International Journal of Security and Its Applications
Vol. 10, No. 6 (2016) pp.281-294
http://dx.doi.org/10.14257/ijsia.2016.10.6.27

Analysis on Influence Domain of Control Flow Modification in Regression Testing based on FCP

Mingjue Hu, Yongmin Mu and Aiping Xu

Beijing Information Science and Technology University, Beijing 100101, China

Abstract

After the version changes,locatingthe influence domain of control flow modificationis a hot issue in software testing, because it helps developersto solve software defects effectivelyand testersto determine the regression test cases. This paper puts forward that an algorithm toanalyzethe influence domain of control flow modification. It can accurately locate the functions which are affected by modified control statements. According to the control flow function calling paths algorithm, a path of nodes is extracted, where control word as well as the function name is regarded as a node,and extract the control block and analysis it, and a static function calling path with the control logic is obtained.Control flow modificationpoint algorithm is put forward to obtain the modified control statements, which is based on Longest Common Substring (LCS) algorithm. An algorithm is also proposed to determine the influence domain of control flow modification.Experimental results show that the proposedalgorithm can determine the influence domain, and provide support for test case reduction in regression testing effectively.

Keywords: *regression testing, control flow; function calling paths, the influence of modification*

1. Introduction

The FCP (Function calling paths) idea from the study of path coverage test optimization and regression testing. In it, a task can be regarded as a function calling path, or as a system is composed of a finite function calling paths. If all the function calling paths are covered, this system is reliable[1]. Analysis the control flow functions logical relationship between the functions and get the function calling paths. It is different from the function relation of inclusion, the function contains relationship just analysis the function which contains, and does not consider the logical relationship [2]. FCP idea makes the basic path test workload exponentially reduced, making path oriented testpossible[3].

Regression testing is the repeated tests that are conducted after the software system is modified or expanded, as to verify that the modification or expansion achieves the desired purpose and does not introduce new errors [4]. In theory, whenever the software is modified varied, regression testing should be carried out to verify the correctness and influence of the modifications or the variations, which are called changes. Practice shows that the software life cycle is subject to change at any stage. Therefore, it is not necessary tore-test all the test cases, and retesting only those affected works. When using the base path testfor re-testing, the mass number of the paths will bealmost impossible for manual testing.From the perspective offunction calling path, the code analysis granularity is extended from a statement to a function, avoiding the massive growth in the number of paths and ensuring the adequacy of software testing. Modifications can cause the changes in data flow, control flow and information flow, which are also the focus ofthe researches on changes. Control statements can change (or stop) program execution sequence and control the execution path of codes in the program. The modification of control

ISSN: 1738-9976 IJSIA

statements make functions calling pathsmodified, andthe workload will be increased greatlyif all the test cases are used for regression testing.Therefore, determining the influence domain of the control flow modificationand just re-testing the affected functions will reducethe workload[5]. The source of static analysis can get the control flow information and function call information, according to the idea of FCP access function call paths [6].

At present, according to the different classification methods, the software change is studied, and the changes are classified from three aspects: the behavior of the change and the life cycle of the software and the logical semantics[7][8].0 shows the classificationsfrom modification behavior, the software life cycle and logical semantic.Software life cycle is subject to changes at any stage, and code changes may lead to a variety of errors. If the influence domain of the control flow modification can be determined, then developers can solve software defects efficiently, and testerscan reduce the number of test cases[4].

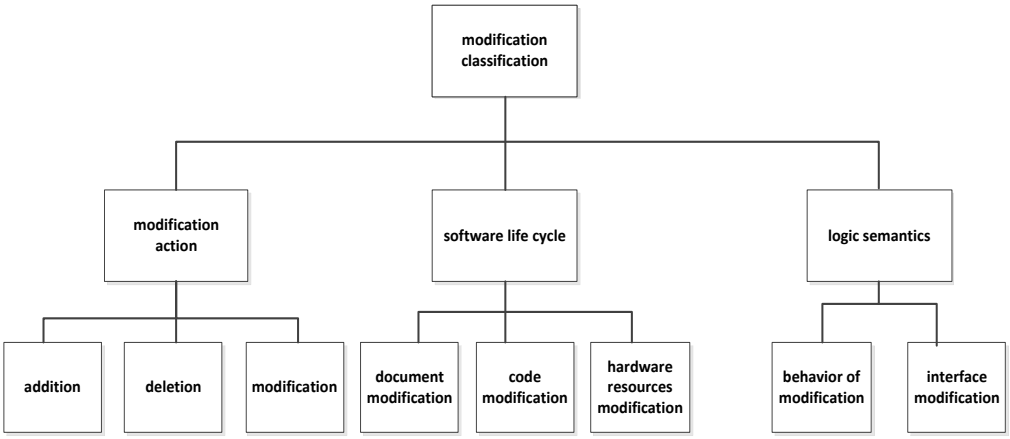

Figure 1. Classification of Modification

The analysis and study of modification influence was initialized by Harrold, Soffa in 1988, in Software Maintenance Conference where he introduced a technique to analyze modification influence based on the model, the main point was to use the data flow diagram to identify the affected definition-use (def- use) pairs and sub-paths[5].In 1994, Rothermel and Harrold proposed to apply modification influence analysis techniques in object-oriented testing environment in IEEE Software Maintenance Conference. Their algorithm constructed a program or class dependency graph, to indicate data and control dependency[9]. The main idea was that when a class is changed, the classes constructed through inheritance, polymorphism, dynamic binding, packaging, some other related classes, and the applications that use such classes are found out via the graph. The study of control flow modification influence is mainly based the control flow graph (CFG)[10].In 1997, GreggRothermel and MaryJean Harrold presented a CFG analysis method in computer science and software engineering methodology conference[11].The main point is that CFG uses simple or conditional states as nodes. The edges between nodes describe the control flow in different states, and are used to identify the affected control flow in a model.

