基于输出代码行的回归测试集生成方法

丰 凯 高建华
(上海师范大学计算机科学与技术系 上海 200234)

摘 要 由于软件产品版本的不断更替、原代码的不断更改,使得测试人员需要具有针对性的回归测试,而目前的方法难以满足这样的需求。提出一种针对输出代码行进行生成回归测试集的方法,该方法通过每个测试用例的运行路径将回归测试集分类,测试用例若通过选定的输出代码行,则被选入测试子集中。此方法考虑代码行输出的功能特性,适用于在小段代码中获取具有针对性的测试集。此外,还将该方法应用于生成基于需求的回归测试集。实例表明,此方法可高效、全面地对已改代码选取回归测试集,能够满足要求严格的回归测试,并具有扩展性。

关键词 回归测试,代码行,选取测试子集,测试需求
中图法分类号 TP311 **文献标识码** A

Method of Regression Test Set Generation Based on Output Lines of Code

FENG Kai GAO Jian-hua
(Department of Computer Science and Technology, Shanghai Normal University, Shanghai 200234, China)

Abstract By reason of the constant turnover of the software version as well as the ceaseless change of the source code, tester needs the regression test with a specific targeted direction. However, the current technical methods are difficult in meeting such requirement. A method generating the test sets of regression test specifically based on the lines of the output code was proposed. The method classifies the test sets of regression test pertinently according to the execution path of each test case. The selection of the test case into the test subset will be made only if the case passes through the selected lines of the output code. Our method roundly takes the output functional properties of code lines into consideration which makes it applicable for obtaining the test sets whit certain pertinence in small block of code. Furthermore, this paper made the application of the method into the generation of the regression test sets based on the user requirements. Some examples show that the selection of regression test sets for modified code through this method can be both efficient and comprehensive, which enables it to suffice the regression test with stringent demand and provide test with expansibility.

Keywords Regression test, Lines of code, Select test subset, Test requirement

1 引言

在软件开发过程中,由于各种原因使得软件产品不断变更,造成程序模块的增加、修改和删除,为确保代码更改没有给软件产品引入缺陷,因此要对已改的软件产品进行回归测试。在软件产品的各个生命周期阶段,我们常用测试来检测软件产品的质量,以期在未造成软件的失效前将错误检测出来。

回归测试旨在检测程序中更改的代码有没有给软件带来错误。在回归测试中有两种测试策略:全部重测和选择性测试[1]。前者需要耗费太多时间和资源,而后者往往测试不够全面。尽管在回归测试中对于减少测试用例是一个NP完全问题,但我们一般可采用选择性测试来降低软件的测试成本。

在回归测试选择测试用例时,虽然有些回归测试选择技术是从程序规约中收集的信息来选择测试用例的,但大部分的回归测试选择技术都是根据程序代码的修改或代码的版本信息来选择测试用例的。这些基于代码的回归测试中有3种测试技术:覆盖技术、最小化技术和安全技术[2]。覆盖技术是从测试集 T 中选择测试用例 t 来覆盖程序中已修改的部分或被更改代码所影响的部分,它具有一定的不确定性,因为选择的那些达到覆盖要求的测试用例不一定是有效的,不能保证测试质量。最小化技术是从测试集 T 中选择一个最小测试集 T' 来覆盖已改的代码部分或是被更改代码影响的部分。而安全技术则是从测试集 T 中选择所有能暴露出已改程序

到稿日期:2012-03-12 返修日期:2012-06-01 本文受国家自然科学基金项目(61073163),上海市科委项目(09220503000),上海市引进技术的吸收与创新计划(2010CH-014)资助。

丰 凯(1986—),男,硕士生,主要研究方向为软件可靠性设计理论与方法、软件测试技术,E-mail:fjjk88@163.com;高建华(1963—),男,博士,教授,主要研究方向为软件可靠性理论与设计、软件开发环境与开发技术、数据安全与计算机安全、网络测试、LSI/VLSI测试等。

错误的测试用例，具有很强的精确性，但实现困难。当前用的最多的就是覆盖技术，如判定覆盖、条件覆盖、边覆盖、路径覆盖等。在传统的选择性测试用例的约简方法中，有启发式算法[3]、混合式算法[4]、贪婪选择法[5]等，它们都有各自的优、缺点和适应环境。

