Journal of Computer Applications
计算机应用,2020,40(10):3025-3033
ISSN 1001-9081
CODEN JYIIDU
2020-10-10
http://www.joca.cn

文章编号:1001-9081(2020)10-3025-09
DOI:10.11772/j.issn.1001-9081.2020030311

基于控制流的软件设计与实现一致性分析方法

张家奇*,牟永敏,张志华
(网络文化与数字传播北京市重点实验室(北京信息科技大学),北京 100101)
(*通信作者电子邮箱879632300@qq.com)

摘　要:针对目前软件设计与实现一致性检测方法需要大量模板集,难以一般化的问题,提出一种基于控制流的软件设计与实现一致性分析方法。首先,将设计文档伪代码和程序源代码转换为具有相同特征的中间表示,从中间表示中分别提取设计特征和实现特征,特征包括能反映系统结构的函数调用关系和能反映函数内部结构的控制流信息;然后,根据设计特征和实现特征分别建立设计特征模型和实现特征模型;最后,通过计算特征相似度来度量特征模型的相似度,进而获取一致性检测结果。实验结果表明,所提方法在软件实现的函数调用关系与设计不一致时,能够正确检测不一致函数调用关系;而在软件实现的函数调用关系与设计一致时,能够正确检测函数内部结构的不一致情况,准确度达到了92.85%。该方法可以有效获取一致性检测结果,不需要模板集,具有更优越的一般性。

关键词:控制流;函数特征;函数调用;伪代码;一致性检测
中图分类号:TP311.56　　**文献标志码**:A

Consistency analysis method of software design and implementation based on control flow

ZHANG Jiaqi*, MU Yongmin, ZHANG Zhihua
(*Beijing Key Laboratory of Internet Culture and Digital Dissemination Research*
(*Beijing Information Science and Technology University*), *Beijing* 100101, *China*)

Abstract: The current consistency detection methods of software design and implementation require a large number of template sets and are difficult to generalize. In order to solve these problems, a consistency analysis method of software design and implementation based on control flow was proposed. Firstly, the pseudocode of the design document and the source code of the program were converted into the intermediate representations with the same features, and the design feature and the implementation feature were respectively extracted from the intermediate representations. The features include the function call relationship which can reflect the system structure and the control flow information which can reflect the internal structure of the function. Then, the design feature model and the implementation feature model were respectively established according to the design feature and the implementation feature. Finally, the similarity of the feature model was measured by calculating the feature similarity, so as to obtain the consistency detection result. Experimental results show that this method can correctly detect the inconsistent function call relationship when the function call relationship realized by the software is inconsistent with the design, and can correctly detect the inconsistency of the internal structure of the function when the function call relationship realized by the software is consistent with the design, with the accuracy reached 92.85%. This method can effectively obtain the consistency detection results without any template set, and has superior generality.

Key words: control flow; function feature; function call; pseudocode; consistency detection

0　引言

近年来,随着软件产业化的发展,软件测试逐渐成为人们关注的焦点。软件测试的目的在于检验某个软件系统是否满足规定的要求,即验证软件设计与实现是否一致。对于小规模的软件系统,人工可以完成一致性的验证。对于软件规模和复杂度较高的软件系统,人工难以完成一致性的验证。自动完成软件设计与实现的一致性验证工作变得越来越重要。

软件实现与设计一致性验证是指对已完成的软件系统,验证其功能实现是否按照软件设计说明书的要求完成,是一种非常重要的测试[1]。Riedl-Ehrenleitner等[2]提出面向设计模型和代码的一致性检测方法,通过一致性规则实时验证代码和设计模型的一致性,用于模型驱动的软件开发,并在开发过程中实时检测代码与设计模型的一致性,没有考虑代码的编写细节。曾一等[3-4]提出基于统一建模语言(Unified Modeling Language, UML)模型与Java代码一致性检测的方法,对设计和实现的动态交互信息进行一致性验证,但没有对具体方法的实现进行验证。牟永敏等[1]提出一种针对文档和

收稿日期:2020-03-18;**修回日期**:2020-05-27;**录用日期**:2020-05-28。
基金项目:网络文化与数字传播北京市重点实验室开放课题(5221935409)。
作者简介:张家奇(1995—),男,山西临汾人,硕士研究生,主要研究方向:软件测试;　牟永敏(1961—),男,山东烟台人,教授,博士,主要研究方向:软件测试、大数据;　张志华(1971—),女,黑龙江宝清人,副教授,硕士,主要研究方向:软件测试、计算机软件。

源代码的测试方法，该方法基于函数调用路径[5]，从设计文档中获取函数的功能描述，建立设计功能簇模型；从源代码中提取实现函数特征，对比已知的模板集获取实现函数的功能描述，建立系统功能簇模型；通过验证两个模型的一致性完成设计与实现一致性验证问题。但是该方法需人工分析设计文档以及大量的模板集，限制了模型建立的效率，并且该方法没有对比设计函数的实现细节与实际函数的实现细节是否一致。

本文参考文献[1]中提出的一致性验证方法，提出了一种面向伪代码的函数特征提取方法，选取设计文档中的伪代码和程序源代码为研究对象，通过验证伪代码和源代码中函数特征的一致性完成设计与实现的一致性检测。此处申明本文研究的前提为设计文档中包含伪代码，且伪代码语义正确。

1 本文方法

本文研究思路基于软件设计的模块化设计原理。模块化原理就是把程序划分成若干个模块，每个模块完成以一个子功能，把这些模块集中起来组成一个整体，可以完成指定的功能，满足问题的需求[6]。模块化原理不仅使软件结构清晰，而且让软件容易测试和调试。

软件设计过程中遵循模块化原理。同样，设计文档也遵循模块化原理，详细描述各个模块的实现算法、逻辑流程等。从文档对各模块的伪代码描述和程序源代码的各模块中提取函数特征，并建立设计特征模型和软件实现特征模型，通过计算模型的相似度解决设计与实现的一致性验证问题。本文一致性检测框架如图1所示。

