第43卷 第6A期
2016年6月

计 算 机 科 学
Computer Science

Vol. 43 No. 6A
June 2016

面向软件错误检测的数据流分析

张广梅[1] 李景霞[2]

(山东农业大学信息科学与工程学院 泰安 271018)[1] (安徽农业大学信息与计算机学院 合肥 230036)[2]

摘 要 程序中某一点的数据流状态与软件的执行路径有关。程序中的部分错误与变量所处的状态相关。提出的MUST数据流和MAY数据流反映了数据流的执行路径具有相关性的特点。根据不同变量的管理特点,从程序的控制结构出发,详细讨论了影响变量状态的各种因素及其之间的关系,提出了基于程序控制结构的、以基本块为最小程序单位的静态数据流分析方法,为精确地进行数据流分析提供了依据。

关键词 软件错误,错误检测,数据流分析

中图法分类号 TP311 文献标识码 A

Data-flow Analysis for Software Error Detection

ZHANG Guang-mei[1] LI Jing-xia[2]

(School of Information Science and Technology, Shandong Agriculture University, Tai'an 271018, China)[1]

(School of Information and Computer Science, Anhui Agriculture University, Hefei 230036, China)[2]

Abstract Definition and reference are two kinds of operations that software variable. The operation that software variable disobeys the variable using rules will result in software error. In order to detect these kinds of software error, the definition-reach data-flow analysis and living-variable data-flow analysis of a program must be made. There may be more than one path to a program's site, and the data-flow states on one program path may be different from the others. So the must-data-flow and the may-data-flow of a program were calculated to depict the accurate data-flow information. The control structure on basic block is used by the data-flow analysis method. The factors that will affect the data-flow such as definition information that can reach to a basic block entry site and exit site, living variable that can reach to a basic block, some special operation such as memory allocate operation and memory free operation, and the relations between them were discussed sufficiently.

Keywords Software error, Software error detection, Data-flow analysis

1 引言

程序中不可避免地存在着缺陷。按照软件失效机理,程序中的缺陷是导致软件故障的根本原因[1]。在众多的软件故障中,部分软件故障与程序中变量的状态[2,3]有关,要准确地检测这类故障需要对程序进行数据流分析。

应用程序中建立在变量上的操作有两种:变量的定值(将一个值赋给一个变量)操作和引用(使用变量的值进行一些运算)操作。数据流分析是确定变量定值与引用关系的工具[1,3]。数据流分析技术在程序切片、程序的可测性分析等方面得到了广泛的应用。常见的数据流分析方法有很多,如流敏感/流不敏感的分析方法、上下文敏感/上下文不敏感的分析方法[3-5]等静态分析方法。这些分析方法从不同的角度反映了程序的数据流状态,但是将这些方法应用在软件测试中存在着数据流分析结果不精确、没有考虑指针变量的动态内存空间的分配与回收操作等问题。

为检测数据流相关的软件故障,本文提出面向软件错误检测的数据流分析方法。该方法从变量的使用特点出发,对程序进行静态分析,仿真系统动态执行过程中变量的定值与引用操作对数据流的影响,从而可以准确描述程序中的数据流状态,为软件错误的诊断提供帮助。

2 数据流分析基础

应用程序中部分软件故障与变量在引用之前是否被定值(如果变量在引用之前没有被定值,程序中存在着变量的未定值引用的错误)或被定值的变量是否被引用有关(如果变量被定值之后未被引用,则程序中存在着变量无用定值的错误)。为检测程序中存在的这些错误,需要对程序进行到达定值数据流分析和活跃变量数据流分析[5-7]。到达定值数据流分析是确定到达变量引用点的所有定值操作的数据流分析技术;活跃变量数据流分析是用来确定被定值的变量在从定值点出发的程序路径上是否被引用的数据流分析技术,这两种数据流分析技术对软件故障检测给予了很好的支持。