Software code modification is mainly studied in two directions, object-oriented and process-oriented. The object-oriented study mainly focuses on classes and objects, including the study on the C ++ language which is an object-oriented language. In 1997, Abdunllah K. and White L. extended the concept offire wall to object-oriented software. Firewall separated the modified program modules from other modules. Modules inside the firewall can interact with the modified program modules, or the direct ancestor/descendant modules of the modified. Firewall enclosed the modified classes or objects, and the classes or objects that interact with them. The relationship between

objects and the one between classes are different, so the methods constructing an object firewall are different from the ones constructing a class firewall[12]. In 2001, Barbara G. Ryder and Frank Tip considered that the modification influence analysis provided feedback message for the semantic influence of a series of program changes[13]. This analysis can be used to design and select test cases for regression testing. Moreover, if a test fails, the modified subsets can identify errors; meanwhile the modified subsets can be combined with the secure test cases that are not affected. Hsia proposed a simple method that was completely different[14]. Some probes are inserted into the source code, to record the classes involved in each test cases. This information can be used to establish the correspondence between test cases and classes. When the modified point and modified classes are marked, it can be determined which test cases can be used to test the modified program which is regression testing. For the newly added functions, new test cases are also required to test them. In 1979, Mark Weiser first proposed program slicing, which creates a test case for slice s1, and if s1 contains a modified state, then *t* is selected as are-test test case [15]. Slicing takes statement into consideration, and analyzes the control dependence and data dependence between all the statements. The unnecessary re-tested parts in regression testing are adequately reduced, but the complexity is too high.

The CGF analysis method is also at statement-level, analyzing each statement in the program and picturing the specific CFG, which presents a new way to determine the influence domain of modified. The method proposed by Hsia is simple, but the modified points and modified classes must be known first, which is also at the class level and the retested subset is not minimal. In short, researchers have classified modifications from different angles and analyzed the influence domain of modified.

According to the characteristics of C ++, this paper directly obtains a function call graph with control logic using static analysis methods. Static analysis works at the function level. Based on the function calling path, this paper analyses the dependence and calling relationship between functions, reducing the complexity. An algorithm is designed to analyze the influence domain control flow modification and obtain the modified statements. According to the matching algorithm, the influence domain control flow modification is obtained.

2. Related Concepts

2.1. Analysis on the Static Extraction of Function Calling Paths

Definition 1: a control flow function calling paths (CFCP) area node sequence *(V_i = V_{i0}, V_{i1},... V_{im})* in a function call graph, where V_{im} is a function name or a number control word, and adjacent nodes represent the calling sequence [16] [17].

Definition 2: CN (control node), extract the control statement in CFCP. The node concludes class which the control statement included, and functions, number. It can be expressed as Node=<C,F ,CS, n>, C represents the class name in the control statement, F represents function's name which is the control statement included, CS represents control node, n represents the number in this function.

Definition 3: CBN (control block node), extract the control statement in CFCP neglect the control statement. It is an important basis for analysis of influence domain. The control block inside the source code called the control block. In order to make it easier to distinguish between control blocks, "{", "}" is the block of beginning and ending marks.

There are two methods for extraction of function calling paths, static analysis, and dynamic cartridge[18].The static analysis method first analysis of the source code, function calling graph is established using automata or other means. The dynamic method is inserted into the program pile or code pile in the source code or assembler code. During the execution of the program can collect each function on entering and exit time, information can be analyzed by function call path. An error may occur after version

changes, the dynamic instrumentation methods cannot be used to obtain the function call path, so the static analysis methods are applied. By scanning the source code, analyzing the control flow, and tracking the data flow, the corresponding mathematical model is constructed. Based on conditional relevance and data flow information, unreachable function paths are excluded, the control statements are analyzed to draw all the possible paths of static function calls.

