(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 104008051 B
(45)授权公告日 2017.04.26

(21)申请号 201410210134.6

(22)申请日 2014.05.16

(65)同一申请的已公布的文献号
申请公布号 CN 104008051 A

(43)申请公布日 2014.08.27

(73)专利权人 南京邮电大学
地址 210023 江苏省南京市栖霞区文苑路9号

(72)发明人 张卫丰 张晓红 王云 王子元 周国强 张迎周

(74)专利代理机构 南京正联知识产权代理有限公司 32243
代理人 王素琴

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

(56)对比文件
CN 103309811 A,2013.09.18,
KR 10-1134735 B1,2012.04.13,
CN 103136103 A,2013.06.05,
马倩等.“基于动态基本块的测试用例约简.《中国科技论文》.2012,第7卷(第1期),第33-41页.

审查员 李维

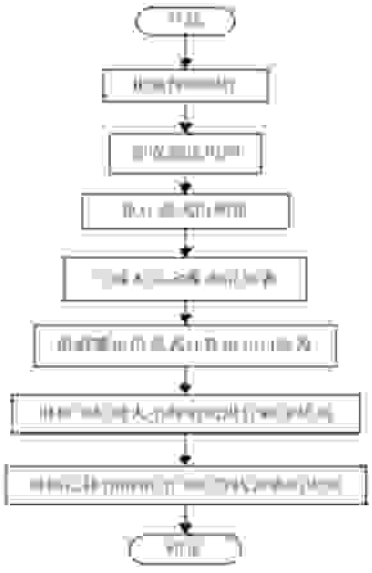
权利要求书3页 说明书7页 附图5页

(54)发明名称

频谱信息去冗优化的软件缺陷定位方法

(57)摘要

本发明提供一种频谱信息去冗优化的软件缺陷定位方法，通过运行测试用例，收集程序运行结果信息即频谱信息；对所得频谱信息进行去冗优化处理，利用频谱信息计算可疑度；根据可疑度值的大小对语句进行降序排列，根据已排序的语句序列逐个进行排错，直至找到引发程序异常的语句。本发明在基于频谱的错误定位方法中，在利用覆盖信息表进行可疑度计算之前，对频谱信息进行去冗余处理，利用有效的频谱信息进行可疑度计算，提高根据可疑度进行缺陷定位的可靠性，进而提高软件缺陷定位的效率。



CN 104008051 B

1.一种频谱信息去冗优化的软件缺陷定位方法，其特征在于：通过运行测试用例，收集程序运行结果信息即频谱信息；对所得频谱信息进行去冗优化处理，利用频谱信息计算可疑度；根据可疑度值的大小对语句进行降序排列，根据已排序的语句序列逐个进行排错，直至找到引发程序异常的语句；具体为：

S1、获取可执行的待测程序以及测试用例集；

S2、执行测试用例，获取程序执行的动态信息；

S3、收集所有测试用例的运行情况，并进行去冗余处理，生成无冗余覆盖信息表；具体为：

S31、首个测试用例默认处理；具体为：

默认第一个测试用例$test_1$为有效测试用例；对第一个有效测试用例$test_1$进行编号，记为$Test_1$，表示第一个有效测试用例；拓展有效测试用例的执行情况为$FF[test_1]\{Test_1,F[test_1]\{T(S_1),T(S_2),T(S_3)\ldots T(S_j)\ldots T(S_n),R\}\}$，即为$FF[test_1]\{Test_1,T(S_1),T(S_2),T(S_3)\ldots T(S_j)\ldots T(S_n),R\}$；收集测试用例$test_i$的运行情况，记为$F[test_i]\{T(S_1),T(S_2),T(S_3)\cdots T(S_j)\cdots T(S_n),R\}$，也即频谱信息，表示测试用例运行轨迹，其中，$T(S_1),T(S_2),T(S_3)\cdots T(S_j)\cdots T(S_n)$分别表示各语句$S_1,S_2,S_3,\ldots,S_j,\ldots,S_n$覆盖情况，$S_j$表示程序P的第j条语句，n为程序语句的总条数，R表示运行结果；

S32、将$FF[test_1]\{Test_1,T(S_1),T(S_2),T(S_3)\ldots T(S_j)\ldots T(S_n),R\}$记入覆盖信息表G，G由$FF[test_1]\{Test_1,T(S_1),T(S_2),T(S_3)\ldots T(S_j)\ldots T(S_n),R\}$累积生成；

S33、获取下一条测试用例的执行情况

$F[test_i]\{T(S_1),T(S_2),T(S_3)\ldots T(S_j)\ldots T(S_n),R\}$；

S34、判定$F[test_i]\{T(S_1),T(S_2),T(S_3)\ldots T(S_j)\ldots T(S_n),R\}$是否与覆盖信息表G已有行F[test]相同，如果“是”，转步骤S35，否则转步骤S37；