2.1 一组概念

为讨论的方便,先介绍一组概念[8,9]。

定义1(基本块) 基本块是划分应用程序的一个单位,

张广梅(1972—),女,博士,副教授,主要研究方向为软件测试,E-mail:lotus@sdau.edu.cn;李景霞(1976—),女,博士,副教授,主要研究方向为模型分析与验证。

是一个按照顺序执行的语句序列。每个基本块只有一个入口语句和一个出口语句。其中,入口语句是符合下列特点的语句:

1)程序的第一条语句;

2)条件转移语句或无条件转移语句的转移目标语句;

3)紧跟在条件转移语句之后的语句。

定义2(变量的定值与定值点) 在一个应用程序中变量在赋值号左边出现时称变量被定值,赋值语句所在的位置(程序中语句行)称为变量的定值点。为讨论方便,用一个二元组(p,l)表示一个定值点,其中,p表示被定值的变量,l表示定值点所在的基本块的编号。

定义3(变量的引用与引用点) 变量在赋值号右边出现时称变量被引用。应用程序中变量被引用的位置称为引用点。

2.2 到达_定值数据流分析

2.2.1 影响到达定值数据流分析的因素

到达_定值数据流是指到达程序中某一个定值点的所有定值操作的集合[7]。要进行到达_定值数据流分析,首先要确定每个基本块内的定值操作,除此之外,一个基本块内变量的定值操作使得其他基本块中对该变量的定值操作无效,因此对每个基本块来讲,需要考虑以下集合:

$GEN[B]$:基本块B内的定值点的集合。

$KILL[B]$:表示在基本块B外定值(在其它的基本块中定值)而在B中又被重新定值,新的定值使得之前的定值操作无效,这些无效的定值点组成了$KILL[B]$集合。

各集合的计算方式如下。

$GEN[B]$的计算:按照该集合的定义,一个基本块的$GEN[B]$的集合可以直接从该基本块中获得;

$KILL[B]$的计算:对于$GEN[B]$中的每一个定值点,如果该定值点所定义的变量v在其他的基本块$(C,D,F\cdots)$中又被重新定值,则基本块B中对变量v的定值将"注销"掉其块中对变量v的定值,因此,其它基本块中关于变量v的定值点属于$KILL[B]$。如果$v_1,v_2,\cdots,v_n$是在基本块B内被定值的变量,All_Define_Sites表示程序中关于$v_1,v_2,\cdots,v_n$的所有定值点的集合,则$KILL[B]$可以形式化地描述为:

$$KILL[B]=All\_Define\_Sites-GEN[B]$$

在应用程序中,被定值的变量可以沿着程序的控制结构传播,对程序的每一个基本块来讲,都有一个到达基本块入口和到达基本块出口的定值操作集合;由于程序控制结构的复杂性,到达基本块入口的定值信息中,有些是与程序的执行路径相关的,因此将到达程序中任意一点的数据流分为$MUST$数据流和MAY数据流,为表示这两类数据流,进行到达定值数据流分析时需要考虑以下4个集合。

$MUST\_IN[B]$:肯定能到达基本块入口的定值操作集合。

$MAY\_IN[B]$:可能到达基本块入口的定值操作集合。

$MUST\_OUT[B]$:肯定能到达基本块出口的定值操作集合。

$MAY\_OUT[B]$:可能到达基本块出口的定值操作集合。

2.2.2 集合间关系

从图1所示的控制流图可以看到,对于一个基本块B的所有前驱节点$b_i(i=1\cdots n)$来讲$(b_i\in proc(B))$,当且仅当一个定值操作属于所有的$MUST_OUT[b_i]$,该定值信息肯定能到达B的入口,由此可以得到式(1):

$$MUST\_IN[B]=\bigcap_{b\in proc(B)}MUST\_OUT[b] \tag{1}$$

对于一个基本块B的所有前驱节点$b_i(i=1\cdots n)$来讲$(b_i\in proc(B))$,当一个定值操作不包含在所有的$MUST_OUT[b_i]$,或者一个定值操作属于$MAY_OUT[b_i]$时,该定值信息可能会到达基本块B的入口,由此可以得到式(2):

$$MAY\_IN[B]=\bigcup_{b\in proc(B)}MAY\_OUT[b]\cup\left(\bigcup_{b\in proc(B)}MUST\_OUT[b]-\bigcap_{b\in proc(B)}MUST\_IN[b]\right) \tag{2}$$