The control words in c ++ include *if, else, while, do while, for, switch case default, break, continue, return, go to*. According to the function calling paths, a control word is treated as a function name in the program process, except some special cases that the local function calling relationship tree will be treated specially

Specific examples are described in Table 1 with the following code:

Table 1. Example of Codes

```
void   f1(){};
void   f2(){};
void   f3(){};
void   f4()
{
  int i=0;
  int j=1;
  if(i>0)
       f1();

     else if(j<5)
          f2();
     else
          f3();
}
int     main(int     argc,char*
argv[])
{
    f4();
    return 0;
}
```

According to CFCP, it can be expressed as the code shown in Figure 2:

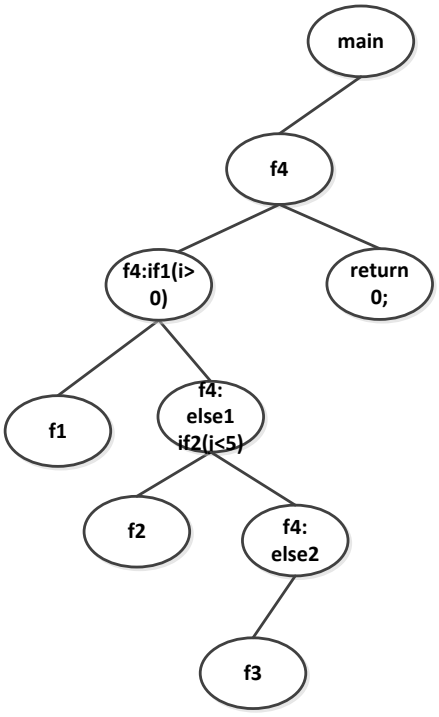

Figure 3. An Example of Control Flow Binary Tree

2.2. The Control Flow Modified Point Algorithm

After the version modification, the source codes of the old and new versions are analyzed to determine the modified control statements. And whether it modified need to special analysis the control statements. At present, there are more text, mainly divided into two categories, one is based on Distance Edit, such as LD algorithm; one is based on the Longest Common Subsequence, such as Needleman/Wunsch algorithm, etc., in addition to the use of linear space for the longest common sub sequence, Nakutsu algorithm using La (A, B) representation [19].According to the different function call path, each node is characterized by Lcs algorithm. The algorithm is used to solve the main problem into public string matrix L (s, t) problems in the use of theorem elements of the matrix during the process according to the two situations were calculated for each column element until the null s+1 found the line out of the loop that matrix L (k, t), B[L (1, t-s+1 (2)]B[L, t-s+2)]... B [L (s, t)] is A, LCS,, LCS,, B, and S.

(1) while (i<k),L(k, i)=null,

(2) while(L(k, i)=k),L(k,i+1)=L(k,i+2)=…L(k, t)=kl;

Definition 4: Subsequence. Given a string A=A[1]A[2]... A[m] (A[i] is in the letters, the character set l<= i<m =A, A, said the length of A), String B is a sub sequence of A, which is B=A[1 i]A[2 i]…A[k i], 1 i< 2 i<... < I k and k<=m.

Definition 5: Common Subsequence: Given some strings A, B, C, The common sub sequence of B and C is that C is not only the sub sequence of A, but also B.

Definition 6: Longest Common Subsequence: Given some strings A, B, C,

C is known as the longest common subsequence of A and B, C is refers to the public sequence A and B, and for any public A sequence of D and B, both D < = C. Given string, A and B, A =m, B =n,

If (m<=n), LCS problem is to find the common sub sequences of A and B.

Definition 7: Given two string A=A[1]A[2]…A[m]and B=B[1]B[2]…[n], A(1:i)is Continuous sub sequence of A,A[1]A[2]…A[i];B(1:j)s Continuous sub sequence of B,B[1]B[2]…[j].Li(k)Represents a string for all A (1:i) with a length of K. Li(k)=Minj (LCS(A(1:i), B(l: j))=k).

Theorem 1:∀ i∈[1, m],Li(l)<Li(2)<Li(3)<…<Li(m).

Theorem 2:∀i∈[l, m-1], ∀k∈[l, m], L i +1(k)<= Li k).

Theorem 3:∀ i∈[l, m-1], ∀ k∈[l, m-l], Li(k)< L i +1(k+l).

Theorem 4:if L i +1(k)exists,

L i +1(k)=Min(j, i L (k)).jis the smallest integer satisfying the following conditions, A[i+l]=B[j] and j>Li(k-1).

LCS matrix L (P, m), the following table is shown in the following figure.

	i=1	2	3	…	m
k=1	L(1,1)	L(1,2)	L(1,3)	…	L(1,m)
2	null	L(2,2)	L(2,3)	…	L(2,m)
3	null	null	L(3,3)	…	L(3,m)
…	…	…	…	…	…
p	null	null	null	null	L(p,m)
p+1	null	null	null	null	null

Figure 3. The Matrix of LCS

Elements in matrix (k, i)=Li(k),(1<i<=m, 1<k<=m),Null indicates that L (k, I) does not exist. If i<k, L(k, i)does not exist. From the analysis of Figure 3, we can only analyze the value of the matrix, and then, according to the characteristics of matrix and the characteristics of the longest common sequence, determine the sequence of matrix. The structure of the control statement block may be nested multiple control statements, simple statement block, including the multi - layer call function statement block, for different situations. For multi - layer nested control statements, the function and control statements are used as the principle of node extraction.