S35、判定测试$test_i$为非有效测试用例；

S36、删除$F[test_i]\{T(S_1),T(S_2),T(S_3)\ldots T(S_j)\ldots T(S_n),R\}$；

S37、判定测试用例$test_i$为有效测试用例；

S38、对有效测试用例$test_i$进行编号，编为$test_I$，表示第I个有效测试用例；

S39、拓展$test_i$运行结果为

$FF[test_i]\{Test_I,F[test_i]\{T(S_1),T(S_2),T(S_3)\ldots T(S_j)\ldots T(S_n),R\}\}$，

即$FF[test_i]\{Test_I,T(S_1),T(S_2),T(S_3)\ldots T(S_j)\ldots T(S_n),R\}$；

S310、将$FF[test_i]\{Test_I,T(S_1),T(S_2),T(S_3)\ldots T(S_j)\ldots T(S_n),R\}$记入覆盖信息表G中；

S311、判断Listsuites中所有测试用例是否都被执行，如果“是”，转步骤S312，否则转步骤S33；

S312、获得最终覆盖信息表G；

S4、根据覆盖信息表进行语句可疑度计算；

S5、根据可疑度值，对程序语句进行降序排列，定位缺陷语句。

2.如权利要求1所述的频谱信息去冗优化的软件缺陷定位方法，其特征在于，步骤S1具体为：

S11、获取待测的可执行程序P；

S12、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；

S13、获取测试用例集，记为Listsuites；

S14、对Listsuites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$test_i$表示测试用例集Listsuites中的第i条测试用例,m为测试用例的总条数。

3.如权利要求1所述的频谱信息去冗优化的软件缺陷定位方法，其特征在于，步骤S2具体为：

S21、根据测试用例的编号按序从Listsuites中读取测试用例$test_i$；

S22、根据测试用例的输入以及执行条件执行读取的测试用例$test_i$；

S23、在测试用例执行过程中对覆盖语句以及未覆盖的语句用“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；

S24、判定测试用例运行结果；

S25、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)...T(S_n),R}，即频谱信息，其中，T(S_j)表示语句S_j被当前测试用例覆盖的情况，R表示当前测试用例的运行通过情况；

S26、判断测试用例集中的所有的测试用例是否均被运行，如果“是”，转步骤S27，否则转步骤S21；

S27、收集所有测试用例的执行情况。

4.如权利要求3所述的频谱信息去冗优化的软件缺陷定位方法，其特征在于，步骤S24具体为：

S241、获取当前测试用例$test_i$在程序中的实际运行结果；

S242、获取当前测试用例$test_i$的预期运行结果；

S243、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S244，否则转步骤S245；

S244、运行结果记为0，记入R中，R表示测试用例运行结果情况；

S245、运行结果记为1，记入R中，R表示测试用例运行结果情况。

5.如权利要求1-4任一项所述的频谱信息去冗优化的软件缺陷定位方法，其特征在于，步骤S4具体为：

S41、针对S_j语句，收集执行完测试用例集Listsuites所提供的<a_{ef},a_{ep},a_{nx},a_{np}>，其中，

$a_{np}=\sum_{i=1}^{m}(1-G_{i,j})*(1-R_i)$ 表示语句S_j没有执行且测试结果正确的次数；

$a_{ef}=\sum_{i=1}^{m}(1-G_{i,j})*R_i$ 表示语句S_j没有执行且测试结果错误的次数；

$a_{ep}=\sum_{j=1}^{n}G_{i,j}*(1-R_i)$ 表示语句S_j被执行且测试结果正确的次数；$a_{ef}=\sum_{i=1}^{m}G_{i,j}*R_i$，

表示语句S_j被执行且测试结果错误的次数；$G_{i,j}$为覆盖信息表的第i行、第j列的元素值；

S42、根据公式(1)计算语句S_j的可疑度f_{S_j}；

$$f_{S_j}(a_{ij},a_{(i+1)j},a_{ik},a_{(i+1)k})=\frac{\frac{a_{ij}}{a_{ij}+a_{ik}}}{\frac{a_{ij}}{a_{ij}+a_{ik}}+\frac{a_{ik}}{a_{ik}+a_{(i+1)k}}}$$ 公式(1)

S43、将f_{S_j}记入Z中，Z表示可疑度值序列；

S44、判断所有语句是否均获得可疑度值，如果“是”，转步骤S45，否则转步骤S41；

S45、获得语句可疑度值序列Z。

6.如权利要求1-4任一项所述的频谱信息去冗优化的软件缺陷定位方法，其特征在于，步骤S5具体为：

S51、语句排序；

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中相应语句的可疑度值序列Z：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中可疑度值进行降序排列；

根据已排序的Z序列，对S进行排序；

S52、错误定位。

7.如权利要求6所述的频谱信息去冗优化的软件缺陷定位方法，其特征在于，步骤S52具体为：

S521、获取重新排序的S序列；

S522、按序获取S中的编号S_j；

S523、查看程序P的语句S_j是否含有缺陷，如果“是”，转步骤S524，否则转步骤S522；

S524、确定程序语句S_j为缺陷语句，缺陷定位成功。

频谱信息去冗优化的软件缺陷定位方法

技术领域

[0001] 本发明涉及一种频谱信息去冗优化的软件缺陷定位方法，属于软件测试领域。

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，而动态信息的收集只要是运行测试用例，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覆盖情况的信息，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成功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，被失效测试用例执行的越多，成功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成功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。