对于一个基本块来讲,如果一个定值操作属于$GEN[B]$,或者如果一个定值操作属于$MUST_IN[B]$且不属于$KILL[B]$,则被定值的变量肯定能到达基本块的出口,由此可以得到式(3):

$$MUST\_OUT[B]=(MUST\_IN[B]-KILL[B])\cup GEN[B] \tag{3}$$

对于一个基本块来讲,如果一个定值操作属于$MAY_IN[B]$且不属于$KILL[B]$和$MUST_OUT[B]$,则该定值信息可能到达基本块的出口,由此,可以得到式(4):

$$MAY\_OUT[B]=MAY\_IN[B]-KILL[B]-MUST\_OUT[B] \tag{4}$$

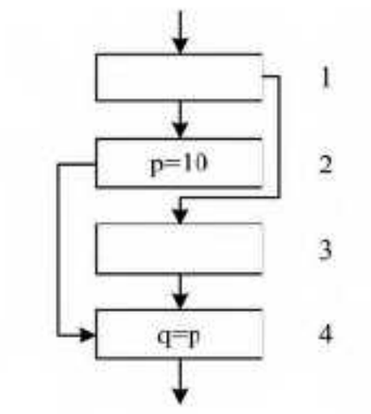

图1 一个程序控制流图(图中的节点是一个基本块)

2.3 活跃变量数据流分析

2.3.1 影响活跃变量数据流分析的因素

活跃变量数据流分析是一种考虑被定值的变量是否被引用的数据流分析方法[7]。按照活跃变量的定义,对于某一变量A在程序中p点的定值操作,如果存在一条从p出发的路径且在这条路径上有变量A的引用点,表示变量A是活跃的,因此检查程序中变量的定值未被引用的问题等价于确定被定值的变量是否活跃的问题。同到达定值数据流分析一样,由于变量的定值与引用操作分散在程序的不同的基本块之中,因此,必须从程序的控制流出发,以基本块为基本计算单元,进行活跃变量数据流的分析。因此,要进行活跃变量数据流分析,需要考虑以下内容。

$DEF[B]$:在基本块中定值,且定值之前未在B中引用过的变量集合。

$USE[B]$:在基本块中引用,但在引用之前未在B中定值的变量的集合。

为考虑基本块之间的活跃信息的传播问题,还需要考虑到达基本块入口和出口的活跃信息。由于程序路径的不唯一性,需要引入以下集合。

$MUST\_INL[B]$:肯定能到达基本块入口的活跃变量集合。

$MAY\text{-}INL[B]$：可能到达基本块入口的活跃变量集合。

$MUST\text{-}OUTL[B]$：肯定能到达基本块出口的活跃变量集合。

$MAY\text{-}OUTL[B]$：可能到达基本块出口的活跃变量集合。

2.3.2 集合间的关系

从图 1 所示的程序控制结构可以得到，只有变量 A 在基本块 B 的出口活跃，该活跃信息才能传导到 B 的后继基本块中，因此，如果一个变量 A 属于基本块 B 的所有后继 $b_i(b_i \in succ(B))$ 的 $MUST\text{-}INL[b_i]$ 集合，表示变量 A 在从基本块 B 出发的所有路径上活跃，所以该变量属于 $MUST\text{-}OUTL[B]$ 集合，由此得到式(5)：

