试用例失败的次数;

[0044] $a_{np}=\sum_{i=1}^{m}(1-G_{i,j})*(1-R_i)$,表示语句$S_j$没有被测试用例覆盖且测试用例通过的次数;

[0045] 步骤3-2:令<A_{ef},A_{ep},A_{nf},A_{np}>=<a_{ef},a_{ep},a_{nf},a_{np}>,将<A_{ef},A_{ep},A_{nf},A_{np}>初始化为<a_{ef},a_{ep},a_{nf},a_{np}>的当前值;

[0046] 步骤3-3:根据公式1计算语句S_j的可疑度f_{Sj};

[0047] $$f_{s_j}(A_{ef},A_{ep},A_{nf},A_{np})=\frac{\frac{A_{ef}}{A_{ef}+A_{nf}}}{\frac{A_{ef}}{A_{ef}+A_{nf}}+\frac{A_{ep}}{A_{ep}+A_{np}}}$$ 公式 1

[0048] 步骤3-4:将$f_{S_j}(A_{ef},A_{ep},A_{nf},A_{np})$记入$Z_{rough}$中,$Z_{rough}$表示可疑度值序列;

[0049] 步骤3-5:判断所有语句是否均获得可疑度值,如果“是”,转步骤3-6,否则转步骤3-1;

[0050] 步骤3-6:获得语句可疑度值序列Z_{rough};

[0051] 步骤4:获取初始定位序列;

[0052] 步骤4-1:获取程序语句编号序列S:S_1,S_2,S_3,…,S_j,…,S_n;

[0053] 步骤4-2:获取语句序列S:S_1,S_2,S_3,…,S_j,…,S_n相应语句的可疑度值序列

[0054] Z_{rough}:Z_1,Z_2,Z_3,…,Z_j,…,Z_n,其中Z_j表示第j条语句的可疑度值;

[0055] 步骤4-3:根据Z_{rough}中可疑度值进行降序排列;

[0056] 步骤4-4:根据已排序的Z_{rough}序列,匹配对应的程序语句编号,对

[0057] S:S_1,S_2,S_3,…,S_j,…,S_n进行排序,排序后的S即为缺陷定位序列,记为S_{rough};

[0058] 步骤5:测试用例增量;

[0059] 步骤5-1:增加一组测试用例并执行;

[0060] 步骤5-1-1:选取下一组测试用例;

[0061] 步骤5-1-2:重复步骤2-2:至步骤3-1;

[0062] 步骤5-2:可疑度修正;

[0063] 步骤5-2-1:获取执行完上一组测试用例获得语句S_j的<A_{ef},A_{ep},A_{nf},A_{np}>;

[0064] 步骤5-2-2:将ef,ep,nf,np统称为ev,利用公式2分别计算ef,ep,nf,np再次发生

的概率p(ef),p(ep),p(nf),p(np);

[0065] $$p(ev)=\frac{A_{ev}}{A_{ef}+A_{ep}+A_{nf}+A_{np}}$$ 公式2

[0066] 步骤5-2-3:利用公式3分别计算ef,ep,nf,np再次发生后所提供的信息量H(ef),H(ep),H(nf),H(np);

[0067] $$H(ev)=\log_2\frac{1}{p(ev)}$$ 公式3

[0068] 步骤5-2-4:利用公式4分别计算ef,ep,nf,np的增量权重α_{ef},α_{ep},α_{nf},α_{np};

[0069] $$\alpha_{ev}=\frac{\frac{1}{k}+H(ev)}{1+H(ev)}$$ 公式4

[0070] 其中,k为测试用例组数,1/k即为选定执行的测试用例条数占测试用例总条数的百分比;

[0071] 步骤5-2-5:获取执行完本选定组测试用例获得的语句S_j的<a_{ef},a_{ep},a_{nf},a_{np}>,利用公式5,分别计算A_{ef},A_{ep},A_{nf},A_{np},

[0072] $A_{ev}=A_{ev}+\alpha_{ev}\times a_{ev}$ 公式5

[0073] 步骤5-2-6:利用公式1重新计算语句可疑度f_{S_j};

[0074] 步骤5-2-7:将f_{Sj}记入z_{slight}中,z_{slight}表示修正后的可疑度值序列;

[0075] 步骤5-2-8:判断所有语句是否均获得修正后的可疑度值,如果“是”,转步骤5.2.9),否则转步骤5-2-1;

[0076] 步骤5-2-9:获得语句可疑度值序列z_{slight};

[0077] 步骤5-3:定位序列修正;

[0078] 步骤5-3-1:获取语句编号序列S:S_1,S_2,S_3,…,S_j,…,S_n;

[0079] 步骤5-3-2:获取语句序列S:S_1,S_2,S_3,…,S_j,…,S_n相应语句的可疑度值序列

[0080] z_{slight}:z_1,z_2,z_3,…,z_j,…,z_n,其中z_j表示修正后的第j条语句的可疑度值;

[0081] 步骤5-3-3:根据z_{slight}中可疑度值进行降序排列;

[0082] 步骤5-3-4:根据已排序的z_{slight}序列,匹配对应的程序语句编号,对

[0083] S:S_1,S_2,S_3,…,S_j,…,S_n进行排序,将排序后的S序列即为修正后定位序列,记为S_{slight};