文献[6]提出一种增量测试软件产品，它的主要思想是：把软件产品分成若干个小的功能模块，即每个软件产品由多个功能模块组合而成。在对软件产品测试时，先对小模块进行测试，然后对下一个功能块进行测试，这样在进行模块测试时可以重用前面模块已有的测试用例，从而达到增量测试，减少测试用例的目的。在这里，测试用例都是通过测试生成工具自动生成的，第一个软件模块的测试用例是用工具直接生成的，后面模块的测试用例是根据上一个模块的测试用例和自身模块一起生成的。

不同于传统测试用例的选择技术，本文提出一种对于软件产品的功能输出来选择测试用例的方法。借鉴上述思想，在进行回归测试时，先把原软件分成不同的模块，每个模块有多个功能需求，在每个功能需求中又有多个小的功能代码段。如果检测各功能能够无误的输出，即可保证其上层的功能需求、模块及软件产品没有错误。本文是选定在程序段中的输出代码行，然后对涉及到此输出代码行的测试用例进行分类选择，以达到减少测试用例的目标。只要对各个已改程序段进行有效的测试，其组成的模块功能就会在预计范围内，从而保证软件产品的功能没有被更改代码所影响或功能影响在已设范围内。

基于输出代码行选取测试子集从本质上说也是基于覆盖的，不同的是它针对于覆盖某一输出行，找出此输出行涉及的测试用例，而不管这些测试用例其他的覆盖路径。由于输出行一般都是功能实现的出口，因此若是小功能的输出结果正确，则由多个小功能组成的软件产品的大功能也是正确的。因此，我们针对程序段的输出代码行对软件进行检测，即针对某一输出代码行来选取所有关于此行的测试用例，从而对这个小功能进行穷尽测试。基于输出代码行选取测试子集也可以看成是一个最小化技术，因为它是针对代码行来对测试集选取一个有关此行的测试子集。

本文第1节介绍在回归测试中现有的测试策略、测试方法和本文研究的主要问题；第2节通过具体实例提出了基于输出代码行的回归测试集生成方法；第3节通过两个实例说明了本文方法的有效性和扩展性；第4节阐述了该方法的特点；最后是全文总结。

2 测试用例选择

2.1 测试用例

本节以程序段 $read(x,y,z)$ 的修改为例来描述回归测试中测试用例的冗余问题。程序段 $read(x,y,z)$ 和修改后的程序段 $mread(x,y,z)$ 代码如下。

1.	read(x,y,z);	1.	mread(x,y,z);
2.	while(x>1&&y>1&&z>1)	2′.	while(x≥2||y≥1&&z≥1)
3.	if (x≥5)	3.	if (x≥5)
4.	z=1/(y−1);	4.	z=1/(y−1);
5.	else	5.	else
6.	y=y+x−z;	6.	y=y+x−z;
7.	endif	7.	endif
8.	x=x−1;	8.	x=x−1;
9.	end while	9.	end while

在函数 $read(x,y,z)$ 中，对第2行进行了修改，得到 $2'$ 行。这里用 $t_i(x_i,y_i,z_i)$ 表示测试用例。假设原函数的测试用例集为 $T=\{t_1(1,1,1),t_2(2,2,2),t_3(5,2,2)\}$，以满足条件覆盖。在第2行修改后，为达到同样的条件覆盖，则需添加测试用例 $t_4(5,1,0),t_5(5,0,1),t_6(1,0,0),t_7(2,1,0),t_8(2,0,1),t_9(5,0,0)$ 和 $t_{10}(2,0,0)$。若将补充的测试用例加入到测试用例集 T 中，则测试用例对程序段 $mread(x,y,z)$ 必然会有所冗余，很显然，$t_1,t_3,t_4,t_7,t_9,t_{10}$ 是冗余的。为减少测试用例、减小测试成本并且使测试更加具有针对性，本文设计一个减少测试用例的方法。

2.2 对于目标行的测试

与传统的减少测试用例的概念不同，这里的测试知识涉及3个概念：测试路径、目标行和目标测试集。下面对这3个概念进行定义。

定义1(测试路径)　把测试用例 t_i 在程序中沿 $m,n,\cdots$ (i,m,n 为正整数)代码行运行的路线称为测试用例 t_i 所运行的测试路径，记为 $p_i=(m,n,\cdots)$。