$$MUST\text{-}OUTL[B]=\bigcap_{b\in succ(B)} MUST\text{-}INL[b] \tag{5}$$

对于程序中的活跃变量而言，只有当活跃信息可能到达基本块 B 的出口时，该活跃信息才可能到达基本块 B 的后继结点 b 中$(b\in succ(B))$。因此，如果一个活跃信息属于基本块 B 的后继 b 的 $MAY\text{-}INL[b]$，或不属于 B 的所有后继的 $MUST\text{-}INL[b]$，表示至少存在从 B 出发的一条路径，在这条路径上变量 A 不活跃，因此该活跃信息属于 $MAY\text{-}OUTL[B]$，由此得到式(6)：

$$MAY\text{-}OUTL[B]=\bigcup_{b\in succ(B)}(MAY\text{-}INL[b]\cup MUST\text{-}INL[b])-MUST\text{-}OUTL[B] \tag{6}$$

对一个基本块来讲，如果一个变量的活跃信息属于 $MUST\text{-}OUTL[B]$ 且不属于 $DEF[B]$，或者属于 $USE[B]$，则按照活跃变量的定义，这个变量的活跃信息肯定属于 $MUST\text{-}INL[B]$，由此得到式(7)：

$$MUST\text{-}INL[B]=(MUST\text{-}OUTL[B]-DEF[B])\cup USE[B] \tag{7}$$

对一个基本块来讲，如果一个变量属于 $MAY\text{-}OUTL[B]$ 且不属于 $DEF[B]$，则该变量属于 $MAY\text{-}INL[B]$，由此得到式(8)：

$$MAY\text{-}INL[B]=(MAY\text{-}OUTL[B]-DEF[B]) \tag{8}$$

3 过程内的数据流分析

3.1 到达定值数据流分析方法

从 2.2 节的讨论可以看到，要得到程序中的到达定值数据流信息，需要根据式(1)—式(4)对到达每个基本块入口和出口的定值信息进行求解。为求得一个基本块入口的到达定值信息，根据到达定值信息传播的特点，首先要求出到达该基本前驱出口的定值信息，然后按照各集合之间的关系计算其他集合的值，因此需要从程序的第一个基本块出发进行到达定值数据流分析。算法 1 描述了到达定值数据流分析过程。

算法 1 到达定值数据流分析算法

根据程序的特点计算每个基本块的 GEN[B]，KILL[B]集合；

将到达基本块入口和出口的定值信息集合初始化为空集；

```
CHANGE=true
while(CHANGE){
    CHANGE=false;
    for(i=1;i<=N;i++){ //N 代表基本块的个数
        new-must-in[Bi]= ∩ MUST-OUT[b]   (b∈pred(B))
        if(new-must-in[B] != MUST-IN[Bi])){
            CHANGE=false; 根据式(1)—式(4)计算相应的集合
        }
    }
}
```

3.2 活跃变量数据流分析

从 2.3 节的讨论可以看到，要得到程序中的活跃变量数据流信息，需要根据式(5)—式(8)计算。为求得到达基本块出口的活跃变量信息，首先要求出到达该基本后继入口的活跃变量信息，然后按照各集合之间的关系计算其他集合的活跃值，因此需要从程序的最后一个基本块出发进行活跃变量数据流分析。算法 2 描述了活跃变量数据流分析过程。

算法 2 活跃变量数据流分析算法

根据程序的特点计算每个基本块的 USE[B]，DEF[B]集合；

将到达基本块入口和出口的活跃信息集合初始化为空集；

```
CHANGE=true;
while(CHANGE){
    CHANGE=false;
    for(i=N;i>=1;i--){ //N 代表程序中最后一个基本块(包含 return 语句)
        根据式(5)、式(6)计算到达基本块出口的活跃信息；
        根据式(7)计算 new-must-inL[Bi]；
        if(new-must-inL[Bi] != MUST-IN[Bi])){
            CHANGE=flase;根据式(8)计算 MAY-INL[Bi] }
    }
}
```