[0004] 对基于程序频谱的程序定位方法，可以从多角度采取优化策略提高软件缺陷定位的效率：第一，提高选取测试用例集的有效性；第二，尽量减少收集程序频谱的开销；第三，提高可以度算法的精确度性。在以往的优化策略的基础上，如何提出一种新的优化策略，对覆盖信息表进行去冗优化，提高依赖程序语句可疑度进行缺陷定位的可靠性，从而达到提高软件缺陷定位的效率是在基于程序频谱的程序定位方法的优化过程中应当予以考虑并解决的问题。

发明内容

[0005] 本发明对已有缺陷定位方法采取优化策略产生一种新的错误定位方法，使缺陷定位的效率更高。通过运行测试用例，收集程序运行结果信息即频谱信息，对频谱信息进行去冗优化处理，利用频谱信息进行可疑度计算，根据可疑度值的大小对语句进行降序排列，最后根据已排序的语句序列逐个进行排错，直到找到引发程序异常的语句。

[0006] 本发明的技术解决方案是：

[0007] 一种频谱信息去冗优化的软件缺陷定位方法，

[0008] 通过运行测试用例，收集程序运行结果信息即频谱信息；

[0009] 对所得频谱信息进行去冗优化处理，利用频谱信息计算可疑度；

[0010] 根据可疑度值的大小对语句进行降序排列，根据已排序的语句序列逐个进行排错，直至找到引发程序异常的语句。

[0011] 优选地，S1、获取可执行的待测程序以及测试用例集；

[0012] S2、执行测试用例，获取程序执行的动态信息；

[0013] S3、收集所有测试用例的运行情况，并进行去冗余处理，生成无冗余覆盖信息表；

[0014] S4、根据覆盖信息表进行语句可疑度计算；

[0015] S5、根据可疑度值，对程序语句进行降序排列，根据排序的语句序列进行缺陷定位。

[0016] 优选地，步骤S1具体为：

[0017] S11、获取待测的可执行程序P；

[0018] S12、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；

[0019] S13、获取测试用例集，记为Listsuites；

[0020] S14、对Listsuites中的测试用例进行编号，按序编为

[0021] $test_1$，$test_2$，$test_3$，...，$test_1$，...，$test_m$，其中$test_i$表示测试用例集Listsuites中的第i条测试用例，m为测试用例的总条数。

[0022] 优选地，步骤S2具体为：

[0023] S21、根据测试用例的编号按序从Listsuites中读取测试用例$test_i$；

[0024] S22、根据测试用例的输入以及执行条件执行读取的测试用例$test_i$；

[0025] S23、在测试用例执行过程中对覆盖语句以及未覆盖的语句用“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；

[0026] S24、判定测试用例运行结果；

[0027] S25、收集测试用例$test_i$的运行情况，记为

[0028] $F[test_i]\{T(S_1),T(S_2),T(S_3)...T(S_j)...T(S_n),R\}$，，即频谱信息，其中，$T(S_j)$表示语句$S_j$被当前测试用例覆盖的情况，R表示当前测试用例的运行通过情况；

[0029] S26、判断测试用例集中的所有的测试用例是否均被运行，如果“是”，转步骤S27，否则转步骤S21；

[0030] S27、收集所有测试用例的执行情况。

[0031] 优选地，步骤S24具体为：

[0032] S241、获取当前测试用例$test_i$在程序中的实际运行结果；

[0033] S242、获取当前测试用例$test_i$的预期运行结果；

[0034] S243、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S244，否则转步骤S245；

[0035] S244、运行结果记为0，记入R中，R表示测试用例运行结果情况；

[0036] S245、运行结果记为1，记入R中，R表示测试用例运行结果情况。

[0037] 优选地，步骤S3具体为：

[0038] S31、首个测试用例默认处理；

[0039] S32、将$FF[test_1]\{Test_1,T(S_1),T(S_2),T(S_3)...T(S_j)...T(S_n),R\}$记入覆盖信息表G，G由

[0040] $FF[test_1]\{Test_1,T(S_1),T(S_2),T(S_3)...T(S_j)...T(S_n),R\}$累积生成；

[0041] S33、获取下一条测试用例的执行情况

[0042] $F[test_i]\{T(S_1),T(S_2),T(S_3)...T(S_j)...T(S_n),R\}$；

[0043] S34、判定$F[test_i]\{T(S_1),T(S_2),T(S_3)...T(S_j)...T(S_n),R\}$是否与覆盖信息表G

已有行F[test]相同，如果“是”，转步骤S35，否则转步骤S37；

[0044] S35、判定测试$test_i$为非有效测试用例；

[0045] S36、删除F[$test_i$]{T(S_1)，T(S_2)，T(S_3)...T(S_j)...T(S_n)，R}；

[0046] S37、判定测试用例$test_i$为有效测试用例；

[0047] S38、对有效测试用例$test_i$进行编号，编为$Test_I$，表示第I个有效测试用例；

[0048] S39、拓展$test_i$运行结果为

[0049] FF[$test_i$]{$Test_I$，F[$test_i$]{T(S_1)，T(S_2)，T(S_3)...T(S_j)...T(S_n)，R}}，