The following code is nested control statements, the functions calling paths in Figure 4 as shown below:

```
void  f4()
{
  int i=0;
  int j=1;
  if(i>0)
   {f1();
    While(a<b)
    {  f2();
       a++;
     }
    }
```

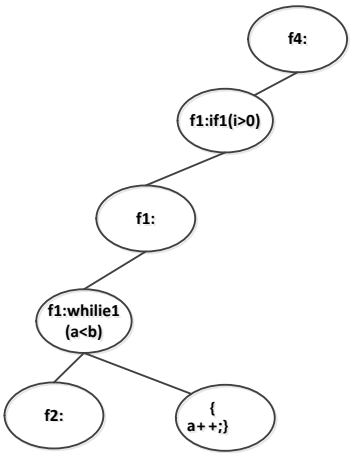

Figure 4. Nested Call Control Statements

From Figure 4 can be found if the control block *a++* changes, the control statement impacted, and regarded control block as modified. The control block parent node is *f1:if1 (i>0); f1:whilie1 (a<b),*and it need to marked, as the analysis on the basis of the influence domain.

2.3. The Matching Algorithm of Influence Domain of Control Flow Modification

According to the control flow modified point algorithm, the modified control statements are obtained. The static function calling path algorithm can obtain the function calling paths with control words. The control flow modification influence domain matching algorithm is designed to obtain the influence domain of control flow modification.

Definitions 8: F = {F_1, F_2,,,F_n} is the set of modified control statements. The S= {S_1,S_2,,,S_n }is the collection of static paths obtained by static analysis. If $F_1$is the modified

control word, all the static paths containing *F1* in S are modified paths, expressed as E = {E_1, E_2,…}. The Functions after the modified function *F1*are modified functions in collection E. In 0, the control statement $f_4\_if_1$ is modified, and then the influence domain of the modification is {f_2, $f_4\_else_2$, f_3}.

Definitions 9: $\sum paths_{modified}$ is the total of modified function calling paths, $\sum paths_{find}$ is the total of modified control paths, $p=\frac{\sum paths_{(find)}}{\sum paths_{(modified)}}$ is the ratio of modified paths by algorithms.

Definitions 10: $\sum nodes_{modified}$ is the total of modified control nodes, $\sum nodes_{find}$is the total of modified control nodes by algorithms, $n=\frac{\sum nodes_{find}}{\sum nodes_{modified}}$ is the ratio of modified paths by algorithms.

3. The Analysis Algorithm of Influence Domain of Control Flow Modification

According to the static control flow algorithm, the static function calling paths with control words got by the source codes. The modified control statements can be obtained by the modified control flow. Meanwhile, the control words are numbered. The scopes of different control words, determine the influence domain of control flow modification. The framework of the algorithm is shown by 0.

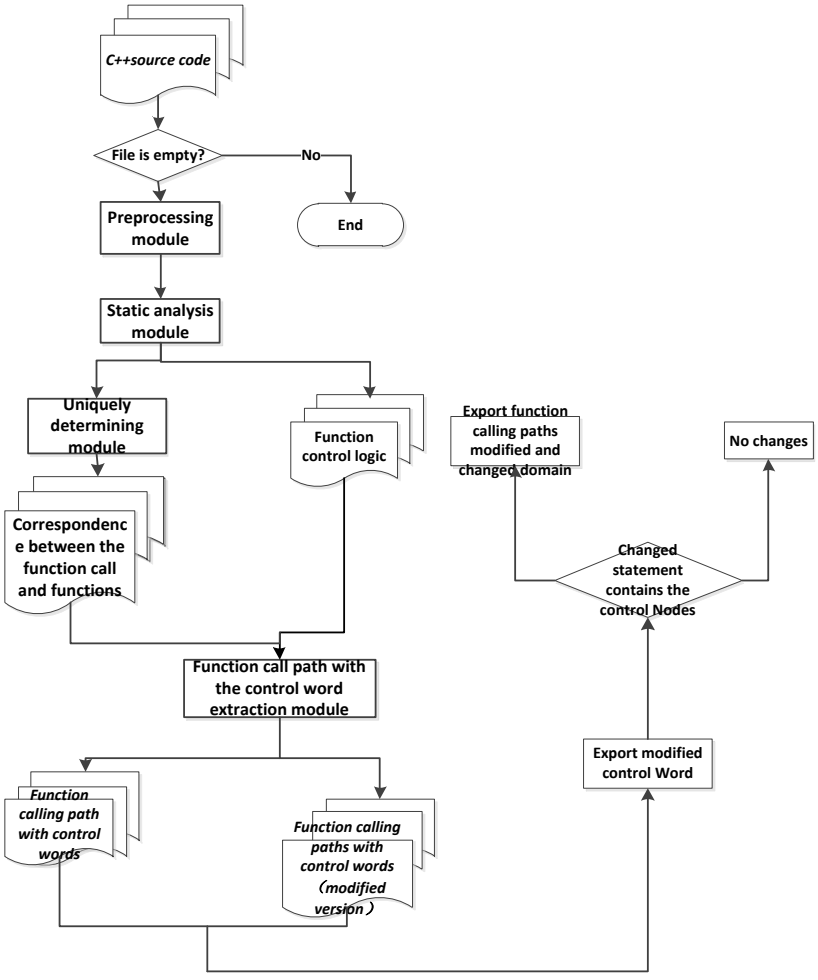

Figure 4. The Framework of the Analysis Algorithm of Influence Domain of Control Flow Modification