[0084] 步骤6:定位序列判定;

[0085] 步骤6-1:检验所有测试用例是否均执行完毕,如果“是”,转步骤6-8;

[0086] 步骤6-2:获取定位序列S_{rough};

[0087] 步骤6-3:获取定位序列S_{slight};

[0088] 步骤6-4:将S_{slight}与S_{rough}进行比较,检验S_{slight}相对于S_{rough}是否稳定,如果“是”,转步骤6-8:,否则转步骤6-5;

[0089] 步骤6-5:令$S_{rough}=S_{slight}$,

[0090] 步骤6-6:S_{slight}清空;

[0091] 步骤6-7:转步骤5-1;

[0092] 步骤6-8:S_{slight}判定为最终定位序列;

[0093] 步骤7:根据定位序列进行缺陷语句定位;

[0094] 步骤7-1:获取最终定位序列S_{slight};

[0095] 步骤7-2:读取S_{slight}中的编号S_j;

[0096] 步骤7-3:查看程序P的语句S_j是否含有缺陷,如果“是”,转步骤7-4:,否则转步骤7-2;

[0097] 步骤7-4:确定程序语句S_j为缺陷语句;

[0098] 步骤7-5:检验程序P所有缺陷是否均被定位,如果“是”,转步骤7-6:,否则转步骤7-2;

[0099] 步骤7-6:缺陷定位结束。

[0100] 有益效果:

[0101] 1、本发明利用增量的方式执行测试用例,在语句可疑度计算中考虑到了多测试用例的边际效应,对缺陷定位序列进行实时修正,提高缺陷定位的效率,降低软件开发过程中缺陷定位的成本。

[0102] 2、本发明采取优化策略产生一种新的缺陷定位方法,使缺陷定位的效率更高,该方法采取一种增量式的方法执行测试用例,并考虑了大量测试用例情况下的边际效应,对后续增加的测试用例对可疑度的贡献进行动态加权,对语句可疑度进行修正,根据修正后的定位序列进行缺陷定位。

[0103] 3、本发明增量式体现在测试用例的分批执行,实时对定位序列进行修正,旨在减少收集程序谱的开销,提高软件缺陷定位的效率。

[0104] 4、本发明从多角度采取优化策略提高软件缺陷定位的效率,提高了测试用例的覆盖率,很好地减少了收集频谱的开销,提高了可疑度算法的精确性。

附图说明

[0105] 图1为本发明测试用例执行过程示意图。

[0106] 图2为本发明程序语句可疑度计算示意图。

[0107] 图3为本发明初始定位序列生成示意图。

[0108] 图4为本发明修正定位序列生成示意图。

[0109] 图5为本发明最终定位序列判定示意图。

[0110] 图6为本发明缺陷定位过程示意图。

具体实施方式

[0111] 下面结合说明书附图对本发明创造作进一步的详细说明。

[0112] 本发明的基于频谱的增量式缺陷定位方法是通过执行测试用例获得动态信息进行缺陷定位的方法。首先选定部分测试用例执行,对程序运行结果进行收集产生覆盖信息表,再根据覆盖信息表进行统计分析,进行可疑度计算,进而根据程序可疑度获得初始定位序列,采取逐步添加测试用例的方式对定位序列进行实时修正,获得排序相对稳定的定位序列,从测试用例集执行到定位缺陷语句的实现步骤为:

[0113] 步骤1:获取可执行的待测程序以及测试用例集,测试用例分组;

[0114] 步骤1-1:获取待测的可执行程序P,即包含缺陷语句的可执行程序;

[0115] 步骤1-2:对程序P的每条语句进行编号,依次编为$S_1,S_2,S_3,\cdots,S_j,\cdots,S_n$,其中,$S_j$表示程序P的第j条语句。对语句进行编号为可疑度计算以及语句排序提供便利;

[0116] 步骤1-3:获取测试用例集,记为Listsuites,测试用例按照特定的规则利用测试工具自动生成。一条测试用例由测试输入、执行条件以及预期结果组成;

[0117] 步骤1-4:将测试用例集Listsuites中的测试用例进行等量随机分组,共分成k组,并对组进行编号,依次编为$Group_1,Group_2,Group_3,...,Group_k$;测试用例的分组为测试用例的增量执行提供便利。随机分组的目的在于使得每组测试用例中包含各种类型的测试用例,保证每组测试用例的多样性。

[0118] 步骤2:运行测试用例,获取程序执行的动态信息,如图1所示;

[0119] 步骤2-1:选取一组测试用例;

[0120] 步骤2-2:对选择的测试用例集中的测试用例进行编号,按序编为

[0121] $test_1,test_2,test_3,\cdots,test_i,\cdots,test_m$,其中$test_i$表示选定测试用例组中的第i条测试用例,m为选定测试用例组中测试用例的总条数;对测试用例进行编号便于对执行情况进行统计分析;

[0122] 步骤2-3:从选定的测试用例集中读取测试用例$test_i$;

[0123] 步骤2-4:根据测试用例的输入以及执行条件执行读取的测试用例$test_i$;

[0124] 步骤2-5:在测试用例执行过程中对未覆盖语句以及覆盖的语句用“0”和“1”进行区分标记,“0”表示语句没有被当前测试用例执行,“1”表示语句被当前测试用例执行;