[0050] 即FF[$test_i$]{$Test_I$，T(S_1)，T(S_2)，T(S_3)...T(S_j)...T(S_n)，R}；

[0051] S310、将FF[$test_i$]{$Test_I$，T(S_1)，T(S_2)，T(S_3)...T(S_j)...T(S_n)，R}记入覆盖信息表G中；

[0052] S311、判断Listsuites中所有测试用例是否都被执行，如果“是”，转步骤S312，否则转步骤S33；

[0053] S312、获得最终覆盖信息表G。

[0054] 优选地，步骤S31具体为：

[0055] 默认第一个测试用例$test_1$为有效测试用例；

[0056] 对第一个有效测试用例$test_1$进行编号，记为$Test_1$，表示第一个有效测试用例；

[0057] 拓展有效测试用例的执行情况为

[0058] FF[$test_1$]{$Test_1$，F[$test_1$]{T(S_1)，T(S_2)，T(S_3)...T(S_j)...T(S_n)，R}}，

[0059] 即为FF[$test_1$]{$Test_1$，T(S_1)，T(S_2)，T(S_3)...T(S_j)...T(S_n)，R}。

[0060] 优选地，步骤S4具体为：

[0061] S41、针对S_j语句，收集执行完测试用例集Listsuites所提供的$<a_{ef}, a_{ep}, a_{nf}, a_{np}>$，其中，$a_{nf}=\sum_{i=1}^{m}(1-G_{i,j})*(1-R_i)$表示语句$S_j$没有执行且测试结果正确的次数；$a_{nf}=\sum_{i=1}^{m}(1-G_{i,j})*R_i$表示语句$S_j$没有执行且测试结果错误的次数；$a_{ep}=\sum_{i=1}^{m}G_{i,j}*(1-R_i)$表示语句$S_j$被执行且测试结果正确的次数；$a_{ef}=\sum_{i=1}^{m}G_{i,j}*R_i$表示语句$S_j$被执行且测试结果错误的次数；$G_{i,j}$为覆盖信息表的第i行、第j列的元素值；

[0062] S42、根据公式(1)计算语句S_j的可疑度f_{S_j}；

[0063] $$f_{S_j}(a_{ef}, a_{nf}, a_{ep}, a_{np})=\frac{\frac{a_{ef}}{a_{ef}+a_{ep}}}{\frac{a_{ef}}{a_{ef}+a_{nf}}+\frac{a_{ep}}{a_{ep}+a_{np}}}$$ 公式(1)

[0064] S43、将f_{S_j}记入Z中，Z表示可疑度值序列；

[0065] S44、判断所有语句是否均获得可疑度值，如果“是”，转步骤S45，否则转步骤S41；

[0066] S45、获得语句可疑度值序列Z。

[0067] 优选地，步骤S5具体为：

[0068] S51、语句排序；

[0069] 获取语句编号序列S：S_1，S_2，S_3...S_j...S_n；

[0070] 获取语句序列S中相应语句的可疑度值序列Z：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；

[0071] 根据Z中可疑度值进行降序排列；

[0072] 根据已排序的Z序列，对S进行排序；

[0073] S52、错误定位。

[0074] 优选地，步骤S52具体为：

[0075] S521、获取重新排序的S序列；

[0076] S522、按序获取S中的编号S_j；

[0077] S523、查看程序P的语句S_j是否含有缺陷，如果“是”，转步骤S524，否则转步骤S522；

[0078] S524、确定程序语句S_j为缺陷语句，缺陷定位成功。

[0079] 本发明的有益效果是，本发明对比已有技术具有以下创新点：在基于频谱的错误定位方法中，在利用覆盖信息表进行可疑度计算之前，对频谱信息进行去冗余处理，利用有效的频谱信息进行可疑度计算，提高根据可疑度进行缺陷定位的可靠性，进而提高软件缺陷定位的效率。

附图说明

[0080] 图1是本发明实施例频谱信息去冗优化的软件缺陷定位方法的说明示意图；

[0081] 图2是本发明实施例中测试用例执行过程的示意图；

[0082] 图3是本发明实施例中无冗余覆盖信息表生成的示意图；

[0083] 图4是本发明实施例中可疑度计算的示意图；

[0084] 图5是本发明实施例中根据可疑度进行缺陷定位的说明示意图。

具体实施方式

[0085] 下面结合附图详细说明本发明的优选实施例。

[0086] 为了提高软件的质量，减小软件开发的成本，首先必须提高软件缺陷定位的效率。本实施例的目的在于在原有基于频谱的软件缺陷定位的方法的基础上采取优化策略，对收集的频谱信息进行去冗余处理，提高了根据可疑度进行缺陷的定位的可靠性，从而提高软件缺陷定位的效率，最终达到减少软件开发的成本的目的。