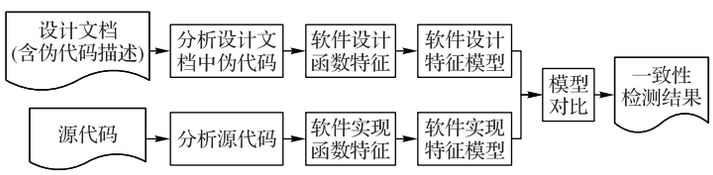


图1 一致性检测框架

Fig. 1 Framework of consistency detection

2 函数特征的提取

在软件实现和设计一致性检测中，伪代码和源代码的表征方式决定了软件设计信息和实现信息的利用程度。在代码克隆领域，代码的表征方式分为文本[7]、词汇[8]、语法[9]和语义[10-11]四个层次[12]。由于伪代码变量不需要声明、缺乏语法规范的特点，文本、词汇的表征方式不能充分利用伪代码的信息。语法的表征方式，例如基于语法分析树的方式，会造成一致性验证算法复杂度的提升。语义的表征方式，利用代码语义信息进行验证，其中，语义信息是指能够反映一段代码功能的信息，比如代码的控制流和数据流[12]971。本文采用基于语义的表征方式，语义信息为代码的控制流，基于代码控制流获取函数特征。

函数特征分为外部特征和内部特征。外部特征为函数间调用关系，内部特征为函数控制流信息，包括基本块序列和控制流图。函数间调用关系反映了系统的结构[13]，每个函数内部的控制流信息反映了函数的内部结构，因此选其作为函数特征并建立特征模型。函数特征提取具体过程如下：

2.1 中间表示的生成

伪代码和源代码中的控制结构会造成控制流的改变，难以直接从中获取控制流信息，因此将其转换成一种按顺序执行的语句序列——中间表示。该过程包括伪代码中间表示的生成和源代码中间表示的生成。

2.1.1 伪代码中间表示的生成

伪代码中间表示(Pseudocode Intermediate Representation，PIR)是一种按顺序执行的语句序列。常见的中间表示有很多形式，如三元式、四元式、后缀式、语法树等[14]，本文采用三元式。在生成PIR过程中，首先对伪代码进行语法规范，根据语法规范对伪代码进行词法和语法分析，生成语法分析树；然后遍历语法分析树，根据语法制导翻译将其中的控制结构转换为顺序执行的语句和跳转语句——中间代码；最后对中间代码划分基本块，得到中间表示。

1)伪代码语法规范。

由于人们在描述算法时使用的伪码并不相同[6]128，难以一般化，本文使用一种较为通用语法规则来规范伪代码。语法规则用扩展巴科斯范式表示，语法规则产生式如下：

```
prog : (NL* functiondef)*;
functiondef :
    FUNCTION id '(' args?  ')' NL stats END FUNCTION NL+;   (1)
args : (expr) (',' expr)*;
stats : stat*;
stat :
    IF expr THEN NL stats END IF NL                         (2)
    | IF expr THEN NL stats ELSE NL stats END IF NL         (3)
    | FOR assignexpr TO expr DO NL stats END FOR NL         (4)
    | WHILE expr DO NL stats END WHILE                      (5)
    | REPEAT NL stats UNTIL expr NL                         (6)
    | id '<-' expr | id '<- ' functionCall                  (7)
    | id'(' args?  ')'                                      (8)
    | BREAK NL | CONTINUE NL
    | RETURN expr?  NL | NL;
expr : logicalOr;
logicalOr : logicalAnd | logicalOr OR logicalAnd;           (9)
logicalAnd : relation | logicalAnd AND relation;            (10)
relation : arithmetic | relation REL arithmetic;           (11)
arithmetic : primaryexpr | arithmetic OP primaryexpr;
primaryexpr: id | number | '(' expr ')';
OP : '*'|'/'|'%'|'+'|'-';
REL : '<''='? |'>''='? |'='|'<>';
```

其中：每一个分号语句为一条规则，大写字母开头为词法规则，小写字母开头为语法规则。NL表示换行符，语法规则中的单引号内容和大写单词表示匹配对应的字符；prog表示

伪代码的一个逻辑结构,包括一个或多个函数定义;functiondef为函数定义,包括语句集stats;id表示标识符;number表示数字。为方便后文引用,语法规则结尾添加编号。

语句集stats包含if语句、if_else语句、while语句、repeat语句、for语句、赋值语句、函数调用语句等。expr为表达式,包含逻辑表达式、关系表达式、算数表达式等。

2)生成语法分析树。

使用ANTLR(ANother Tool for Language Recognition)生成语法分析树和树遍历器。ANTLR是一款强大的语法分析器生成工具,可以根据语言的语法生成一个语法分析器和语法分析树遍历器,并自动建立语法分析树。

3)语法制导翻译。

语法制导翻译是在语法分析过程中,根据每个产生式所对应的语义子程序进行翻译的方法[14]22。通过语法制导翻译可以将伪代码语句翻译为按顺序执行的语句。使用ANTLR内置的树遍历器遍历语法分析树,树遍历器采用递归下降的分析方法遍历语法分析树,在遍历的过程中使用语法制导翻译,生成按顺序执行的中间代码。

伪代码中函数定义语句、函数调用语句、赋值语句、关系表达式并不影响模块的控制流,这些语句的产生式对应语法规则(1)、(7)、(8)、(11),其语义子程序(翻译规则)不需要对原语句进行处理,即在中间表示中复用原语句。

伪代码中布尔表达式经常用来改变控制流和计算逻辑值,但其使用意图要根据其语法上下文来确定,例如关键字IF后面的布尔表达式用来改变控制流,而一个赋值语句右部的布尔表达式用来表示一个逻辑值。因此将控制语句(IF-ELSE、WHILE语句等)以及布尔表达式结合进行语法制导翻译,再将翻译的结果填写在中间表示中。

布尔表达式是布尔运算符作用于布尔变量或关系表达式上而构成的[14]315,控制结构是布尔表达式作用于控制语句而构成的。布尔表达式生成文法的产生式对应语法规则(8)~(10)。控制结构生成文法的产生式对应语法规则(2)~(6)。由于篇幅限制,只列出一种while循环语句,for和repeat循环语句语义子程序与while循环语句类似。

由于逻辑运算短路的特点,伪代码中会出现短路代码,进而影响程序控制流。因此将运算符and、or、not翻译成goto语句。运算符本身不出现在中间表示中,布尔表达式的值通过语句序列中的位置来表示。如表1所示,使用转移语句来翻译控制语句以及布尔表达式。

表1　转移语句

Tab. 1　Transfer statements

转移语句	类型	含义
If x goto L1	条件转移	如果x为真,下一步执行标号为L1的语句;
else goto L2		如果x为假,下一步执行标号为L2的语句
goto L3	无条件转移	下一步执行标号为L3的语句