[0125] 步骤2-6:测试用例执行结束,获取实际运行结果,将执行结果与测试用例的预期结果进行比较,用“0”和“1”进行区分标记执行结果。“0”表示实际执行结果与预期结果一致,测试用例为通过的测试用例;“1”表示实际执行结果与预期结果不一致,测试用例为失败的测试用例;

[0126] 步骤2-7:用gcoverage收集测试用例的执行情况,执行情况包括测试用例执行过程中的语句覆盖情况以及最终执行结果情况,由“0”和“1”表示为频谱

[0127] $F[test_i]\{T(S_1),T(S_2),T(S_3),\cdots,T(S_j),\cdots,T(S_n),R_i\}$,也即频谱信息,表示测试用例运行轨迹。其中,$T(S_j)$表示测试用例$test_i$对语句$S_j$的覆盖情况,$R_i$表示第i条测试用例的运行结果;步骤2.9将$F[test_i]\{T(S_1),T(S_2),T(S_3),\cdots,T(S_j),\cdots,T(S_n),R_i\}$按行存入覆盖信息表G中,G由$F[test_i]\{T(S_1),T(S_2),T(S_3),\cdots,T(S_j),\cdots,T(S_n),R_i\}$累积生成,形成覆盖信息表目的在于对覆盖信息进行统计分析。

[0128] 步骤2.8:转步骤2-3:,读取并执行下一条测试用例,收集测试用例的运行情况,持续读取测试用例并执行,直至选定的测试用例集中的测试用例均被执行并获得覆盖信息;

[0129] 步骤2.9:获得包含选定测试用例集中所有测试用例的覆盖信息表G;

[0130] 步骤3:用可疑度值来衡量语句含有缺陷的可能性,进行语句可疑度计算,如图2;

[0131] 步骤3-1:针对S_j语句,对覆盖信息表进行统计分析,收集执行完选定测试用例集所提供的$\langle a_{ef},a_{ep},a_{nf},a_{np}\rangle$,其中,ef表示测试用例i执行语句$S_j$,且执行实际结果与预期结果不一致;

[0132] ep表示某测试用i例执行语句S_j,且执行实际结果与预期结果一致;

[0133] nf表示某测试用例i没有执行语句S_j,且执行实际结果与预期结果不一致;

[0134] np表示某测试用例i没有执行语句S_j,且执行实际结果与预期结果一致;

[0135] $a_{ef}=\sum_{i=1}^{m}G_{i,j}*R_i$，表示语句$S_j$被测试用例覆盖且测试用例失败的次数，$G_{i,j}$表示覆盖信息表G的第i行第j列的元素值，$R_i$表示覆盖信息G中R列的第i行元素值；

[0136] $a_{ep}=\sum_{i=1}^{m}G_{i,j}*(1-R_i)$，表示语句$S_j$被测试用例覆盖且测试用例通过的次数；

[0137] $a_{nf}=\sum_{i=1}^{m}(1-G_{i,j})*R_i$，表示语句$S_j$没有被测试用例覆盖且测试用例失败的次数；

[0138] $a_{np}=\sum_{i=1}^{m}(1-G_{i,j})*(1-R_i)$，表示语句$S_j$没有被测试用例覆盖且测试用例通过的次数；

[0139] 步骤3-2：令$\langle A_{ef},A_{ep},A_{nf},A_{np}\rangle=\langle a_{ef},a_{ep},a_{nf},a_{np}\rangle$，即将$\langle A_{ef},A_{ep},A_{nf},A_{np}\rangle$初始化为$\langle a_{ef},a_{ep},a_{nf},a_{np}\rangle$的当前值；

[0140] 步骤3-3：根据公式1计算语句S_j的可疑度f_{Sj}，语句可疑度值越大，含有缺陷的可能性就越大，语句可疑度值越大，含有缺陷的可能性就越大；

[0141] $$f_{s_j}(A_{ef},A_{ep},A_{nf},A_{np})=\frac{\frac{A_{ef}}{A_{ef}+A_{nf}}}{\frac{A_{ef}}{A_{ef}+A_{nf}}+\frac{A_{ep}}{A_{ep}+A_{np}}}$$ 公式 1

[0142] 步骤3-4：对计算的语句可疑度值进行存储，将$f_{S_j}(A_{ef},A_{ep},A_{nf},A_{np})$记入$Z_{rough}$中，$Z_{rough}$表示可疑度值序列；

[0143] 步骤3-5：转步骤3-1：，计算下一条语句的可疑度值，持续进行，直到所有的语句均进行统计分析，获得可疑度值；

[0144] 步骤3-6：获得完整语句可疑度值序列Z_{rough}；

[0145] 步骤4：获取初始定位序列如图3；

[0146] 步骤4-1：获取程序语句编号序列S：$S_1,S_2,S_3,\cdots,S_j,\cdots,S_n$，S为程序语句编号集合；

[0147] 步骤4-2：获取语句序列S：$S_1,S_2,S_3,\cdots,S_j,\cdots,S_n$相应语句的可疑度值序列

[0148] Z_{rough}：$Z_1,Z_2,Z_3,\cdots,Z_j,\cdots,Z_n$，其中$Z_j$表示第j条语句的可疑度值，可疑度作为错误定位的依据，可疑度值越大的语句含有缺陷的可能性就越大；