[0087] 本实施例的频谱信息去冗优化的软件缺陷定位方法是通过执行测试用例获得动态信息进行错误定位的方法。本实施例利用程序信息和测试信息找出错误语句或者预测错误语句可能存在的范围。首先对程序运行结果进行收集并进行去冗余处理产生覆盖信息表，再根据覆盖信息表进行可疑度计算，进而根据可疑度值大小对语句进行降序排列，最终根据已排序语句序列进行缺陷定位，从测试用例集执行到定位缺陷语句的实现步骤为：

[0088] 步骤1)获取可执行的待测程序以及测试用例集；

[0089] 步骤1.1)获取待测的可执行程序P,即包含缺陷的可执行程序;

[0090] 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,...,S_j,...,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;

[0091] 步骤1.3)将S_1,S_2,S_3,...,S_j,...,S_n语句编号序列记为S,S为程序P语句编号集合;

[0092] 步骤1.4)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;

[0093] 步骤1.5)对Listsuites中的测试用例进行编号,依次

[0094] $test_1$,$test_2$,$test_3$,...,$test_i$,...,$test_m$,其中,$test_i$为测试用例集中的第i条测试用例。对测试用例进行编号便于对执行情况进行统计分析;

[0095] 步骤2)执行测试用例,获取程序执行的动态信息,如图2;

[0096] 步骤2.1)根据测试用例的编号按序从Listsuites中读取测试用例$test_i$;

[0097] 步骤2.2)根据测试用例的输入以及执行条件执行读取的测试用例$test_i$;

[0098] 步骤2.3)在测试用例执行过程中对覆盖语句以及未覆盖的语句用“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;

[0099] 步骤2.4)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,测试用例执行结果正确;“1”表示实际执行结果与预期结果不一致,测试用例执行结果错误;

[0100] 步骤2.5)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

[0101] $F[test_i]\{T(S_1),T(S_2),T(S_3)...T(S_j)...T(S_n),R\}$,也即频谱信息,表示测试用例运行轨迹。其中,$T(S_j)$表示语句覆盖情况,R表示运行结果;

[0102] 步骤2.6)读取并执行下一条测试用例,收集测试用例的运行情况,重复执行,直至所有的测试用例都被执行结束;

[0103] 步骤3)收集所有测试用例的运行情况,并进行去冗余处理,生成无冗余覆盖信息表,如图3;

[0104] 步骤3.1)默认第一个测试用例为有效测试用例,并对第一个有效测试用例进行编号,并将运行情况进行拓展,加上有效测试用例编号,最终运行情况为

[0105] $FF[test_i]\{TestI,F[test_i]\{T(S_1),T(S_2),T(S_3)...T(S_j)...T(S_n),R\}\}$,

[0106] 即$FF[test_i]\{TestI,T(S_1),T(S_2),T(S_3)...T(S_j)...T(S_n),R\}$表示测试用例集中的第i个测试用例作为第I个有效测试用例的的运行情况。其中,$\{TestI,T(S_1),T(S_2),T(S_3)...T(S_j)...T(S_n),R\}$表示运行情况的元素组成;

[0107] 步骤3.2)将拓展后的测试用例执行情况$FF[test_i]$记入如表1所示覆盖信息表G,G由$FF[test_i]$累积生成,覆盖信息表的行表示测试用例的语句覆盖情况,以及本条测试用例最终执行通过情况;覆盖信息表的列(除最后一列)表示程序语句在各个测试用例中的覆盖情况,最后一列则表示所有测试用例的运行结果情况;

[0108] 表1覆盖信息表G

[0109]

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	R

$Test_1$	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

[0110]

$Test_{19}$	0	1	1	0	0	1	1	0
$Test_{20}$	1	0	0	1	1	1	0	1

[0111] 步骤3.3)获取下一条测试用例的的执行情况

[0112] $F[test_i]$ $\{T(S_1),T(S_2),T(S_3)\ldots T(S_j)\ldots T(S_n),R\}$；

[0113] 步骤3.4)对测试用例进行有效性判定，判定原则为判断

[0114] $F[test_i]$ $\{T(S_1),T(S_2),T(S_3)\ldots T(S_j)\ldots T(S_n),R\}$ 是否与覆盖信息表G已有行相同。若存在相同则判定当前测试用例为未有效测试用例，排除覆盖信息表的冗余性，直接删除当前测试用例的执行情况；若不存在相同行，则判定当前测试用例为有效测试用例，对其进行有效测试用例编号，拓展执行结果，将有效测试用例编号加入运行结果，并将运行结果记入覆盖信息表中；

[0115] 步骤3.5)重复步骤3.3)至步骤3.4)，直至所有的测试用例均被判别有效性，将最终拓展的运行情况记入覆盖信息表；

[0116] 步骤3.6)获得最终覆盖信息表；

[0117] 步骤4)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4；

[0118] 步骤4.1)针对单条语句S_j，对覆盖信息表进行统计分析，获得执行完整个测试用例集Listsuites所提供的$\langle a_{ef},a_{ep},a_{nf},a_{np}\rangle$；

