A Method of Fast Module Location Test Based on Requirements Changes *

Guo Xuepin\textsuperscript{a}, Chen Huandong\textsuperscript{a},\textsuperscript{*}, Wu Shulei\textsuperscript{a}, Zhan Jinmei\textsuperscript{b}, Zhong Sheng\textsuperscript{a}

\textsuperscript{a}College of Information Science and Technology  
Hainan Normal University  
haikou, China  
chd@hainnu.edu.cn  
\textsuperscript{b} Department of Information Technology  
Qiongtai Teachers College  
haikou, China

Abstract

As the existence of objective and subjective factors in developing software, changes in requirements is inevitable, often lead to changes in program modules also. Because of correlation among program modules, to modify a module may lead errors to related modules. Put forward to establish association diagrams of modules, and use methods of traversal of graphs, to fast track, positioning error or affected modules, and narrow test range, to some extent which improve the efficiency of software testing.

© 2011 Published by Elsevier Ltd. Open access under CC BY-NC-ND license.
Selection and/or peer-review under responsibility of the Intelligent Information Technology Application Research Association.
Keywords: requirements changes; software testing; association; association diagram; efficiency of testing;

1. Introduction

The software's power and higher reliability are being demanded as the increased requirements by information society, software system structure complicated, modifiability, testability, and maintainability of software degraded. It’s difficult to ensure software quality, so improving efficiency of testing plays an important role to guarantee the software quantity\cite{1}.

As the existence of objective factors and subjective factors in developing software, changes in requirements is inevitable, often lead to changes in program modules also\cite{2}. Because of correlation among program modules, to modify wrong modules, add a new one or delete unnecessary all may lead errors to
related modules. If the affected area of module changes cannot be determined accurately, need to do regression testing to all of program modules which is difficult and unnecessary[3]. To this point, determine accurately, fast track and positioning error or affected modules, and narrow regression testing range, which are questions of the last importance, to some extent which improve the efficiency of software testing.

2. Fast location testing ideas based requirements changes

Before testing, need to analysis structural relationship, particularly calling relationships, namely analysis association between program modules. Association refers to the mutual linkages and constraints relationships between the software modules and other parts. In software testing, there are the relationship between modules, and modules associated with test cases [4]. Association graph can be used to represent relationships between modules, by which to understand the structural relationship of modules much easier.

Analysis association of modules refers to analysis and research relationships of them. These relationships could be abstracted, and represented in the form of graphics, that help testers determine the effects of the error range quickly, and improve the efficiency of software testing and maintenance greatly.

When requirements change, before modify codes of a module, need to according criterion of positive association to assess its effect. Find out associated modules by association graphs, and do regression testing to them. Other non-related modules are without regard. More over, if somewhere have errors in testing, need to analysis the roots of the errors by method of reversal association, which caused by the original module or passed from the uppers.

3. Fast location testing implementation based requirements changes

3.1. Establish association relationship between modules

Establish association relationship between modules, to divide system procedures into basic modules, and establish association diagrams of modules to describe the modules controlling and calling relationships.

Modify procedure modules as requirements changes, find out associated modules through traversal association graph, and judge whether affected by modified module. If affected, must do necessary change to avoid the inconsistency. The modified modules can be the start of analysis and testing procedure in process of software modification.

3.2. Establish association relationship between modules and test cases

In software testing, test cases design for the modules to be tested and specialized service in one or some of the program modules. The test case will have real meanings only those modules for testing [5]. That is to say, must establish and maintenance association relationships between test cases and program modules, unless when do regression testing to the modules to be tested can not sure which test case or test cases to be used. It’s unnecessary to choose all of them.

As requirements often change, it is impossible to choose all of the test cases in each change. Besides, modify a module may affect some other modules in program. So must to do regression testing for its associated modules too. Without affecting the quality of test to choose a proper test method to simplify the testing process, decrease the number of test times and finally to reduce the testing effort.

When do regression testing, need to determine which set of test cases for the modules to be tested and which test cases need to be modified. Avoid designing test cases blindly and increasing the cost of test [6], and increase the test cases only in the case of the necessary.
Represent the information about relationships between the test cases and the modules in the form of two-dimensional array. Define a two-dimensional integer array, with m rows n columns of the matrix form to represent the m test cases and n associated modules.

4.Case study

4.1.Cases

For example, solving quadratic equation of one variable \( ax^2 + bx + c = 0 \), print outs.

Solving analysis:
- \( a=0 \), is not a quadratic equation.
- \( b^2 - 4ac = 0 \), there are two equal real roots
- \( b^2 - 4ac > 0 \), there are two unequal real roots
- \( b^2 - 4ac < 0 \), there are two complex roots.

Use C language to develop this program, and formatted, divided into basic modules by function, then coded them. As shown below.

```c
#include <stdio.h>
#include <math.h>
float a,b,c,x1,x2,disc,

float a,b,c,x1,x2,disc,realpart,imagpart;

equationout( )// ---------------------------------------------M1
{ printf("n please input a, b, c:\n");
scanf("%f,%f,%f",&a,&b,&c);
printf("the equation");
aif( );
}

aif( )// -------------------------------M2
{ if(fabs(a)<=1e-6)
   outnotquadr( );
else
   delta( );
}

outnotquadr( )// -------------------------------M3
{ printf("is not a quadratic");
}

delta( )// -------------------------------M4
{ disc=b*b-4*a*c;
if(fabs(disc)<=1e-6)
   outequar( );
else if(disc>1e-6)
   outdistr( );
else
   outcompr( );
}

outequar( )// -------------------------------M5
{ printf("has two equal roots:%8.4f\n",-b/(2*a));
}

outdistr( )// -------------------------------M6
{ x1=(-b+sqrt(disc))/(2*a);
x2=(-b-sqrt(disc))/(2*a);
```
printf("has distinct real roots:%8.4f and %8.4f\n",x1,x2);
}
outcompr( ) //---------------------------------------------M7
{  realpart=-b/(2*a);
  imagpart=sqrt(-disc)/(2*a);
  printf("has two complex roots is:\n");
  printf("%8.4f+%8.4fi\n",realpart,imagpart);
  printf("%8.4f - %8.4fi\n",realpart,imagpart);
}
main( )
{   equationout( );
}