[0149] 步骤4-3：根据Z_{rough}中可疑度值进行降序排列，排列越前的语句含有缺陷的可能性越大；

[0150] 步骤4-4：根据已排序的Z_{rough}序列，匹配对应的程序语句编号，对

[0151] S：$S_1,S_2,S_3,\cdots,S_j,\cdots,S_n$进行排序，可疑度值大的语句编号排列在前，若遇到可疑度值相同的语句则编号小的语句排列在前，排序后的S即为缺陷定位序列，记为S_{rough}，S_{rough}即为初始定位序列；

[0152] 步骤5：增量式的缺陷定位方法中，本发明采取逐步增加测试用例的方式，并实时对缺陷定位序列进行修正。测试用例增量执行，修正定位序列生成如图4；

[0153] 步骤5-1：选择下一组测试用例并执行，收集覆盖信息，生成覆盖信息表；

[0154] 步骤5.2)针对语句S_j，对覆盖信息表进行统计，获得执行完本组测试用例获得的$\langle a_{ef},a_{ep},a_{nf},a_{np}\rangle$；

[0155] 步骤5-3：利用$\langle A_{ef},A_{ep},A_{nf},A_{np}\rangle$，利用公式2分别计算ef，ep，nf，np再次发生的概率，并利用公式3计算各自发生所提供的信息量，利用公式4求得加权因子，如公式5中利用

加权因子对<A_{ef},A_{ep},A_{nf},A_{np}>进行修正,获得修正后的<A_{ef},A_{ep},A_{nf},A_{np}>;

[0156] $$p(ev)=\frac{A_{ev}}{A_{ef}+A_{ep}+A_{nf}+A_{np}}$$ 公式2

[0157] $$H(ev)=\log_2\frac{1}{p(ev)}$$ 公式3

[0158] $$\alpha_{ev}=\frac{\frac{1}{k}+H(ev)}{1+H(ev)}$$ 公式4

[0159] $A_{ev}=A_{ev}+\alpha_{ev}\times a_{ev}$ 公式5

[0160] 其中,ev为ef,ep,nf,np的统称,计算过程中分别用ef,ep,nf,np进行替换。

[0161] 步骤5-3:获取语句S_j的<A_{ef},A_{ep},A_{nf},A_{np}>,利用公式1重新计算语句的可疑度值,并存入可疑度值序列z_{slight}中;

[0162] 步骤5.4:转步骤5-2,计算下一条语句的可疑度,持续进行,直至所有语句均重新获得可疑度值,并计入z_{slight};

[0163] 步骤5.5:对可疑度值序列z_{slight}进行降序排列,根据已排序的z_{slight}同步对程序语句编号序列S进行排序,可疑度值大的语句编号在前,若遇到可疑度值相同的语句则语句编号小的语句在前,获得修正后的定位序列S_{slight};

[0164] 步骤6:定位序列作为缺陷定位的依据,需要判定修正后的定位序列是否为最终的定位序列,定位序列的判定如图5;

[0165] 步骤6-1:判定所有测试用例是否均被执行,如果均被执行则判定S_{slight}为最终的定位序列,否则转步骤6-2;

[0166] 步骤6-2:将定位序列S_{rough}与S_{slight}进行比较,若S_{slight}相对于S_{rough}稳定则判定S_{sligh}为最终的定位序列,否则转步骤6-3;

[0167] 步骤6-3:将S_{slight}序列赋给S_{rough},S_{slight}清空,转步骤5-1:,执行一组新的测试用例,获取新的S_{slight},进行定位序列判定;

[0168] 步骤7:根据定位序列定位缺陷语句,缺陷定位过程如图6;

[0169] 步骤7-1:定位序列作为缺陷定位的依据,获取最终定位序列S_{slight};

[0170] 步骤7-2:语句在S_{slight}中的顺序决定了缺陷查找的顺序,读取S_{slight}中的编号S_j;

[0171] 步骤7-3:查看程序P中语句S_j是否含有缺陷,如果含有缺陷,则断定语句S_j为缺陷语句,否则转步骤7-2;

[0172] 步骤7-5:检验程序P所有缺陷是否均被定位,如果程序P的缺陷均已定位,缺陷定位结束,否则转步骤7-2。

[0173] 表1:覆盖信息表

[0174]

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	R_i
$Test_1$	1	0	0	0	0	0	1	1
$Test_2$	1	1	1	1	1	0	0	0
$Test_3$	1	1	1	1	0	1	0	0
$Test_4$	0	1	1	1	1	1	1	1

$Test_5$	1	0	0	0	1	0	1	1
$Test_6$	0	1	0	0	0	1	1	0
$Test_7$	1	0	1	1	0	0	0	0
$Test_8$	0	1	0	0	1	1	0	1
$Test_9$	1	0	1	0	0	0	0	0
$Test_{10}$	1	1	0	1	0	1	1	1
$Test_{11}$	1	1	1	1	1	0	1	0
$Test_{12}$	1	1	1	0	0	1	0	1
$Test_{13}$	0	0	0	1	1	0	1	0
$Test_{14}$	1	1	1	0	0	0	0	1
$Test_{15}$	0	0	0	0	1	1	1	1
$Test_{16}$	0	1	1	1	1	0	1	0
$Test_{17}$	1	1	1	0	0	1	1	0
$Test_{18}$	1	0	1	1	0	0	0	1
$Test_{19}$	0	1	1	0	0	1	1	0
$Test_{20}$	1	0	0	1	1	1	0	1