[0119] 其中，

[0120] $a_{np}=\sum_{i=1}^{m}(1-G_{i,j})*(1-R_i)$ 表示语句S_j没有执行且测试结果正确的次数，

[0121] $a_{ij}=\sum_{i=1}^{m}(1-G_{i,j})*R_i$ 表示语句S_j没有执行且测试结果错误的次数，

[0122] $a_{ep}=\sum_{i=1}^{m}G_{i,j}*(1-R_i)$ 表示语句S_j被执行且测试结果正确的次数，

[0123] $a_{ef}=\sum_{i=1}^{m}G_{i,j}*R_i$ 表示语句S_j被执行且测试结果错误的次数；

[0124] $G_{i,j}$为覆盖信息表的第i行、第j列的元素值；

[0125] 步骤4.2)根据公式(1)计算语句S_j的可疑度f_{s_j}，语句可疑度值越大，含有缺陷的可能性就越大。可疑度越小，含有缺陷的可能性就越小。

[0126] $$f_{s_j}(a_{ef},a_{ep},a_{nf},a_{np})=\frac{\frac{a_{ef}}{a_{ef}+a_{nf}}}{\frac{a_{ef}}{a_{ef}+a_{nf}}+\frac{a_{ep}}{a_{ep}+a_{np}}}$$ 公式(1)

[0127] 步骤4.3)对计算的语句可疑度值进行存储，将可疑度f_{s_j}记入到可疑度值序列Z中；

[0128] 步骤4.4)重复步骤4.1)至步骤4.3)，直到所有的语句均进行统计分析，获得可疑度值。

[0129] 步骤5)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，越先进行排错，根据语句可疑度进行缺陷定位，如图5；

[0130] 步骤5.1)根据可疑度值对语句进行排序。对可疑度值序列Z进行降序排序，根据已排序的可疑度值Z序列对语句编号序列S进行重新排序，可疑度值大的语句编号在前，可疑度值小的语句编号在后，获得新的语句编号序列；

[0131] 步骤5.2)进行缺陷语句定位。根据已排序的语句编号序列，逐个排查程序P中的相应语句。可疑度值越大，越先进行排错，按序进行排错，直到找到含有缺陷的语句。