4.2. Association analysis

According to analysis of program above, get 7 basic modules, M1, M2, M3, M4, M5, M6, M7 respectively. By analysis calling relationships on these 7 modules, set up association diagram for the corresponding modules. As shown in figure 1.

![Figure 1. Associated modules](image)

Use adjacency list to represent association diagram figure 1, as shown in figure 2.
Figure 2. Represent association diagram Figure 1 in Adjacency lists

According to represent association diagram figure 1 and adjacency list figure 2, get 4 execution paths, expressed as: PATH1=<M1, M2, M3>; PATH2=<M1, M2, M4, M5>; PATH3=<M1, M2, M4, M6>; PATH4=<M1, M2, M4, M7>.

According to four execution paths to design four test cases, TESTCASE1, TESTCASE2, TESTCASE3, TESTCASE4, correspond PATH1, PATH2, PATH3, PATH4 respectively. Design data for test case as follow:

- #TESTCASE1 Coefficient a=0
  #Expected: the equation is not a quadratic
  Test data: a=0, b=5, c=2

- #TESTCASE2 two equal real roots
  #Expected: the equation has two equal roots: -1.0000
  Test data: a=1, b=2, c=1

- #TESTCASE3 two unequal real roots
  #Expected: the equation has distinct real roots: -1.0000 and -7.0000
  Test data: a=1, b=8, c=7

- #TESTCASE4 two complex roots
  #Expected: the equation has complex roots:
  -1.0000+1.0000i
  -1.0000-1.0000i
  Test data: a=1, b=2, c=2

Association relationships between four test cases and other modules as shown in figure 3.
According figure 3, design a two dimensional array of integers, to store in the form of matrix, as shown in figure 4. Rows express the test cases, and columns say the basic modules. Value of 1 indicates an association relationship, while value of 0 means no. For example, the first rows express the association relationship between test case 1 and 7 modules, and the fifth columns represent the association relationship between the No.5 module and 4 test cases.

![Figure 3. Modules associate with test cases](image)

\[
\begin{pmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 1 & 1 & 0 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 0 & 1
\end{pmatrix}
\]

![Figure 4. Storage of Association relationship](image)

### 4.3. Test validation

Carry out these 4 test cases respectively, remember each passed module number of every case when it’s be run, so can get executed module series corresponding with each module. For example, run TESTCASE2, expected path of which is PATH2=<M1, M2, M4, M5>, if detected module series is PATH3=<M1, M2, M4, M6>, obviously the results do not match with the expected path. Know from analysis, there are changes in module M4, just examine codes of M4 only. If module M4 has errors, modify M4, and then start from M4, reverse traversal association graphics of modules to find out directly associated preceding module M2, and use forward traversal method to find out directly associated subsequent module M5, M6 and M7, and other non-related modules without regard. Directly associated sub-graph of module M4 are shown in figure 5.
When modify codes of M4, must consider and analysis M2, M5, M6, M7, whether these 4 associated modules will be affected. Regression testing need to test not only M4, but also M2, M5, M6, M7. TESTCASE1 are not associated with module M4, M5, M6, M7 known from figure 3, so regression testing for module M4, M5, M6, M7 just need to execute TESTCASE2, TESTCASE3, TESTCASE4.

When requirements change, if b2-4ac<0, print out “has two complex roots is:”, but not give two complex roots.

Run original codes, execute TESTCASE4 can show two complex roots, prove that procedure codes have mistakes, and corresponding module 7 need to modify.

Modify codes of M7 as follow:
```c
outcompr( ) //---------------------------------------------M7
{ printf( "has two complex roots is:\n" );
}
```

Choose test cases which associated with M7 to validate its correctness. According figure 3, TESTCASE4 can be chosen as it meets condition only.

Modified M7 may induce errors into other modules that associated with it. According associated graph figure 1, find out that M4 is associated with M7, and then do regression testing for M4, choose test cases TESTCASE2, TESTCASE3, TESTCASE4, which are associated with M4, execute them and get the correct results. This proves that to modify M7 do not affect associated module M4. But if module M4 has errors, must be modified, and associated modules M2, M5 and M6 need to be tested also.

5. Summaries

In develop and maintenance software, changes in requirements often lead to changes in corresponding program modules. Because of correlation among program modules, to modify a module may lead errors to related modules. In order to quickly find out the affected modules, put forward to establish association diagrams of modules to describe the modules structure and calling relationships. When a program changed, use methods of traversal associated graphs to find the corresponding modules, test to validate their correctness. To some extent, this method can reduce the testing effort, and improve the efficiency of software testing, particularly increase efficiency of regression testing.

Acknowledgment

This research was supported by Natural Science Foundation of Hainan Province (808154). Corresponding author is Chen Huandong, contacted by E-mail: chd@hainnu.edu.cn.
References