3.1. The Control Flow Function Calling Paths Algorithm

The input of the algorithm is C++source code files of the older version. The pre-processing module removes comments and filter redundant codes, and then a single or multiple source code file scan be obtained. Lexical and pattern matching are conducted for the source code file, to get the logical structure of the function, the function call information, function prototypes, variables, classes and control words The control words are numbered, and the static analysis information of the program is obtained. The static analysis information will be overloaded and uniqueness determined to the modified calling points. The class, function name and function list of the function corresponding to the function calling point are determined. The corresponding relations between the function calling points and the control words as well as the function are obtained. The logical structure is combined with the control flow function calling information to get function calling paths with local control words. Finally, starting from the main function, the function calling relationships with local control words of all functions are iteratively expanded to obtain the global control flow function calling path and global control flow binary tree. The above process can be expressed by Algorithm 1.

Algorithm 1:The control flow function calling paths algorithm

```
1:  input: C++ source files
2:  output: control flow function calling paths
3:  whilefunction read the function node and control node
4:  if ( file is null )then return NULL; // no action
5:  elseif match function definition or control node or control block
    thenbeginNum= line number of function entry or control node begin or
    control block begin
6:  endNum= line number of function exit or control node exit or control
    block exit
7:  If function or control node or control block has not created then
8:  create function node or control node or control block node with it's name
    and location
9:  end if
10: Call LinkFunction(Fn, beginNum, endNum)
11: end if
12: end while
```

The whole control flow function calling paths get by the linked functions, the linked function designed as follows:

Algorithm 1:The linked functions algorithm

```
1:  input: the local function calling paths, the parent nodes
2:  output:the next function or functions
3:  currentNum = beginNum;  lastfn=fn;
4:  while currentNum< endNum
5:  if match function subfn or control node or control block then
6:  Link subfn or control node or control block and fn
7:  Call Linkfunction(subfn, currentNum - beginNum, endNum)
8:  Lastfn=sunfn;currentNum++;
9:  end if
10: Call LinkFunction(fn, beginNum, endNum)
```



```
11: end while
12: return lastfn
```

3.2. The Control Flow Modified Point Algorithm

The control flow modified point algorithm is a new version of the old and the new version contains a control statement and control statements whether the block is changed, the design is mainly based on the LCS algorithm to determine whether the two line of code is completely same and design. Description of the improved LCS algorithm, so you can set up p=Maxk (L (k, m) = (null), L can prove that the L matrix*L (P, m)* where the diagonal. *L(1,m-p+1),L(2,m-p+2)…L(p-1,m-1),L(p,m),*and the Corresponding

sub sequence is *B[L(1,m-p+1)]B[L(2,m-p+2)]…B[L(p,m)]*.IT is the *LCS of A and B* ,the *p* is length of *LCS*. Thus, the solution of the LCS problem is transformed into the solution of the $m \times l$matrix. L (B, L, A) implementation of the pseudo code is shown as follows: The *A* string input length is *m*, *B* string input is *n*, the longest common sub sequence of *A* and *B* is *LCS(A,B).*According to the principle of its solution can quickly determine the longest common sub string, determine the change point.

Algorithm 2:The control flow modified point algorithm

```
1:  input: OldVersionText, NewVersionText
2:  output:CurrentLD
3:  begin:
4:  L(A,B,L){//the length of A is m
5:  for(k=1;k<=m;k++){  //Control line number, loop to find a trace node
6:  for(i=1;i<=m;i++){ //Number of internal loop control loops
7:  If(i<k) //Determine if the I value is less than the column number n is not the last
    column.
8:  L[k][i]=N; //i<k,L(k,i)=null, N stands for Infinity
9:  if(L[k][i]==k) //L(k,i)=k
10: L(k,i+1)=L(k,i+2)=…L(k,m)=k; // Assignment for each row
11: for(l=i+1;l<=m;l++)//Calculate and fill in the cell
12: {
13:  L[k][l]=k;//Assignment of values in a matrix
14: Break;
15: }//If the value is k, then jump out of the loop.
16: for(j=1;j<=n;j++)
17: {//Realization of Theorem 4 in 4.1.3
18: if(A[i+1]==B[j]&&j>L[k-1][i])
19: {
20: L[k][i+1]=(j<L[k][i]?j:L[k][i]);//Assignment for each value in the matrix.
21: break;
22: }//forloop ends
23: if(L[k][i+1]==0)
24: L[k][i]=N;//If the n value is 0, the maximum value of the N assignment
25: if(L[k][m]==N)
26:     {
27: p=k-1;break;//Value of P
28:     }
29: p=k-1;//Finally the value of P is assigned to n-1
30: }
```

3.3. The Matching Algorithm of Influence Domain Control Flow Modification

There are some control words that can be treated using the same method, just as *if, else, while, for, switch, case, default, do, while*. The influence domain of some control words needs specific analysis, just as *break, continue, go to, return*. The algorithm is shown as follows:

Algorithm 3: The matching algorithm of influence domain control flow modification