PIR中的语句将按顺序执行,当遇到以上语句时,执行过程就会跳转。在一个语句前加上前缀"Li:",表示将标号Li附加到该语句,其中i为正整数。同一个语句可以同时拥有多个标号。

为方便描述,需赋予文法符号(非终结符)属性:text、code、begin、next、true、false。其中,text表示语句或表达式未经处理的token流,code表示语句翻译后的代码,begin、next、ture、false表示标号。如stat. code表示语句stat翻译后的代码;stat. begin表示语句stat的标号;stat. next表示语句stat后一条语句的标号;expr. true表示布尔表达式expr为真时跳转到目标语句的标号。后文图中"="表示赋值,newlabel表示产生一个新的前缀标号"Li";gen()表示一个语句,如:假设expr. true的值为"L1",则gen("goto" expr. true),表示语句"goto L1"。

由于关系表达式、函数定义语句、函数调用语句、赋值语句对应生成文法的语法制导规则不需要对原语句进行特殊处理,因此本文只列出语法规则(1)所对应语法制导规则。如图2(a)所示。

```
stats.begin = newlabel;
stats.next = newlabel;
functiondef.code =
    gen("FUNCTION")
  ‖ gen(functiondec.text)
  ‖ gen(stats.begin)
  ‖ gen(stats.code)
  ‖ gen(stats.next)
  ‖ gen("return")
  ‖ gen("END FUNCTION");
```

(a) 函数定义语句语义子程序

```
stat.next = newlabel;
expr.true = newlabel;
expr.false = stat.next;
stats.next = stat.next;
stat.code = expr.code
      ‖ gen(expr.true)
      ‖ stats.code
      ‖ gen(stat.next ":");
```

(b) If语句语义子程序

```
stat.next = newlabel;
expr.true = newlabel;
expr.false = newlabel;
stats1.next = stat.next;
stats2.next = stat.next;
stat.code = expr.code
  ‖ gen(expr.true ":")
  ‖ stats1.code
  ‖ gen("goto" stats1.next)
  ‖ gen(expr.false)
  ‖ stats2.code
  ‖ gen(stat.next ":");
```

(c) Else语句语义子程序

```
stat.begin = newlabel;
stat.next = newlabel;
stats.begin = newlabel;
expr.begin = newlabel;
expr.true = stats1.begin;
expr.false = stat.next;
stat.code =
    gen("goto" expr.begin)
  ‖ gen(stats.begin)
  ‖ stats.code
  ‖ gen(expr.begin)
  ‖ expr.code
  ‖ gen(stat.next) ;
```

(d) While语句语义子程序

图2　语法制导定义

Fig. 2　Syntax-directed definitions

布尔表达式的语法制导翻译规则与编译原理[14]317中布尔表达式制导翻译规则类似,不做详细描述。图2(b)~(d)分别列出了语法规则(2)、(3)、(5)对应的语法制导翻译规则。根据图2语法制导翻译规则,可将伪代码翻译为中间代码,如图3(a)中伪代码翻译为图3(b)中的中间代码。

4)中间代码优化和基本块的划分。

如图3(b)中语句L7,语法制导翻译生成的中间代码存在空语句,需对中间代码进行优化。使用ANTLR生成中间代码对应的语法分析树,遍历语法分析树并使用算法1对其优化,优化结果如图3(c)。

算法 1　中间代码优化算法。

输入　中间代码语法分析树 mTree；

输出　中间代码优化结果。

```
1) For node in mTree
2)   If empty(node)                                    //空语句节点
3)     newNode.begin = node.begin
4)     newNode.next =node.next
5)     emptyList.add(newNode)
                                          //将空语句信息保存到列表
6)   End If
7) End For
8) For emptyNode in emptyList
9)   For node in mTree                    //遍历语法分析树中每个结
10)    If jumpNode(node)                          //跳转语句节点
11)      If node.goto == emptyNode.begin
                  //跳转语句的目标语句标号与空语句标号相等
12)        node.goto = emptyNode.next
13)      End If
14)    End If
15)  End For
16)  delet(emptyNode)            //删除语法分析树中空语句节点
17)End For
18)out(mTree)                          //输出优化后语法分析树内容
```

为了更好地分析控制流，将中间代码划分为基本块。基本块是满足下列条件的最大的连续中间表示语句序列[15]：

a)控制流只能从基本块的第一个语句进入该块。即没有跳转到基本块中间的转移指令。

b)除了基本块的最后一个语句，控制流在离开基本块之前不会停止或跳转。

因此，每一个基本块对应一个入口语句，只需确定入口语句即可划分基本块，选择入口语句规则如下：

a)中间代码的第一个语句；

b)能由转移语句转移到的语句；

c)紧跟在一个转移语句之后的语句。

每个入口语句对应的基本块包括了从它自己开始，直到下一个入口语句或结尾语句之间的所有语句。如图 4 所示，对图 3(c)中优化结果的基本块划分，其中 bb 代表基本块。

```
FUNCTION test()
  WHILE x < y DO
    x<- x+1
    IF x < 1 THEN
      x<- y
    END IF
  END WHILE
  x <- x+y
END FUNCTION
```

(a) 伪代码

```
FUNCTION test()
L1:
  goto L4
L3:
  x<- x+1
  if x <1  goto L6
  else goto L7
L6:
  x<- y
L7:
L4:
  if x <y goto L3
  else goto L5
L5:
  x<- x+y
L2:
  return
END FUNCTION
```

(b) 中间代码

```
FUNCTION test()
L1:
  goto L4
L3:
  x<- x+1
  if x <1  goto L6
  else goto L4
L6:
  x<- y
L4:
  if x <y goto L3
  else goto L5
L5:
  x<- x+y
L2:
  return
END FUNCTION
```

(c) 优化结果

图 3　伪代码翻译及优化结果

Fig. 3　Pseudocode translation and optimization results

```
FUNCTION test()
bb1:
  goto bb4
bb3:
  x<- x+1
  if x<1  goto bb6
  else goto bb4
bb6:
  x<- y
bb4:
  if x<y  goto bb3
  else goto bb5
bb5:
  x<- x+y
bb2:
  return
END FUNCTION
```