[0175] 本发明是请根据上述表1的覆盖信息表所示,进行描述。

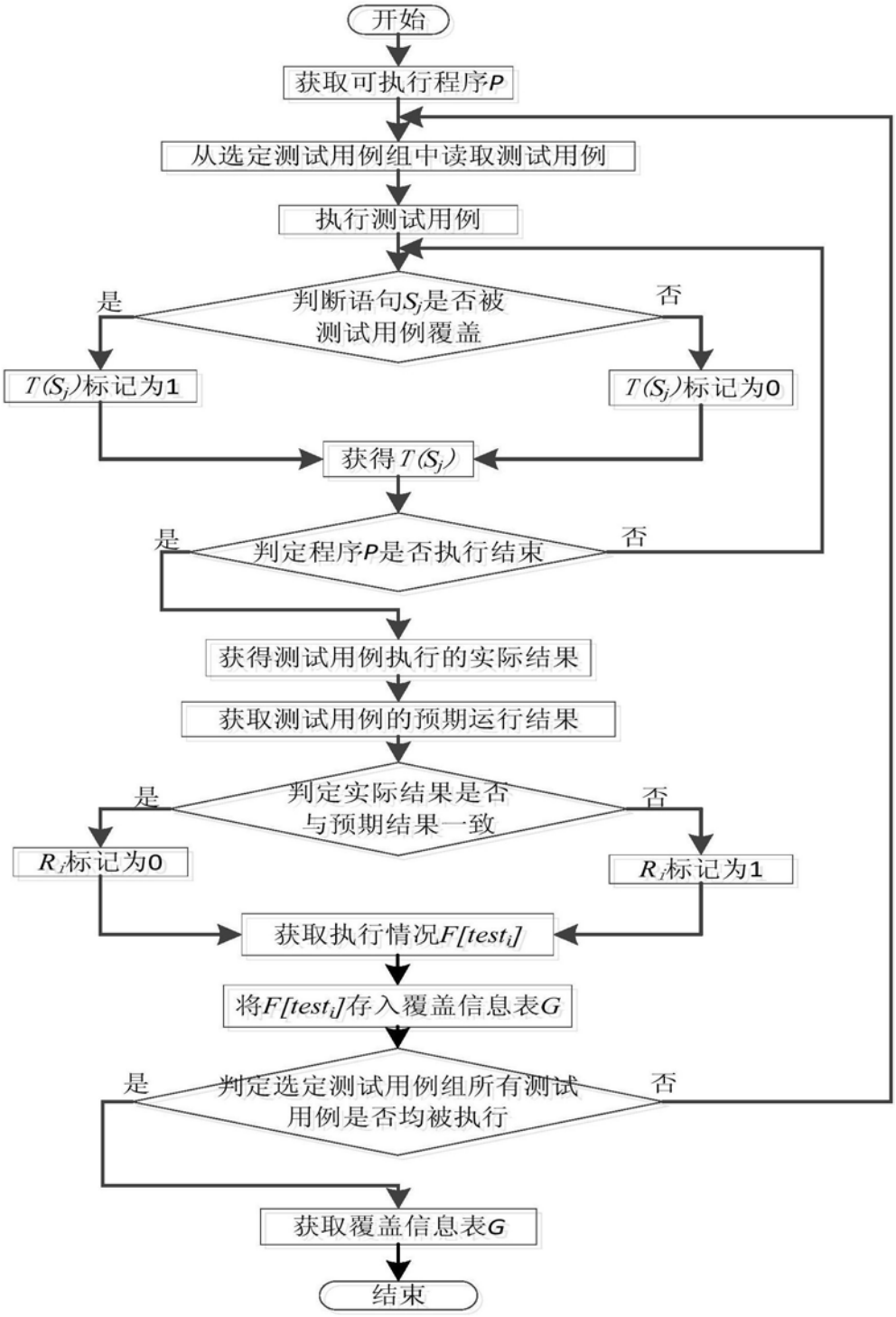

开始
获取可执行程序P
从选定测试用例组中读取测试用例
执行测试用例
判断语句S_j是否被测试用例覆盖
是
否
$T(S_j)$标记为1
$T(S_j)$标记为0
获得$T(S_j)$
判定程序P是否执行结束
是
否
获得测试用例执行的实际结果
获取测试用例的预期运行结果
判定实际结果是否与预期结果一致
是
否
R_i标记为0
R_i标记为1
获取执行情况$F[test_i]$
将$F[test_i]$存入覆盖信息表G
判定选定测试用例组所有测试用例是否均被执行
是
否
获取覆盖信息表G
结束