1: **input S:**collection of (modified path) s,F=*{F_1,F_2,,,,,F_n}*(collection of modified statements)
2: **output:**the influence domainof *{ F_1...F_n}*
3: **begin:**
4: **if***Fi*∈*{ if else ,while; for, switch case default.}* Then the route is after the *Fi;*
5: **elseif***Fi* ∈*{ continue, goto, return }*the route is the function which include *Fi*
6: **else***Fi*∈*{break}*
if it is in switch ,the route is the next node and the switch case code
7: **End**

4. The Experiment and Evaluation

Specific codes are designed according to the above described algorithm, and influence domain of control flow modification is judged whether to be consistent with the expected results. Experiments are carried out with an example, and the source codes are shown Table 2. This example contains many control words such as *if, else, etc.* This example is the classical sorting algorithm.

Table 2.The Test Codes

```
#include <iostream>
using namespace std;
void SelectSort(int arr[], int length)
 {
    int temp, min;
    for (int i = 0; i < length - 1; ++i)
    {
        min = i;
       //Finding the minimum value
       for (int j = i + 1; j < length; ++j)
        {
            if (arr[j] < arr[min])
                min = j;
        }
        // exchange
       if (min != i)
        {
            temp = arr[i];
            arr[i] = arr[min];
            arr[min] =temp;
        }
    }
 }
int main()
 {
    int arr[10] = {2, 4, 1, 0, 8, 4, 8, 9, 20, 7};
    SelectSort(arr, sizeof(arr) / sizeof(arr[0]));
    for (int i = 0; i < sizeof(arr) / sizeof(arr[0]); ++i)
    {
        cout<<arr[i]<<" ";
    }
    cout<<endl;
    return 0;
 }
```

```
#include <iostream>
using namespace std;
void SelectSort(int arr[], int length)
 {
    int temp, min;
    for (int i = 0; i < length - 1; ++i)
    {
min = i+1;
        // Finding the minimum value
for (int j = i + 1; j < length; j++)
        {
            if (arr[j] < arr[min])
                min = j;
        }
        // exchange
if (min != i+1)
        {
            temp = arr[i];
            arr[i] = arr[min];
            arr[min] =temp;
        }
    }
 }
int main()
 {
int arr[10] = {2, 4, 1, 8, 4, 8, 9, 20, 7,12};
    SelectSort(arr, sizeof(arr) / sizeof(arr[0]));
    for (int i = 0; i < sizeof(arr) / sizeof(arr[0]); ++i)
    {
        cout<<arr[i]<<" ";
    }
    cout<<endl;
//end
 }
```


According to the characteristics of the algorithm, the algorithm is designed and implemented. The design page is shown in Figure 6:

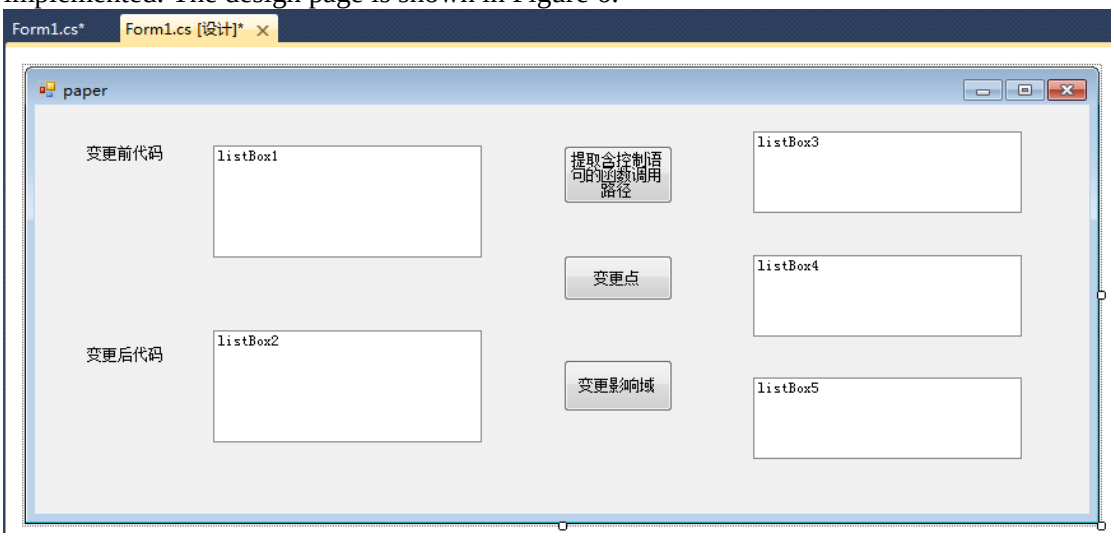

Figure 6. Programming Interface

The control flow function calling path algorithm can obtain the function calling paths with control words. As shown in 0