图 4　基本块划分结果

Fig. 4　Result of basic block partitioning

2.1.2　源代码中间表示的生成

GNU 编译器套件(GNU Compiler Collection，GCC)可将源代码转换为中间表示。通过 GCC 调试选项-fdump-tree-cfg 并输入源代码，即可得到 GCC 编译器生成的 cfg 中间文件。文献[1]中使用该方法提取源代码函数结构特征。cfg 文件中的中间代码是 GCC 编译器对源代码进行词法和语法分析后得到的中间表示，其中包含划分基本块后的结果以及基本块的执行顺序。如图 5(a)中源代码以及通过 GCC 编译器生成图 5(b)中 cfg 中间码。

2.2　函数特征提取

相关定义如下：

定义 1　函数调用关系图可以表示为一种有向图 $G=\langle V, E\rangle$。其中：V 是节点集，每一个节点代表一个函数；$E=\{(x, y) \mid x, y\in V\}$ 是弧集，表示函数 x 调用了函数 y。

定义 2　控制流图是一种简单的控制流表示方法，描述逻辑控制流。控制流图 G 是一种有向图 $G=\langle B, D\rangle$。其中：B 为节点集合，每一个节点代表一条或多条语句，即一个基本块；$D=\{(x, y) \mid x, y\in \mathbf{N}_+\}$ 是弧集，表示控制流向。

定义 3　基本块序列(Basic Block Sequence，BBS)为一个有序序列 $bbs=b_1, b_2, \cdots, b_i, b_j\in\{seq, con\_jump, uncon\_jump, seq\_cj, seq\_nj, return\}$。其中：$b_j$ 表示第 j 个基本块；*seq* 为只包含顺序语句的基本块；*uncon_jump* 为只包含无条件转移语句的基本块；*con_jump* 为只包含条件转移语句的基本块；*seq_cj* 为包含顺序语句和条件转移语句的基本块；*seq_nj* 表示包含顺序语句和无条件转移语句的基本块；*return* 表示包含 return 的基本块。

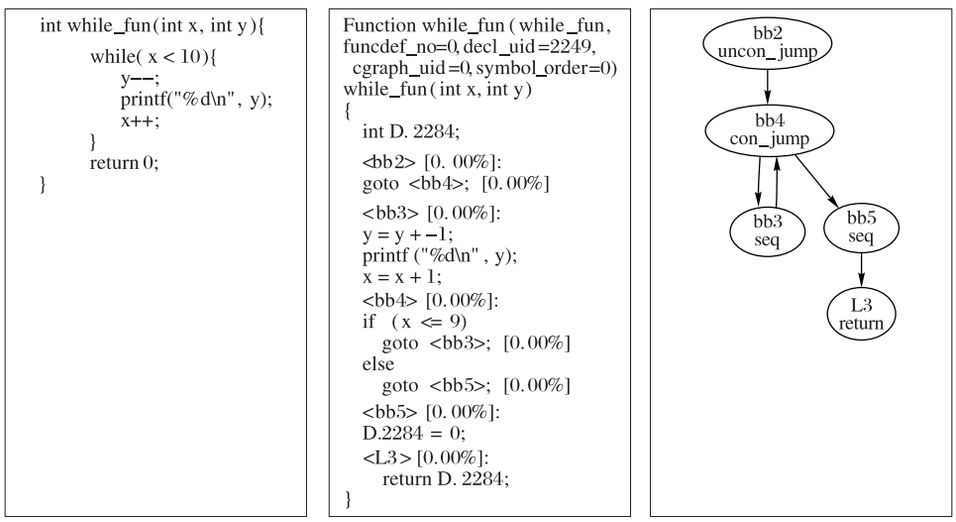
1)函数调用关系的提取。

伪代码函数调用关系的获取。在遍历语法分析树的过程中，匹配 functiondef 规则可获取主调函数名称及参数信息，匹配 stat 规则中函数调用规则“id '(' args? ')'”可获取被调函数名称及参数信息。functiondef 规则结束则完成主调函数调用信息的提取，遍历整个语法分析树，即可获取全部函数调用信息并生成函数调用关系图。

源代码函数调用关系的获取。使用 ANTLR，将 cfg 文件输入，生成语法分析树。遍历语法分析树，匹配函数定义和函数调用语句，获取函数调用关系。图 6 所示为源码以及根据提取的函数调用关系生成的函数调用关系图。

2)控制流信息的提取。

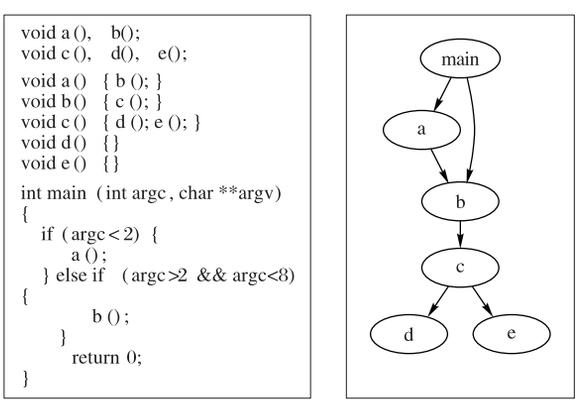
控制流信息包括函数内部的控制流图和基本块序列。在遍历语法分析树的过程中,根据基本块中是否包含条件转移语句、无条件转移语句以及return语句,判定基本块,获取基本块序列。同时匹配转移语句中的"goto bbi",获取基本块流向,得到控制流图。如图5(c)中控制流图,图中节点为基本块,其对应的基本块序列 *bbs=uncon_jump*, *seq*, *con_jump*, *seq*, *return*。



(a) 源代码　　(b) cfg中间代码　　(c) 控制流图

图5　中间代码和控制流

Fig. 5　Intermediate code and control flow



(a) 源代码　　(b) 函数调用关系

图6　源代码及其对应的函数调用关系

Fig. 6　Source code and corresponding function call relationship

3　特征模型的建立对比

3.1　特征模型的建立

设计文档中的伪代码描述了每一模块的实现方式,包括实现算法、逻辑流程等,通过分析设计文档中伪代码,获取设计函数间调用关系以及设计函数控制流,建立设计特征模型。通过静态分析源代码获取实现函数调用关系以及每个函数的控制流,建立软件实现特征模型。由于图能有效表示对象的属性以及对象之间的关系[16],因此采用图来表示函数调用关系和函数控制流信息。

根据函数特征中的函数间调用关系和函数控制流信息,分别建立函数调用关系图和控制流图。以函数调用关系图为载体,将图中每个节点映射到对应函数的基本块序列和控制流图,从而建立设计特征模型和软件实现特征模型。