图1

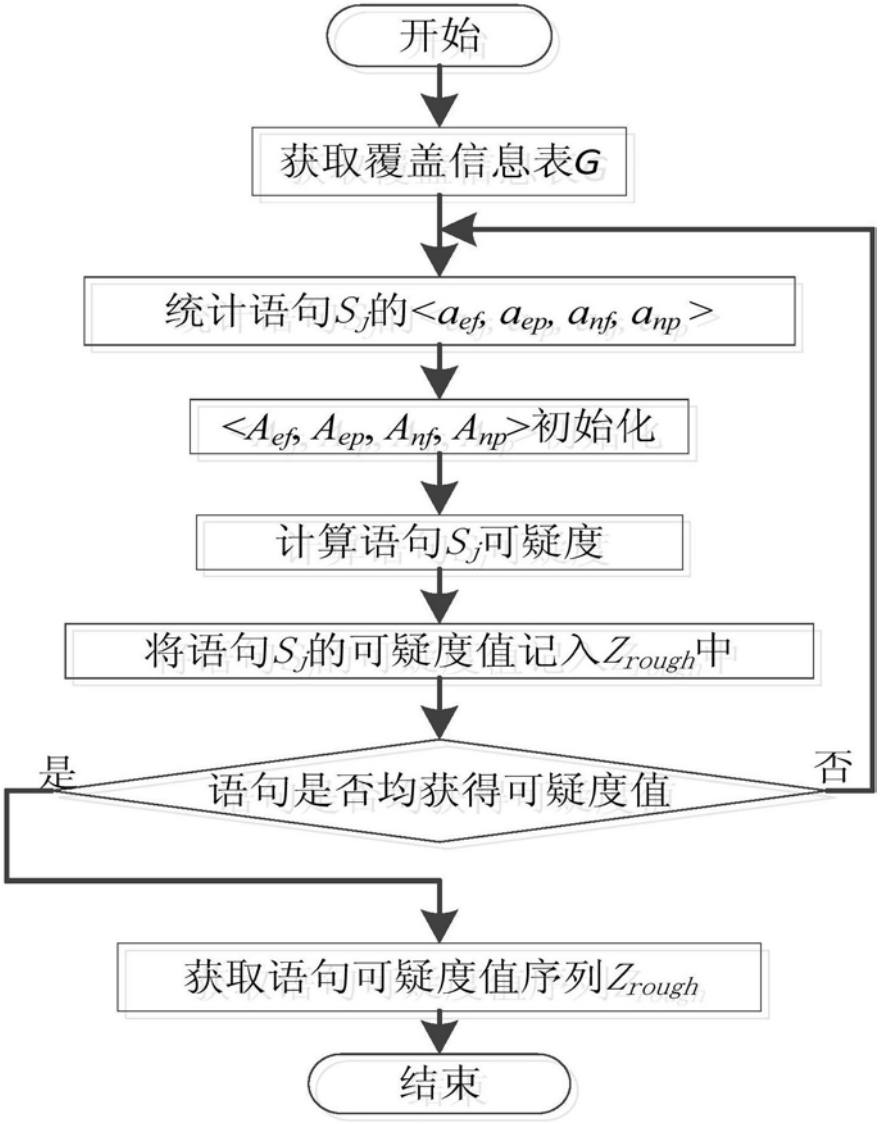

开始
获取覆盖信息表G
统计语句S_j的<a_{ef}, a_{ep}, a_{nf}, a_{np}>
<A_{ef}, A_{ep}, A_{nf}, A_{np}>初始化
计算语句S_j可疑度
将语句S_j的可疑度值记入Z_{rough}中
是
语句是否均获得可疑度值
否
获取语句可疑度值序列Z_{rough}
结束

图2

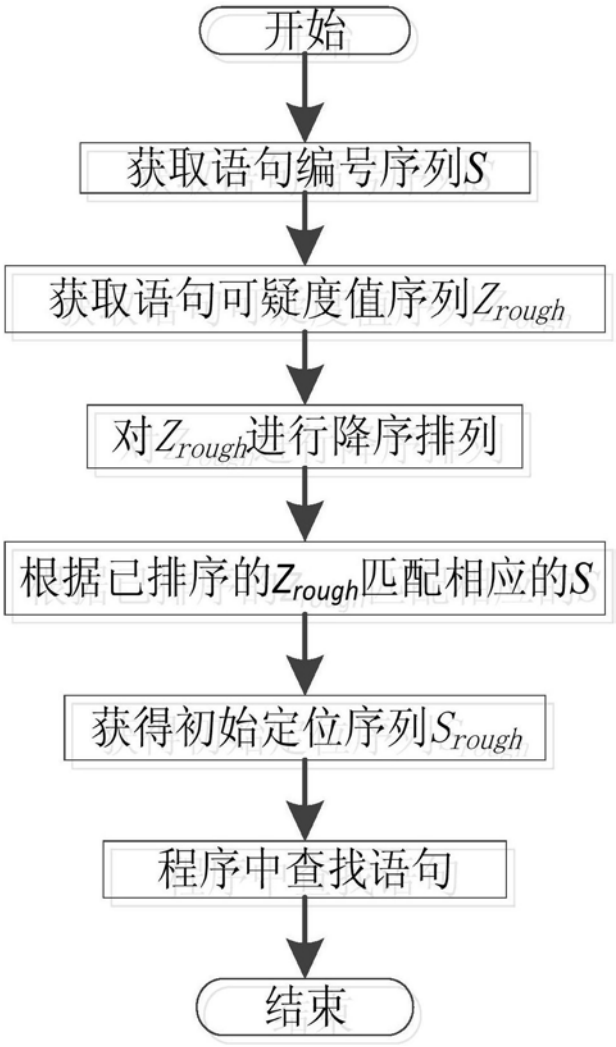

开始
获取语句编号序列S
获取语句可疑度值序列Z_{rough}
对Z_{rough}进行降序排列
根据已排序的Z_{rough}匹配相应的S
获得初始定位序列S_{rough}
程序中查找语句
结束