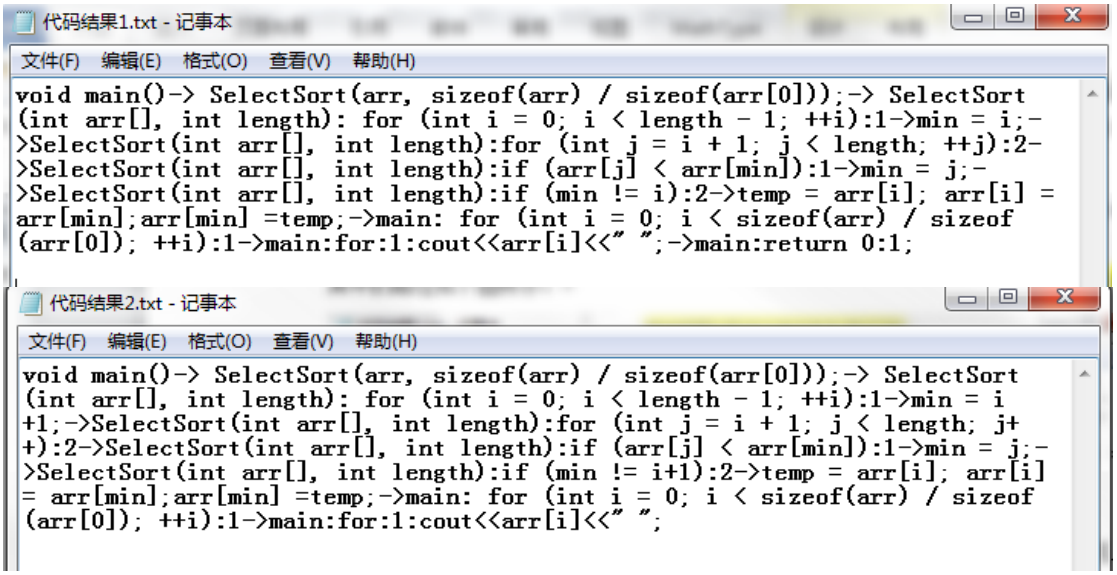

Figure 7. The Function Calling Path with Control Words

The control flow modified node algorithm can get the modified control statements, as is shown in 0.

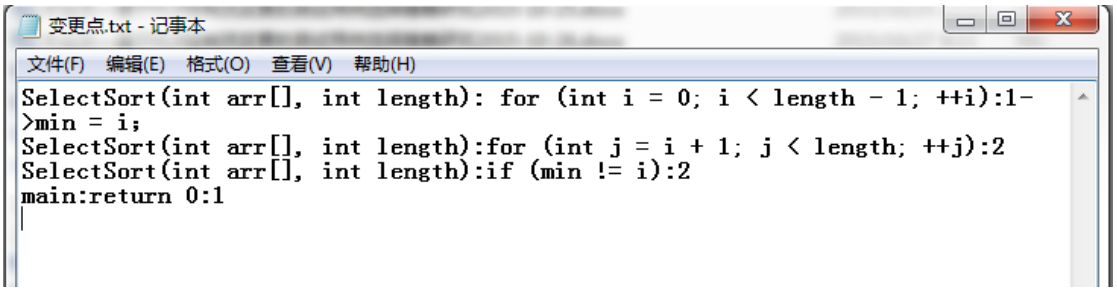

Figure 8. The Modified Control Words

The influence domain will be obtained by the matching algorithm, as shown in 0.

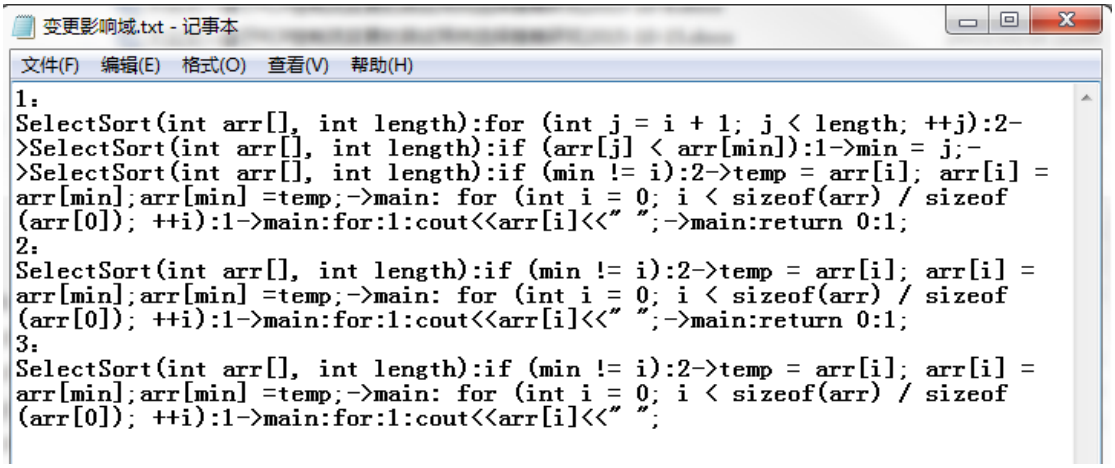

Figure 9. The Influence Domain of Control Flow Modification

The experiments focused on the analysis and verification whether the proposed algorithm scan accurately identify the modified control statements and influence domain of control flow modifications. Due to space limitations, the illustrated codes are simple and representative as the experimental data. In order to better experiment the algorithms, so through several sorting algorithms commonly used as the accuracy of the test algorithm, as shown in Table 3.