3.2　特征模型对比

对比软件设计与实现的函数调用关系图可以验证实际系统是否按照设计要求的系统结构实现。对比软件设计与实现函数的控制流图可以检查每个函数否是按指定的逻辑流程实现。因此,特征模型的对比包括函数调用关系图和函数控制流图的对比。

特征模型对比过程:首先进行外部特征(函数调用关系图)的对比,通过计算函数调用关系图的相似度来验证外部特征的一致性。如果外部特征(函数调用关系图)不一致,说明实际系统没有按照设计要求的系统结构进行实现,直接给出系统结构不一致的检测结果。如果外部特征(函数调用关系图)一致,则获取函数调用关系图中函数的匹配状态,并对匹配函数进行内部特征(控制流图)的对比。通过计算控制流图的相似度验证内部特征的一致性,过程中可获取独立路径的匹配状况;然后逐一比对已匹配的独立路径,对路径中每一个节点(基本块)类型进行匹配,并标记类型不匹配的基本块;最终获取类型不匹配的基本块标号,作为一致性检测结果。

3.3　相似度的计算

定义4　图 G 使用二元组表示,即 $G=(V,E)$。其中:V 为图 G 中的所有节点的集合;$E \subseteq V \times V$ 表示边的集合。

图相似度的计算基于图编辑距离,采用图匹配算法领域中常用方法——二分图编辑距离,将图编辑距离问题转化为二分图匹配问题[17]。对源图 $G_1(V_1,E_1)$ 和目标图 $G_2(V_2,E_2)$ 构造完全二分图 G,图 G 中边的权值为图中节点的相似度,节点相似度越大,权值越大。因此,图 G_1、G_2 的相似度问题转变为带权二分图最优匹配问题,采用Kuhn-Munkers算法即可求出最大权匹配。Kuhn-Munkers算法是一种二分图最佳匹配算法,文献[18-19]使用Kuhn-Munkers算法分析恶意代码函数调用图的相似度。

3.3.1　函数调用图相似度计算

函数调用图 g_1 和 g_2 所构成的二分图 G 的边权值通过对应

函数的相似度和函数调用关系相似度联合计算得到。下文中v_1和v_2分别表示图g_1和g_2中的函数。

1)函数相似度计算。

函数v_1和函数v_2的相似度通过计算函数内基本块序列bbs的相似度得到。代码块序列的相似度度量方法有多种,如最长公共子序列、后缀数组[20]、哈希比较法[21]、机器学习[22]等。在v_1和v_2的相似度度量中,bbs长度一致时,更能说明二者的相似性,即函数v_1和v_2对应的基本块序列bbs_1和bbs_2长度是否相同,对应的权重是不同的。而最长公共子序列,后缀数组的方法,只能说明bbs_1是否包含bbs_2,精确度较低。哈希比较法又过于敏感,容易造成误判。机器学习的方法需要大量的模板集,难以一般化。

本文采用莱温斯坦距离(Levenshtein Distance, ld),莱温斯坦距离是一种度量方法,通常用来比较基于字符序列的两个字符串[23]。字符序列定义了字符串的结构[23],基本块序列类似于字符串序列,可以使用该方法计算其相似度。由定义3基本块有6种类型,用1~6表示,则图4中函数基本块序列可表示为bbs = 3, 4, 1, 2, 1, 6。函数v_1和v_2的相似度计算公式如下:

$$SimFun(v_1, v_2) = 1 - \frac{ld(bbs_1, bbs_2)}{|bbs_1| + |bbs_2|}$$

其中:bbs_1、bbs_2为函数v_1、v_2对应的基本块序列;$ld(bbs_1, bbs_2)$为基本块序列bbs_1和bbs_2的莱温斯坦距离;$|bbs_1|$、$|bbs_2|$为两个序列的基本块个数。当基本块序列得相似性达到一定的阈值,就认为两个函数是相似的。大量实验结果表明,基本块序列的相似度达到80%,则可判定两个函数是相似的。

2)函数调用关系相似度计算。

函数调用关系相似度计算公式如下:

$$SimFC(v_1, v_2) = \frac{n_1 + n_2}{N}$$

其中:n_1为函数v_1和v_2的主调函数的相似对个数;n_2为函数v_1和v_2的被调函数的相似对个数;N为函数v_1、v_2的主调函数和被调函数组成的所有函数对个数。

3)二分图边权值计算。

二分图边权值计算公式如下:

$$w(v_1, v_2) = \frac{SimFun(v_1, v_2) + SimFC(v_1, v_2)}{2}$$

函数调用图g_1和g_2对应的二分图构造完成后,采用KM(Kuhn-Munkers)算法求其最大权匹配M,匹配M的边权值占比为函数调用图g_1和g_2的相似度,权值越大,相似度越大。图相似度计算公式如下:

$$GSim(g_1, g_2) = \frac{w(M)}{N}$$

其中:$w(M)$为最大权匹配M的权值;N为图g_1和g_2中函数构成的函数对个数。

3.3.2 控制流图相似度计算

控制流cg_1和cg_2构成二分图G_{cg},二分图G_{cg}的顶点为控制流图cg_1和cg_2中的独立路径,图G_{cg}的边权值为图cg_1、cg_2中各独立路径之间的相似度。

1)独立路径的相似度计算。

控制流图中的独立路径为基本块序列,相似度计算同样采用莱温斯坦距离。计算公式如下:

$$SimPath(p_1, p_2) = 1 - \frac{ld(pbbs_1, pbbs_2)}{|pbbs_1| + |pbbs_2|}$$

其中:$pbbs_1$、$pbbs_2$为独立路径p_1与p_2对应的基本块序列;$ld(pbbs_1, pbbs_2)$为基本块序列$pbbs_1$与$pbbs_2$的莱温斯坦距离;$|pbbs_1|$、$|pbbs_2|$为两个序列的基本块个数。

2)控制流图相似度计算。

采用KM算法获取二分图G_{cg}的最大权匹配M,并根据匹配M计算控制流图的相似度。计算公式如下:

$$SimCG(cg_1, cg_2) = \frac{w(M)}{PN}$$

其中:$w(M)$为最大权匹配M的权值;PN为图cg_1和cg_2中独立路径的对数。

4 实验与分析

实验采用数据结构中常规排序算法和查找算法,源代码下载自GitHub(https://github.com/TheAlgorithms/C)。伪代码则根据各排序算法和查找算法的算法思路以及伪代码语法规则人工编写完成。由于函数间调用关系是否一致会导致实验结果的不同,因此实验一验证不同函数调用关系情况下的一致性检测结果。由于本文一致性检测方法依赖于设计文档伪代码,因此实验二验证在设计文档伪代码功能描述不完善情况下的一致性检测结果。使用本文方法,分析功能完全的设计文档伪代码获取函数特征并建立设计特征模型。如图7所示。