图3

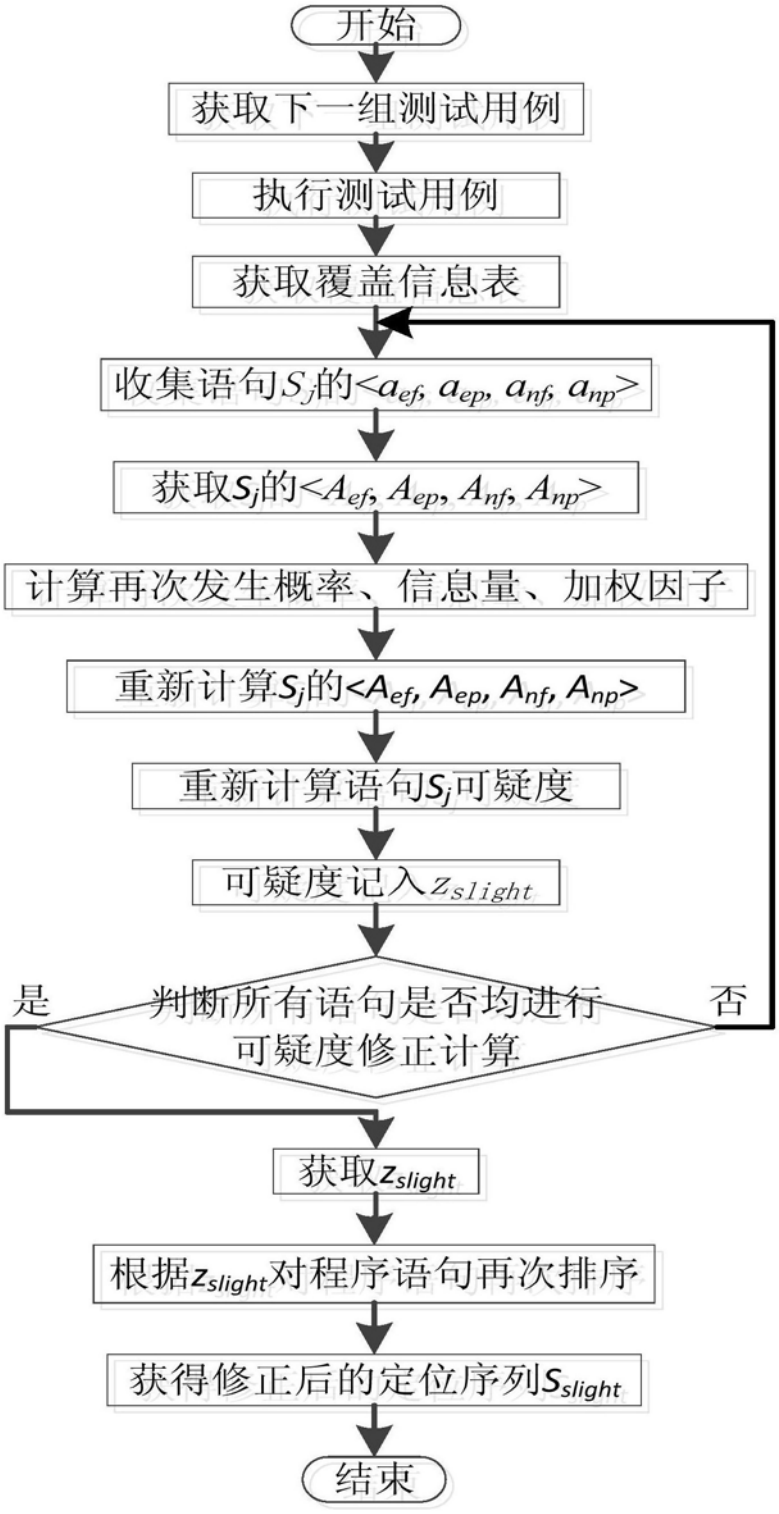

开始
获取下一组测试用例
执行测试用例
获取覆盖信息表
收集语句S_j的<a_{ef}, a_{ep}, a_{nf}, a_{np}>
获取S_j的<A_{ef}, A_{ep}, A_{nf}, A_{np}>
计算再次发生概率、信息量、加权因子
重新计算S_j的<A_{ef}, A_{ep}, A_{nf}, A_{np}>
重新计算语句S_j可疑度
可疑度记入z_{slight}
是
判断所有语句是否均进行可疑度修正计算
否
获取z_{slight}
根据z_{slight}对程序语句再次排序
获得修正后的定位序列S_{slight}
结束