Table 3.The Results about Sort Algorithms

Sort algorithms	Node modified number	Node find number	n(probability)	Path modified number	Path find number	P(probability)
Bubble sort	21	19	90.48%	89	75	84.3%
Insertion sort	15	13	86.7%	92	85	92.4%
Quick sort	18	18	100%	75	55	73.3%
Heap sort	22	21	95.4%	102	96	94.1%
Merge sort	10	9	90%	86	73	84.9%
Heap adjust	17	15	88.2%	164	152	92.3%

Experiments show that the algorithm can accurately identify the modified control statements and influence domain of control flow modifications. And the result is same as artificial analysis.

5. Conclusion

Frequent changes in software make the work load of the regression test soar. It will significantly reduce the workload of testers to determine the influence domain of control flow modification. Developers can quickly locate software defects and improve the efficiency of dealing with software defects. For the problem of influence domain of software modification in regression testing for C++, this paper puts forward an algorithm to analyze the influence domain of control flow modification, which is an exploratory step in this filed. The function calling paths with control words are obtained based on the analysis on control flow function calling paths. The control flow modified point algorithm is designed to determine the modified control words. The influence domain of control flow modificationis obtained by matching function call paths. Experimental data show that the proposed algorithm can correctly identify the modified control words, and precisely get the modified influence domain. Using modified influence domain, test cases can be reduced in regression testing, and test efficiency can be improved.

References

[1] Y. Zheng, Y. Mu and Z. Zhang, "Research on the Static Function Call Path Generating Automatically", The 2nd IEEE International Conference on Information management and engineering[J], vol. 12, **(2012)**, pp. 236-240.
[2] S. Baharom and Z. Shukur, "An experimental assessment of module documentation-based testing", Inform Software Tech, vol. 53, **(2011)**, pp. 747–760.
[3] M. Yongmin, Z. Yuhui, Z. Zhihua and L. Mengting, "The Algorithm of Infeasible Paths Extraction Oriented the Function Calling Relationship [J]", Chinese Journal of Electronics, vol. 21, no. 2, **(2012)**, pp. 236-240.
[4] Z. Zhihua and M. Yongmin, "Research of Path Coverage Generation Techniques Based Function Call Graph", Acta Electronica Sinica[M], vol. 138, no. 8, **(2010)**, pp. 1808-1811.
[5] S. Harrold, "An Incremental Approach To Unit Testing During Maintenance", Proc, Conference on Software Maintenance, **(1988)**, pp. 362-367.
[6] ZX. Chen, JY. Zhan and ZB. He, "Method for Static Function Call Analysis with Control Flow [J].", Computer Engineering, vol. 37, no. 9, **(2011)** May.
[7] Y. Ming and X. Li, ResearchandImplementationofOrient — objectSoftwareChangeImpactAnalysisTool, **(2009)** March.
[8] HB. Yang and C. Zhen, "Research on analysis model for code change impact in object oriented program", Computer Engineering and Design, **(2010)**, vol. 31, no. 19.
[9] G. Rothermel and M. J. Harrold, "A safe, efficient regression test selection technique", ACM Transactions on software Engineering and Methodology, vol. 6, no. 2, **(1997)**, pp. 173-210.
[10] G. Rothermel and M. J. Harrold, "Selecting Regression Tests for Object-Oriented Software", Proc.of IEEE International Conference On Software Maintenance, **(1994)**, pp. 14-25.
[11] Y. Mu and M. Liu, "The Static Analysis of Function Calling Paths Based on Control Logic for C++", 2013 International Conference on Control Engineering and Information Technology, pp. 403-408.
[12] K. Abdullah and L. White, "A Firewall Approach for the Regression Testing of Object-Oriented Software", International software Quality Week Conferences, **(1997)**.
[13] D. Kung, J. Gao and P. Hsia, Class Firewall, test order, and regression testing of object-oriented. Programs, Journal of Object-Oriented Programming, vol. 8, no. 2, **(1995)**, pp. 51-65.
[14] F. Fei, S. JiaPian, W. Lifu and Y. Fuqing, "Object-oriented software regression testing technology research", Journal of software, **(2001)** December, vol. 12, no. 3.
[15] Li Jin-nuo, "A Constraint-based Analysis Method to Simplify Control Flow Graph", Computer and Modernization, **(2013)** March, vol. 10.
[16] A. Xu,Y. Mu, Z. Zhang and H. Li, "The dynamic function calling path generationbased oninstrumentation", Applied Mechanics and Materials, vol., **(2014)**, pp. 568-570.
[17] J. Hua, H. Anqi, W. Meijia, W. Zheng and W. Xueling, "String similarity algorithm based on improved edit distance", Computer Engineering, **(2014)**.
[18] Y. Mu and Z. Yang, "Verify consistency of software implementation and design basedon function call path", [J] Science China: Information Science, vol. 10, **(2014)**, pp. 190-1304.
[19] MF. Zhao, XM. Wang and L. Rui, "Analysis of text comparison algorithm", Electronics World, vol. 4 **(2014)**, pp. 174-174.