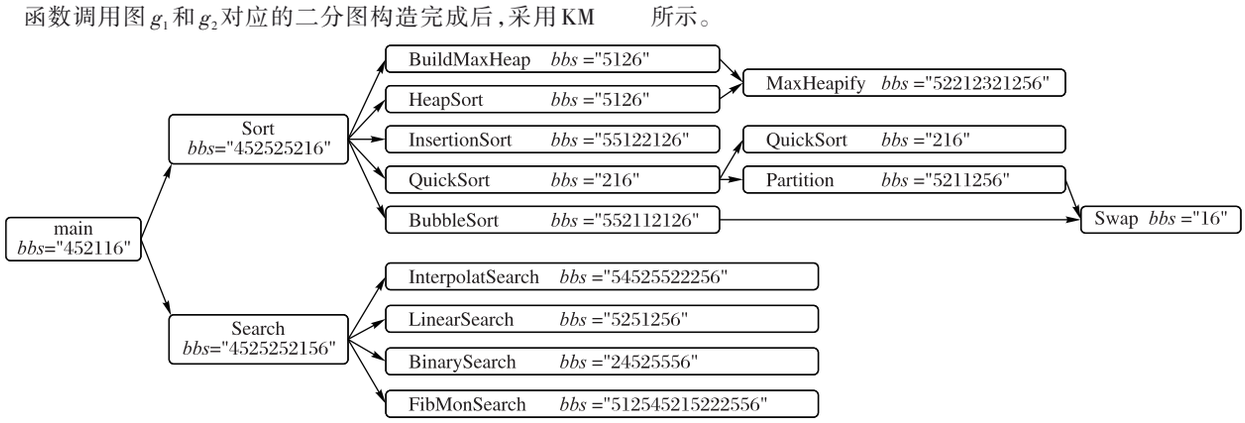


图7 设计特征模型

Fig. 7 Design feature model

4.1　实验一

软件设计与软件实现之间可能存在以下几种情况:

1)实现系统完全按照设计要求实现了所有函数,实现系统函数调用关系满足设计要求,需要进行函数调用关系图和控制路图的对比,如实验Code1。

2)实现系统按照设计要求实现了所有函数,但存在部分函数调用关系与设计要求不一致,如实验Code2中函数Search和Sort未调用任何函数。

3)实现系统实现系统未实现所有函数,但其函数调用关系与设计要求部分一致,如实验Code3只实现了排序功能。

分析被测代码,获取实际函数特征模型,与设计特征模型进行对比。表2显示了实现与设计系统中各函数匹配结果以及函数调用关系相似度*simFC*。*simFC*>0.8表示匹配成功,且函数基本块序列和函数调用关系非常相似;0<*simFC*<0.8表示匹配到函数但函数基本块序列或者函数间调用关系一致度较低;*simFC*=0表示未匹配到对应函数。Code1为与设计特征模型中函数调用关系一致的代码;Code2为与设计特征模型中函数调用关系部分一致的代码,其中函数Sort和Search未调用任何函数;Code3为只实现了查找功能的代码。

表2　函数调用关系图匹配结果

Tab. 2　Function call relationship matching results

函数名	Code1	Code2	Code3
Sort	1.00	0.64	0.00
BuildMaxHeap	0.98	0.52	0.00
HeapSort	0.98	0.54	0.00
InsertionSort	0.96	0.51	0.00
QuickSort	1.0	0.90	0.00
BubbleSort	0.91	0.75	0.00
MaxHeapify	0.88	0.88	0.00
Partition	0.96	0.96	0.00
Swap	0.98	0.98	0.00
Search	0.94	0.61	0.94
InterpolSearch	0.90	0.41	0.90
LinearSearch	0.92	0.46	0.92
BinarySearch	0.93	0.45	0.93
FibMonSearch	0.93	0.44	0.93

表2所示Code1实验结果中,各函数调用关系相似度均大于0.8,表明Code1的函数调用关系与设计要求一致。可以得出,在设计和实现的函数调用关系一致的情况下,检测结果与预期结果较为一致,说明在设计特征模型和实际特征模型一致时,本文方法能够正确匹配对应函数。

Code2实验结果中,每个函数都得到匹配结果,而函数Sort和Search的函数调用关系相似度均低于0.8,表明Code2虽然实现了所有函数,但函数Sort和Search的函数调用关系与设计要求不一致。说明在设计和实现的函数调用关系部分不一致的情况下,本文方法可以检测到函数调用关系不一致的部分。另外,其他函数的函数调用关系相似度也低于0.8,如所有查找函数和部分排序函数,说明检测结果虽然与预期方向一致,但部分结果粗糙。这是由于Code2中Sort和Search函数未调用任何函数,使相关的排序算法和查找算法的被调用关系受到影响。导致即使实现了设计要求的函数,比如函数InterpolatSearch、LinearSearch、BinarySearch等,却由于被调用关系的不一致,仍被视为匹配度较低,而函数maxHeapify、partition、Swap未受到影响,检测结果仍然大于0.8。但是实验检测结果仍能说明实现代码Code2的函数调用关系与设计要求不一致。

Code3实验结果中,排序相关函数的匹配结果均为0,表明Code3没有实现排序功能。查找相关的函数匹配结果均大于0.8,表明Code3完成了查找功能。实验结果与预期一致,说明在实现代码只完成部分功能时,仍可以正确匹配实现代码中功能完成部分的函数,如Code3中实现查找功能的函数。对于未完成部分函数并不能匹配到对应函数,如排序功能相关的函数。同时,检测结果能够说明实现代码的函数调用关系与设计要求不一致。