图4

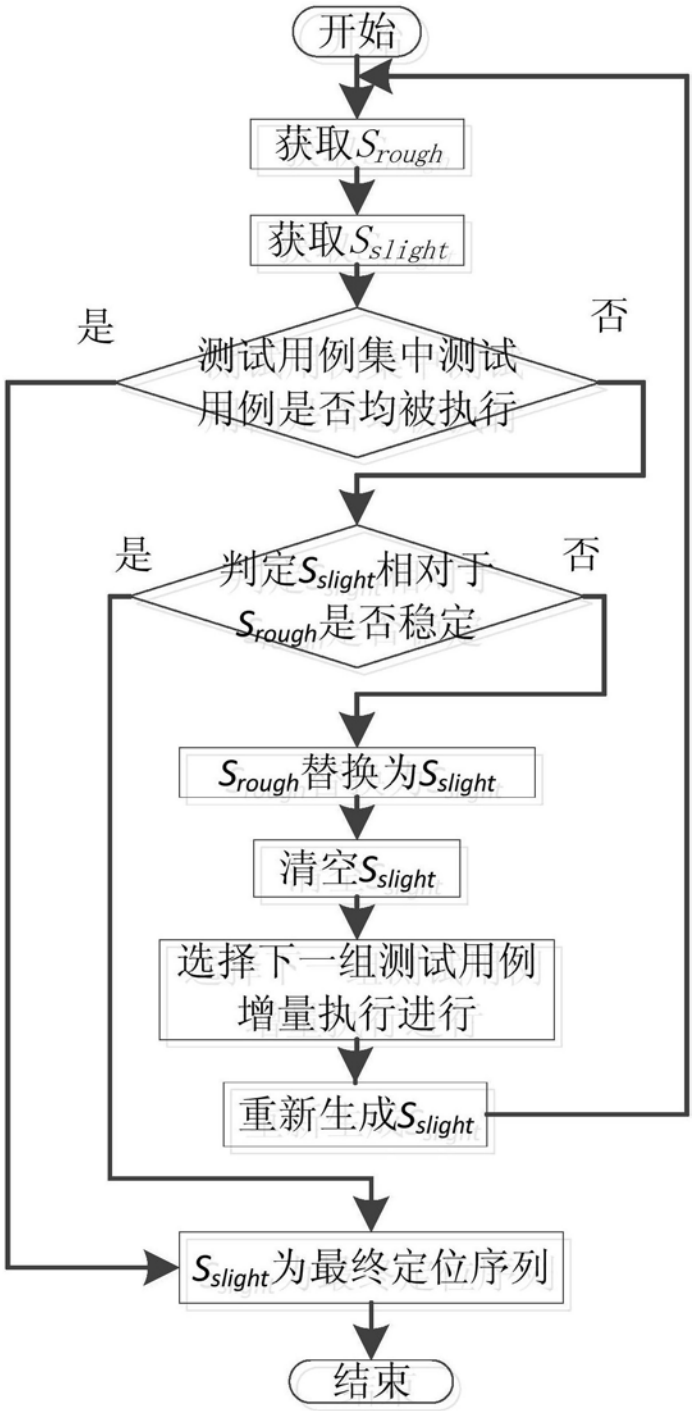

开始
获取S_{rough}
获取S_{slight}
是
否
测试用例集中测试用例是否均被执行
是
否
判定S_{slight}相对于S_{rough}是否稳定
S_{rough}替换为S_{slight}
清空S_{slight}
选择下一组测试用例增量执行进行
重新生成S_{slight}
S_{slight}为最终定位序列
结束

图5

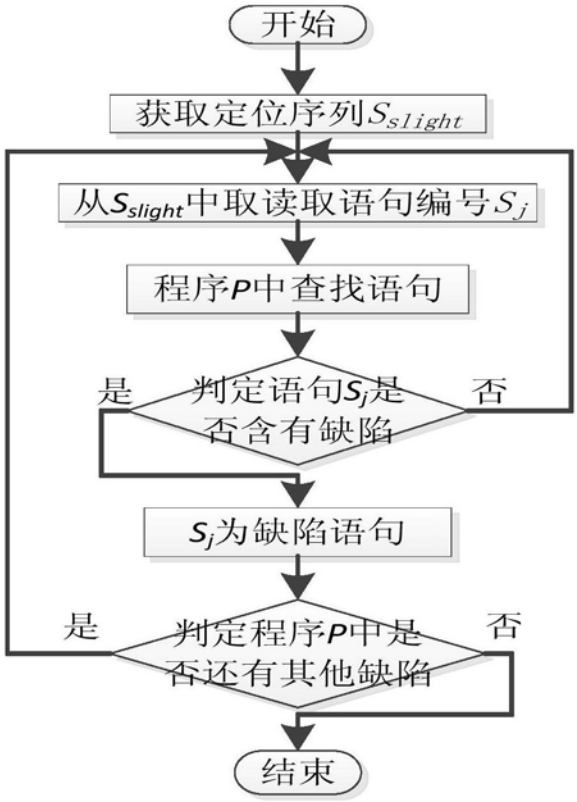

开始
获取定位序列S_{slight}
从S_{slight}中取读取语句编号S_j
程序P中查找语句
判定语句S_j是否含有缺陷
是
否
S_j为缺陷语句
判定程序P中是否还有其他缺陷
是
否
结束

图6