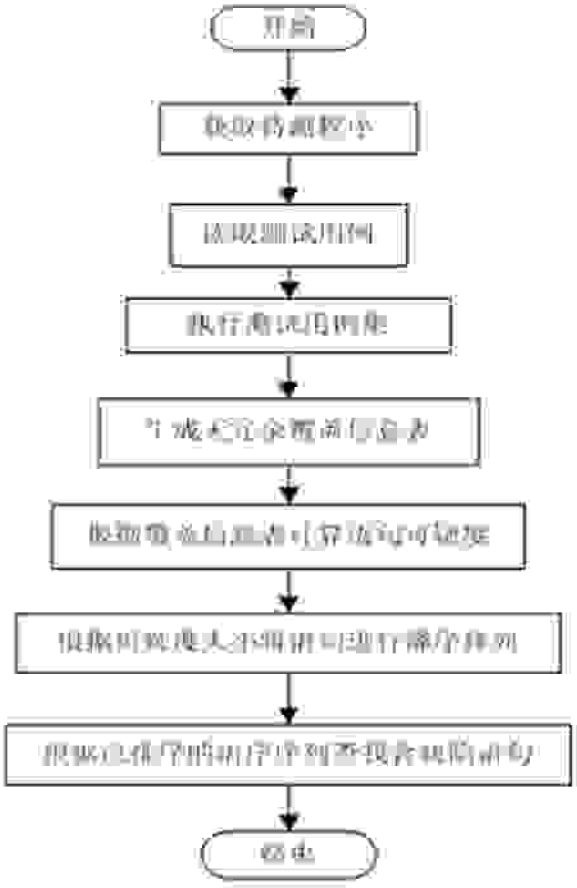


图1

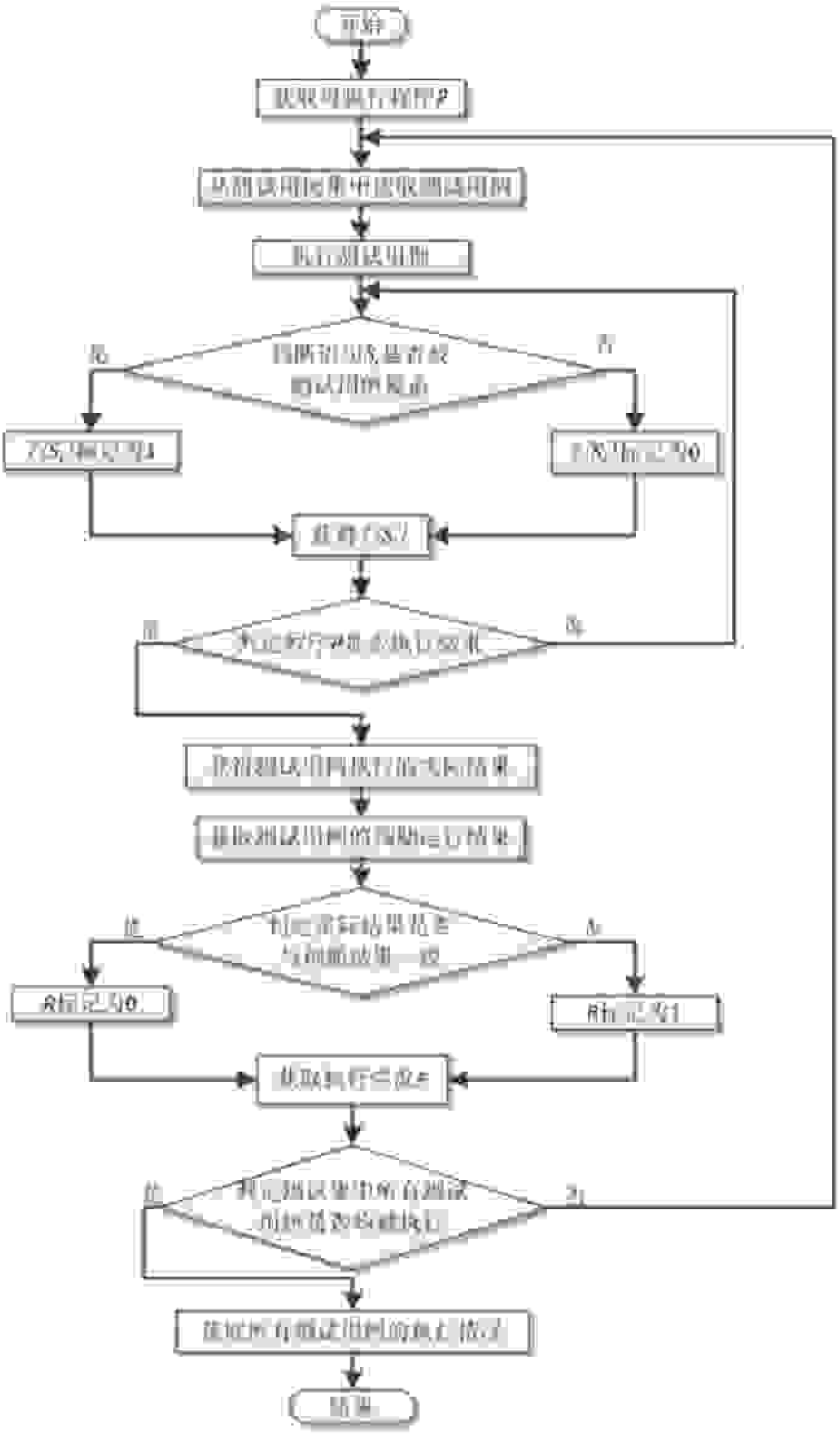


图2

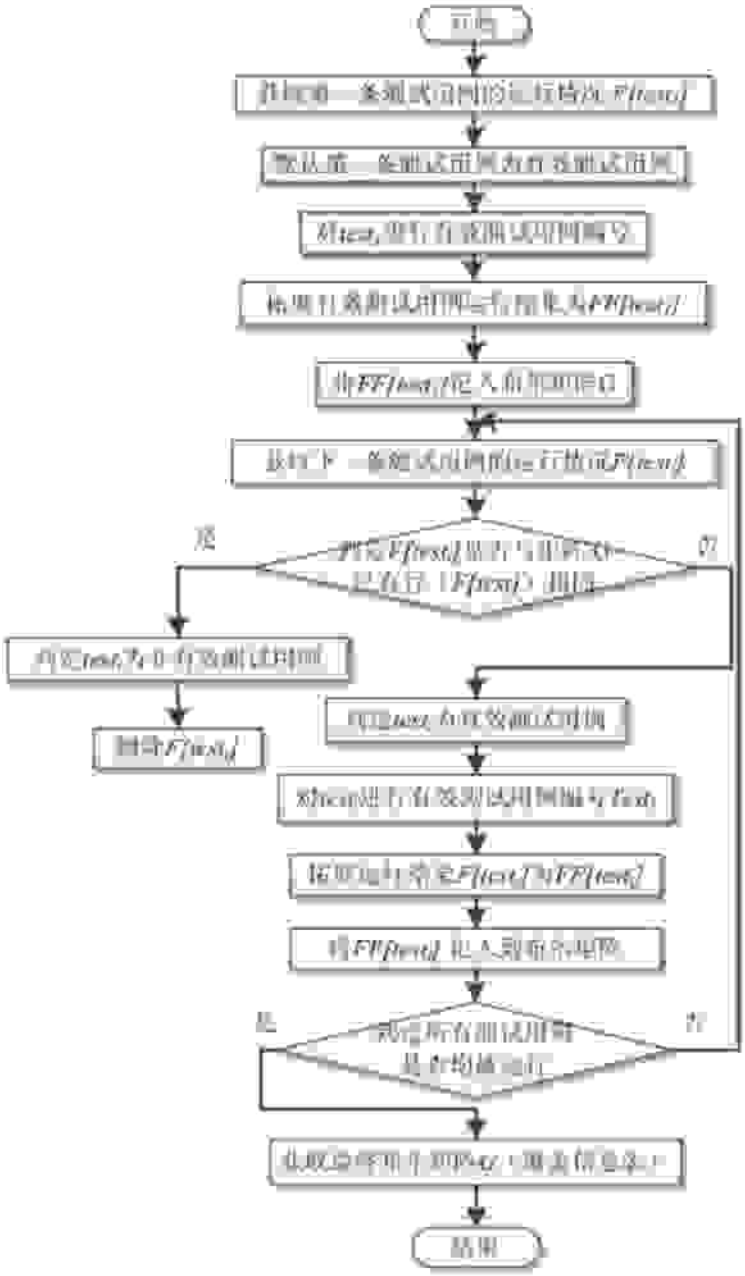