由于Code2和Code3的函数调用关系与设计要求不一致,不进行函数内部控制流的验证。Code1函数调用关系与设计要求一致,则根据函数对应关系,对函数内部控制流进行对比。表3列出了设计函数与实现函数控制流图的相似度以及控制流图中基本块不一致检测结果,基本块的标号为实际函数中基本块标号。实验结果中除函数bubbleSort外,控制流图相似度均大于0.8,说明这些函数实现逻辑与设计要求一致。实验14个函数中,仅有函数bubbleSort与预期不符,控制流图匹配准确度达到92.8%,总体实验结果与预期较为一致。另外,Code1中函数bubbleSort的实现逻辑虽与设计要求一致,但由于控制流图中3个不同类型基本块在所有独立路径中,导致控制流图相似度较低,但可通过基本块检测结果进行精确验证。因此实际检测中应以控制流图相似度作为参考,并根据基本块检测结果进行精确验证。

表3　对应函数控制流一致性检测结果

Tab. 3　Consistency detection results of corresponding function control flow

设计函数名称	Code1函数名称	相似度	基本块标号
Sort	sort	0.91	9
BuildMaxHeap	buildMaxHeap	1.00	无
HeapSort	heapSort	1.00	无
InsertionSort	insertionSort	0.96	6
QuickSort	quickSort	1.00	无
BubbleSort	bubbleSort	0.76	4,7,9
MaxHeapify	maxHeapify	0.84	4,6,8,11
Partition	partition	0.90	3,7
Swap	swap	1.00	无
Search	search	0.89	9
InterpolSearch	interpolSearch	0.81	5,9,10
LinearSearch	linearSearch	0.89	3,7
BinarySearch	binarySearch	0.90	5,8
FibMonSearch	fibMonSearch	0.87	8,9,13,15

4.2　实验二

设计文档伪代码中函数Searh不调用任何函数,建立设计特征模型,与Code1进行一致性检测,Code1为按设计要求完成的代码。实验结果如图8所示,说明在设计文档伪代码不准确时会导致误判,如函数InterpolatSearch、LinearSearch、BinarySearch、FibMoncianSearch。由于是以设计文档伪代码为根据,建立设计特征模型,并以此作为检测依据,因此在设计文档伪代码部分不准确时,会造成不准确部分函数调用关系的误判,但并不影响其余准确部分功能的检测。另外,后续的函数内部结构的检测可以对该问题进行弥补,所以在实际软件系统一致性检测中,应保证设计文档伪代码的质量,设计文档伪代码质量越高,一致性检测越准确。

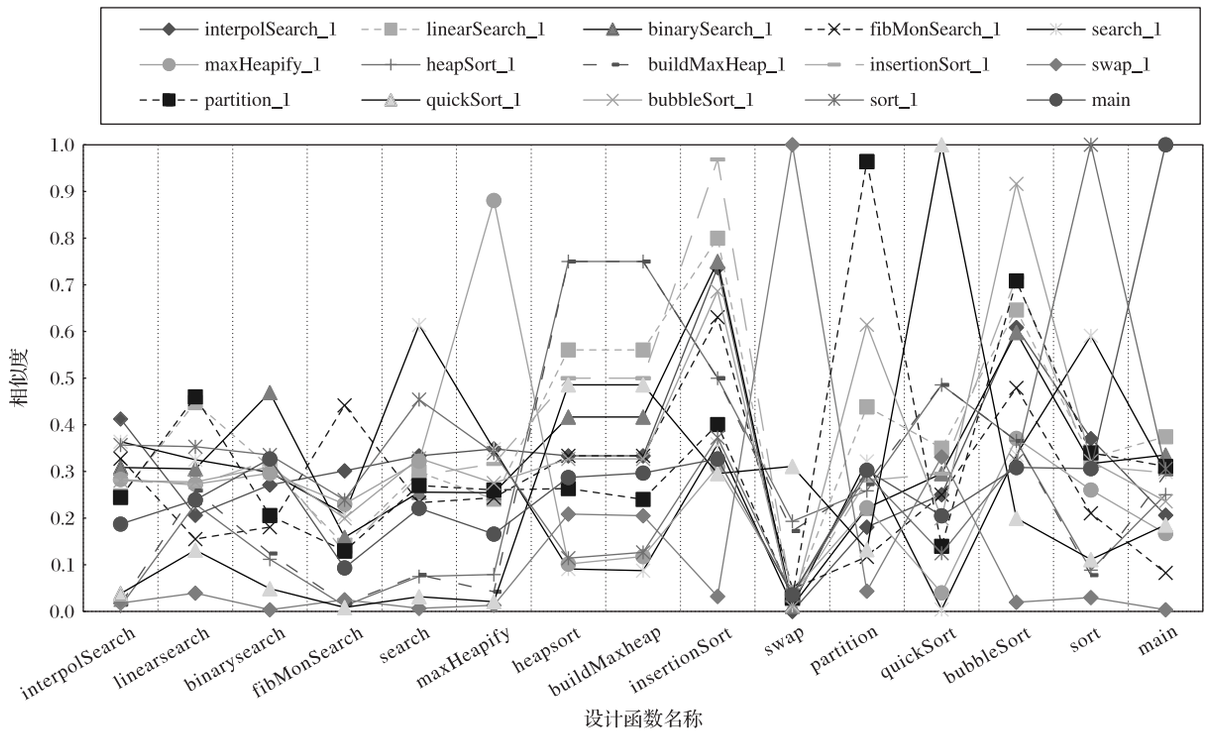


图8 设计系统与实际系统函数相似度

Fig. 8 Functional similarity between design system and actual system

5 结语

软件实现与设计一致性验证是软件测试的重要部分之一。本文提出了一种面向设计文档伪代码的设计与实现一致性检测方法，为基于文档的测试提供一种新的研究思路。从设计文档伪代码和程序源代码自动提取函数特征，建立特征模型，并利用二分图最大权匹配算法验证特征模型相似度，获取函数对应关系，完成函数内部结构的对比，进而完成软件设计与实现的一致性检测。该方法不需要大量的模板集，即可获取一致性检测结果，提升了一致性验证的工作效率，并具有更优越的一般性。

但是该方法还存在以下不足：1)在对函数内部结构验证过程中，并没有对条件转移语句的条件进行验证；2)对基本块验证的过程中，没有考虑其对数据的依赖关系。后续工作可以充分研究基本块内部结构的验证，提高验证的准确度。

参考文献（References）

[1] 牟永敏，杨志嘉. 基于函数调用路径的软件实现与设计一致性验证[J]. 中国科学：信息科学，2014，44(10)：1290-1304.（MU Y M，YANG Z J. Verify consistency of software implementation and design based on function call path [J]. SCIENTIA SINICA Informationis，2014，44(10)：1290-1304.）

[2] RIEDL-EHRENLEITNER M，DEMUTH A，EGYED A. Towards model-and-code consistency checking [C]// Proceedings of the IEEE 38th Annual Computer Software and Applications Conference. Piscataway：IEEE，2014：85-90.