定义2(目标行)　把要测试的特定的代码行 l_i(i 为正整数，即是程序中的代码行数)称为目标行，记为 $g(i)$。

定义3(目标测试集)　把经过目标行 l_i 的所有测试用例的集合称为目标测试集，记为 $T(i)=\{t_k,t_u,\cdots\}$($i,k,u,\cdots$ 为正整数)。

表1给出了各个测试用例对函数 $mread(x,y,z)$ 的运行情况，“1”表示测试用例通过此行，“0”表示未通过此行。

表1　测试用例对 $mread(x,y,z)$ 的运行情况

T	1	2′	3	4	5	6	7	8	9
t_1	1	1	1	0	1	1	1	1	1
t_2	1	1	1	0	1	1	1	1	1
t_3	1	1	1	1	1	0	1	1	1
t_4	1	1	1	1	1	0	1	1	1
t_5	1	1	1	1	1	0	1	1	1
t_6	1	1	0	0	0	0	0	0	1
t_7	1	1	1	0	1	1	1	1	1
t_8	1	1	1	0	1	1	1	1	1
t_9	1	1	1	1	1	0	1	1	1
t_{10}	1	1	1	0	1	1	1	1	1

表1中有10个测试用例，它们的测试路径如下：

$p_1=(1,2',3,5,6,7,8,9)$

$p_2=(1,2',3,5,6,7,8,9)$

$p_3=(1,2',3,4,5,7,8,9)$

$p_4=(1,2',3,4,5,7,8,9)$

$p_5=(1,2',3,4,5,7,8,9)$

$p_6=(1,2',9)$

$p_7=(1,2',3,5,6,7,8,9)$

p_8=(1,2′,3,5,6,7,8,9)

p_9=(1,2′,3,4,5,7,8,9)

p_{10}=(1,2′,3,5,6,7,8,9)

由上可知,有些测试用例的路径相同,对路径覆盖而言,路径相同的测试用例是等价的,也是冗余的。于是,又可表示如下:

$\{p_1,p_2,p_7,p_8,p_{10}\}$= (1,2′,3,5,6,7,8,9)

$\{p_3,p_5,p_5,p_9\}$= (1,2′,3,4,5,7,8,9)

p_6= (1,2′,9)

2.3 减少测试用例

从测试某一代码行来找它们所对应的测试用例,进而达到减少测试用例的目的,而且这样的分类测试更具有目的性。设3行输出代码为目标行,即对 $g(4)$,$g(6)$,$g(8)$进行测试,于是得到其各自的目标测试集如下:

$T(4)=\{t_3,t_4,t_5,t_9\}$

$T(6)=\{t_1,t_2,t_4,t_7\}$

$T(8)=\{t_1,t_2,t_3,t_4,t_5,t_7,t_8,t_9,t_{10}\}$

在回归测试中针对某一目标代码行的测试选出其测试用例集,避免了用所有的测试用例进行测试。这样具有针对性地得到了目标行的测试用例集,以达到减少测试用例的目的,如以第4行代码为目标行,则目标测试集 $T(4)=\{t_3,t_4,t_5,t_9\}$。

2.4 算法

基于输出代码行的回归测试集生成方法的算法如下:

(1)标出回归测试集中每个测试用例的运行路径;

(2)按运行路径对测试用例进行分类,把路径相同的测试用例放到一起,并写出它们的运行路径;

(3)按所要测试的目标行从(2)中找出目标测试集。

本方法主要是在回归测试中对目标行选出所有的测试用例进行测试,而不考虑其它代码行。

3 方法的有效性和扩展性

3.1 方法的有效性

下面用本文的方法解决文献[4]中的一个实例来显示此方法的有效性和优越性,程序段 $read(a,b,c,d)$如下。

```
1.   read(a,b,c,d);                    满足分支覆盖的测试用
2.   if(a > 0)                          例集 T
3.     x=2;                           t1.(a=1,b=1,c=-1,d=0)
4.   else                             t2.(a=-1,b=-1,c=1,d=-1)
5.     x=5;                           t3.(a=-1,b=1,c=-1,d=0)
6.   endif                            t4.(a=-1,b=1,c=1,d=1)
7.   if(b > 0)                        t5.(a=-1,b=-1,c=1,d=1)
8.     y=1+x;
9.   endif
10.  if(c>0)
11.    if(d>0)
12.      output(x);
13.    else
14.      output(10);
15.    endif
16.  else
17.      output(1/(y-6));
18.  endif
```