图3

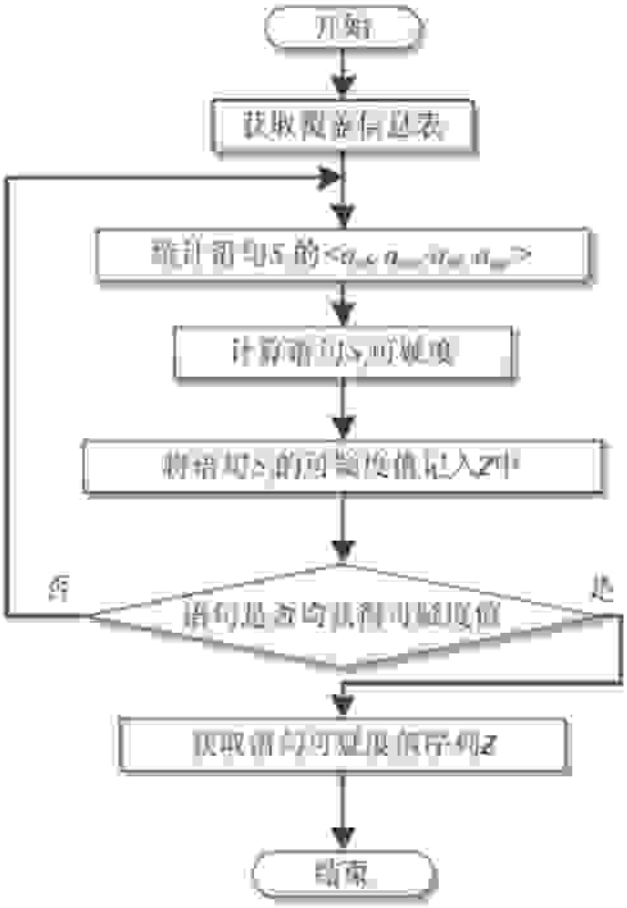


图4

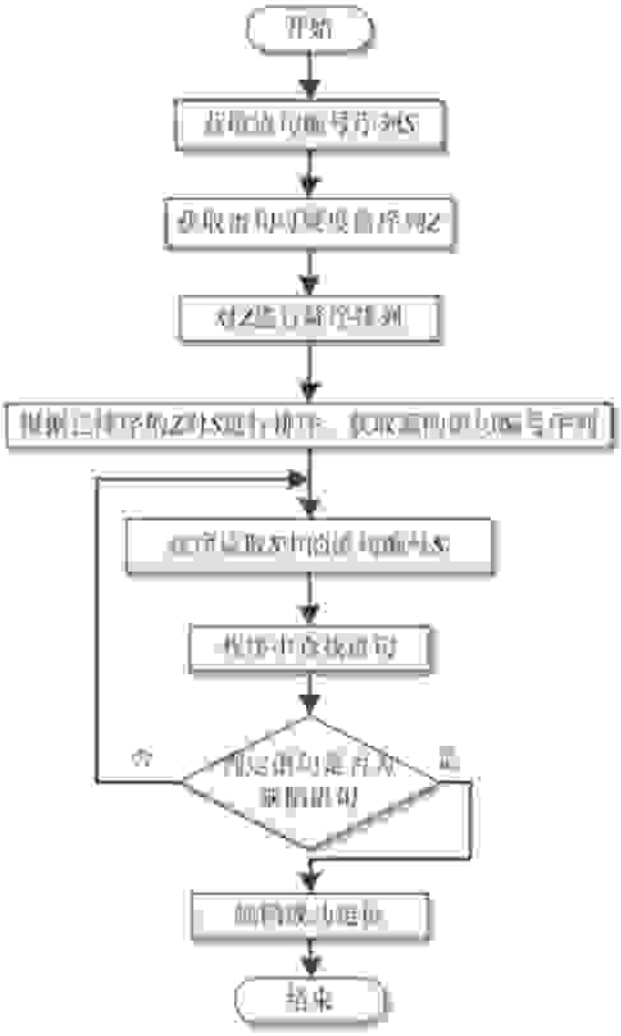


图5