[3] 曾一，李函逾，刘慧君，等. UML模型和Java代码之间的一致性检测方法[J]. 计算机科学，2015，42(4)：151-155.（ZENG Y，LI H Y，LIU H J，et al. Consistency detection method between UML model and Java source code[J]. Computer Science，2015，42(4)：151-155.）

[4] 余双双，曾一，刘慧君，等. 基于UML模型的多态性与Java接口代码信息一致性检测的方法[J]. 计算机应用与软件，2017，34(2)：8-13，47.（YU S S，ZENG Y，LIU H J，et al. The consistency detection method of polymorphism and Java interface code information based on UML model[J]. Computer Applications and Software，2017，34(2)：8-13，47.）

[5] ZHENG Y，MU Y，ZHANG Z. Research on the static function call path generating automatically [C]// Proceedings of the 2nd IEEE International Conference on Information Management and Engineering. Piscataway：IEEE，2010：405-409.

[6] 张海藩，牟永敏. 软件工程导论[M]. 6版. 北京：清华大学出版社，2013：94-100.（ZHANG H F，MU Y M. Introduction to Software Engineering[M]. 6th ed. Beijing：Tsinghua University Press，2013：94-100.）

[7] 徐参语. 基于静态检测的克隆代码检测工具的设计与实现[D]. 成都：电子科技大学，2019：1-82.（XU C Y. Design and implementation of clone code detecting tool based on static detecting [D]. Chengdu：University of Electronic Science and Technology of China，2019：1-82.）

[8] 李锁，吴毅坚，赵文耘. 基于代码克隆检测的代码来源分析方法[J]. 计算机应用与软件，2020，37(2)：8-14.（LI S，WU Y J，ZHAO W Y. Code provenance analysis based on code clone detection[J]. Computer Applications and Software，2020，37(2)：8-14.）

[9] WEI H，LI M. Supervised deep features for software functional clone detection by exploiting lexical and syntactical information in source code [C]// Proceedings of the 26th International Joint Conferences on Artificial Intelligence. Palo Alto，CA：AAAI Press，2017：3034-3040.

[10] KAMALPRIYA C M，SINGH P. Enhancing program dependency graph based clone detection using approximate subgraph matching

[C]// Proceedings of the IEEE 11th International Workshop on Software Clones. Piscataway: IEEE, 2017: 1-7.

[11] SUDHAMANI M, RANGARAJAN L. Code similarity detection through control statement and program features [J]. Expert Systems with Applications, 2019, 132: 63-75.

[12] 陈秋远,李善平,鄢萌,等. 代码克隆检测研究进展[J]. 软件学报, 2019, 30(4): 962-980. (CHEN Q Y, LI S P, YAN M, et al. Code clone detection: a literature review[J]. Journal of Software, 2019, 30(4): 962-980.)

[13] 吕云翔,杨颖,朱涛,等. 软件测试实用教程[M]. 北京:清华大学出版社, 2014: 1-427. (LYU Y X, YANG Y, ZHU T, et al. Practical Tutorial on Software Testing [M]. Beijing: Tsinghua University Press, 2014: 1-427.)

[14] AHO A V, LAN M S, SETHI R, et al. 编译原理[M]. 2版. 赵建华,郑滔,戴新宇,译. 北京:机械工业出版社, 2009: 1-524. (AHO A V, LAN M S, SETHI R, et al. Compilers: Principles, Techniques and Tools[M]. 2nd ed. ZHAO J H, ZHENG T, DAI X Y, translated. Beijing: China Machine Press, 2009: 1-524.)

[15] 张晔,陆余良. PLC程序控制流分析方法[J]. 计算机应用, 2017, 37(12): 3581-3585. (ZHANG Y, LU Y L. Control flow analysis method of PLC program [J]. Journal of Computer Applications, 2017, 37(12): 3581-3585.)

[16] FOGGIA P, PERCANNELLA G, VENTO M. Graph matching and learning in pattern recognition in the last 10 years [J]. International Journal of Pattern Recognition and Artificial Intelligence, 2014, 28(1): No. 1450001.

[17] 徐周波,张鹃,宁黎华,等. 图编辑距离概述[J]. 计算机科学, 2018, 45(4): 11-18. (XU Z B, ZHANG K, NING L H, et al. Summary of graph edit distance[J]. Computer Science, 2018, 45(4): 11-18.)

[18] 江志雄. 基于函数调用图比对的恶意代码相似性分析技术研究与实现[D]. 长沙:国防科学技术大学, 2015: 1-76. (JIANG Z X. Research and implementation of malware similarity analysis technology based on function-call graph comparison [D]. Changsha: National University of Defense Technology, 2015: 1-76.)

[19] 刘星,唐勇. 恶意代码的函数调用图相似性分析[J]. 计算机工程与科学, 2014, 36(3): 481-486. (LIU X, TANG Y. Similarity analysis of malware's function-call graphs[J]. Computer Engineering and Science, 2014, 363): 481-486.

[20] MURAKAMI H, HOTTA K, HIGO Y, et al. Folding repeated instructions for improving token-based code clone detection [C]// Proceedings of the IEEE 12th International Working Conference on Source Code Analysis and Manipulation. Piscataway: IEEE, 2012: 64-73.

[21] WANG P, SVAJLENKO J, WU Y, et al. CCAligner: a token based large-gap clone detector [C]// Proceedings of the 2018 IEEE/ACM International Conference on Software Engineering. Piscataway: IEEE, 2018: 1066-1077.

[22] DING Y, ZHU S. Malware detection based on deep learning algorithm[J]. Neural Computing and Applications, 2019, 31(2): 461-472.

[23] BEHARA K N S, BHASKARA A, CHUNGB E. A novel approach for the structural comparison of origin-destination matrices: Levenshtein distance [J]. Transportation Research Part C: Emerging Technologies, 2020, 111: 513-530.

This work is partially supported by the Open Project of Beijing Key Laboratory of Internet Culture and Digital Dissemination Research (5221935409).

ZHANG Jiaqi, born in 1995, M. S. candidate. His research interests include software testing.

MU Yongmin, born in 1961, Ph. D., professor. His research interests include software testing, big data.

ZHANG Zhihua, born in 1971, M. S., associate professor. Her research interests include software testing, computer software.