由函数 $read(a,b,c,d)$的可知,它的出口在 l_{12},l_{14}和 l_{17} 3个输出行。表2中有5个测试用例,它们的测试路径如下:

p_1=(1,2,3,6,7,8,9,10,16,17,18,19)

p_2=(1,2,4,5,6,9,10,11,13,14,15)

p_3=(1,2,4,5,7,8,9,10,16,17,18,19)

p_4=(1,2,4,5,6,7,8,9,10,11,12,15)

p_5=(1,2,4,5,6,7,9,10,11,12,15)

表2 测试用例对 read(a,b,c,d)的运行情况

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
t_1	1	1	1	0	0	1	1	1	1	1	0	0	0	0	0	1	1	1	1
t_2	1	1	0	1	1	1	0	0	1	1	1	0	1	1	1	0	0	0	0
t_3	1	1	0	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1
t_4	1	1	0	1	1	1	1	1	1	1	1	1	0	0	1	0	0	0	0
t_5	1	1	0	1	1	1	1	0	1	1	1	1	0	0	1	0	0	0	0

然后,根据5个测试用例的测试路径找出3个输出行各自的目标测试集如下:

$T(12)=\{t_4,t_5\}$

$T(14)=\{t_2\}$

$T(17)=\{t_1,t_3\}$

所以可以根据目标行找到目标测试集,从而得到相应的测试用例子集。

文献[4]提出一种混合式算法(RSR算法):首先用 HGS 算法[3,4]和分支覆盖关系找到测试用例(t_1,t_2,t_4);然后根据 Definition-Use Pair 覆盖关系,从冗余的测试用例(t_3,t_5)中找出能检测出 l_{17}程序错误的测试用例 t_3;最后得到约简后的测试用例集(t_1,t_2,t_3,t_4)。而本文根据出口代码约简测试用例,可根据想要测试的目标行得到相应的测试用例集。很显然,不管输出结果是否正确,l_{12}、l_{14}都不会出错,在此以 l_{17}为目标行,从而得到目标测试用例集 $T(17)$。这样一次性地约简了测试用例,既得到了能检测出错误的测试用例 t_3,也得到了对目标行测试的另一测试用例。所以本文方法是简单高效的。在此也可以用 l_{12}、l_{14}为目标行进行测试。

RSR算法与通常的约简方法一样,是对所有的测试用例进行约简,约去冗余的测试用例,用最少的测试用例完成程序检测。不同于通常的约简方法,本文是针对目标行对测试用例集进行分类,得到目标行的所有的测试用例,对目标行进行穷尽测试,而不考虑除目标行之外的其它代码。

由上可知,RSR算法首先根据 HGS 算法和一种覆盖准则得到测试用例,再利用另一种覆盖准则在被判断为冗余的测试用例中继续寻找测试用例,若存在有效的则保留,否则就舍弃。而本文方法直接找到目标行就能找到目标测试集,从而一次性地找出有效的测试用例。所以本文方法更具有针对性。

3.2 方法的扩展性

在回归测试时,有时候软件产品的精确度要求不高,要降低测试代价时,并不想要基于代码的精确测试,而希望上层的需求、模块的测试。在此引用文献[3]中需求和测试用例的关系并对其稍作修改,假设这些需求是软件产品中的某一模块的所有需求,使用本文的思想来体现本文方法的扩展性。

文献[3]提出一个启发式局部优化算法。表3列出了各个需求对应的测试用例，与其它的基于覆盖技术进行约简测试用例的思想一样，它是从测试用例覆盖需求的角度进行约简测试用例，用最少的测试用例覆盖所有的需求，得到一个能覆盖所有需求的测试用例集{1,3,5}。所以，它具有覆盖技术约简测试用例的优缺点。

表3 测试信息

i	r_i	T_i
1	REQ1	[2,5]
1′	REQ1′	[1,2,5]//修后的需求
2	REQ2	[5]
2′	REQ2′	[3,5] //修后的需求
3	REQ3	[1,2,3]
4	REQ4	[3,6]
5	REQ5	[1,4]
6	REQ6	[1,6]
7	REQ7	[3,4,7]
8	REQ8	[2,3,4,7]

在此假设更改代码后 REQ1 和 REQ2 会改变，如表3所列，若用文献[3]的测试用例覆盖需求的方法，则测试用例集会变成{1,3}，连最初的测试用例5都省略了。一个需求包含多个代码段，所以具体需求的代码行数，要看具体的代码段，在此只考虑需求中包含的测试用例即可。这里针对需求进行测试，如已改后的需求 REQ1′，选取它包含的所有的测试用例[1,2,5]，不管其中的测试用例是否会暴露错误。所以，在扩展到需求时，只针对需求进行测试，对这些需求选取它所包含的所有的测试用例，若选择目标需求为 REQ1′，则目标测试集则为[1,2,5]。

由上可知，本文的思想不仅对于代码段有效，而且可以扩展到上一层基于需求的测试。

4 基于输出代码行的测试集生成方法的特点

根据输出代码行约简测试用例，得到测试集，能够根据所要检测的内容对其进行测试用例分类，以达到每个测试集对其中一个目标的所有情况进行测试。这样，在回归测试中我们只要根据此法对软件产品中的某一模块、需求、程序段进行检测即可。第2节中的例子说明了此法的有效性和扩展性。

由于越是基于底层的测试越精确，而回归测试归根到底是更改代码行引起的，因此基于代码行的测试才是最有效、最具有针对性的。只要根据代码行测试，需求、模块等就得到了精确的测试。本文基于代码行的测试可以用于输出代码行，也可以用于任一代码行，只是输出代码行一般是功能出口，更具有代表性。若有特定需求，也可以测试其它指定代码。

本文的方法与传统的测试子集选取的区别如下：

(1)传统测试子集的选取方法通常是针对程序代码分支、程序流程图的分支等。而本文的方法是针对代码行和需求的。

(2)传统测试集的约简一般对于程序代码分支、程序流图分支仅获得一个测试用例，以减少测试用例。而本文的方法是针对性地选取关于此种情况的所有测试用例，以达到约简测试用例的目的，而不管其它的代码。

(3)传统测试子集选取往往是根据测试用例所覆盖的需求、分支等约简测试用例，以达到用最少的测试用例来覆盖所有的需求、分支等目的。而本文的方法是根据某一功能输出，针对修改项来找出与其相关的测试用例，以找到所修改部分的测试用例集，然后对这一已修改的软件功能进行全面的回归测试。

本文方法也有一定的局限性，主要表现在：

(1)本方法只选择了对已改部分的测试用例，保证了它能达到预期的功能，而没有测试已改代码是否对其它未改程序段造成影响。

(2)本文研究的是软件产品的小的功能代码段，所以此代码段所属需求和模块最好也是根据功能代码划分的。要使本文方法更加有效，各功能段之间、功能需求之间及模块之间的耦合性要很低。

结束语 本文通过代码的功能输出行找出对应的测试用例集，以达到约简测试用例的目的。在文中先通过一个实例阐明思想，然后通过两个实例说明了基于输出代码行的测试集生成方法在回归测试中的有效性、优越性和扩展性。此方法还需要进一步地研究：(1)对于在软件产品中如何有效地划分模块以适应本文方法，在不同的软件产品中还需要再做研究；(2)当代码段中有诸多代码输出行，找出有效的目标行及要在具体的软件产品中运行的具体算法都是研究的难点。

参 考 文 献

[1] Kim J-M, Adam P, Gregg R. An Empirical Study of Regression Test Application Frequency[J]. Software Testing, Verification & Reliability, 2005, 15(4): 257-279

[2] Rothermel G, Harrold M J. Analyzing Regression Test Selection Techniques[J]. IEEE Transactions on Software Engineering, 1996, 22(8): 529-551

[3] Harrold M J, Gupta R, Soffa M L. A Methodology for Controlling the Size of a Test Suite[J]. ACM Transactions on Software Engineering and Methodology, 1993, 2(3): 270-285

[4] Jeffrey D, Gupta N. Improving Fault Detection Capability by Selectively Retaining Test Cases During Test Suite Reduction[J]. IEEE Transactions on Software Engineering, 2007, 33(2): 108-123

[5] Black J, Melachrinoudis E, Kaeli D. Bi-criteria Models for All-uses Test Suite Reduction [C]// Proceedings of the 26th International Conference on Software Engineering. 2004, 5: 106-115

[6] Uzuncaova E, Khurshid S, Batory D. Incremental Test Generation for Software Product Lines[J]. IEEE Transactions on Software Engineering, 2010, 36(3): 309-322