4 实例分析

本节以过程内的数据流分析为例，根据第 3 节介绍的数据流分析的算法，对过程内的各基本块的到达定值数据流进行分析。

4.1 到达各基本块的定值信息的计算

为完成到达各基本块的定值信息的计算，首先以基本块为单位，根据程序的控制结构，对程序中的基本块进行拓扑排序。根据拓扑排序算法[10]，对图 2 所示的控制流图进行分析，得到以下拓扑排序：1，2，3，4，5。表 1 列出了按照根据前面介绍的数据流分析方法得到的各基本块的 $GEN[B]$ 和 $KILL[B]$ 集合。

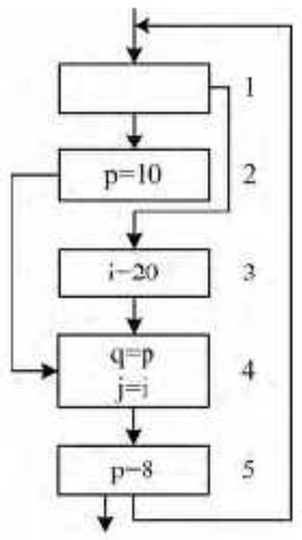

图 2 含有循环的程序控制流图

表 1 各基本块的 $GEN[B]$ 和 $KILL[B]$ 集合

基本块	该基本块的前驱	GEN[B]	KILL[B]
1	5	Ø	Ø
2	1	{(p,2)}	{(p,5)}
3	1	{(i,3)}	Ø
4	2,3	{(q,4),(j,4)}	Ø
5	4	{(p,5)}	{(p,2)}

接下来，根据算法 1 和算法 2，对图 2 所示程序进行数据

流分析。表 2—表 4 列出了到达定值数据流分析的结果。

表 2　各基本块的到达定值信息集合(第 1 次迭代结果)

基本块	MUST-IN [B]	MUST-OUT [B]	MAY-IN [B]	MAY-OUT [B]
1	{(p,5)}	{(p,5)}	Ø	Ø
2	{(p,5)}	{(p,2)}	Ø	Ø
3	{(p,5)}	{(p,5),(i,3)}	Ø	Ø
4	{(p,2),(p,5)}	{(p,2),(p,5),(q,4),(j,4)}	{(i,3)}	{(i,3)}
5	{(p,2),(p,5),(q,4),(j,4)}	{(p,5),(q,4),(j,4)}	{(i,3)}	{(i,3)}

表 3　各基本块的到达定值信息集合(第 2 次迭代结果)

基本块	MUST-IN [B]	MUST-OUT [B]	MAY-IN [B]	MAY-OUT [B]
1	{(p,5),(q,4),(j,4)}	{(p,5),(q,4),(j,4)}	{(i,3)}	{(i,3)}
2	{(p,5),(q,4),(j,4)}	{(p,2),(q,4),(j,4)}	{(i,3)}	{(i,3)}
3	{(p,5),(q,4),(j,4)}	{(p,5),(q,4),(j,4),(i,3)}	Ø	Ø
4	{(p,2),(p,5),(q,4),(j,4)}	{(p,2),(p,5),(q,4),(j,4)}	{(i,3)}	{(i,3)}
5	{(p,2),(p,5),(q,4),(j,4)}	{(p,5),(q,4),(j,4)}	{(i,3)}	{(i,3)}

表 4　各基本块的到达定值信息集合(第 3 次迭代结果)

基本块	MUST-IN [B]	MUST-OUT [B]	MAY-IN [B]	MAY-OUT [B]
1	{(p,5),(q,4),(j,4)}	{(p,5),(q,4),(j,4)}	{(i,3)}	{(i,3)}
2	{(p,5),(q,4),(j,4)}	{(p,2),(q,4),(j,4)}	{(i,3)}	{(i,3)}
3	{(p,5),(q,4),(j,4)}	{(p,5),(q,4),(j,4),(i,3)}	Ø	Ø
4	{(p,2),(p,5),(q,4),(j,4)}	{(p,2),(p,5),(q,4),(j,4)}	{(i,3)}	{(i,3)}
5	{(p,2),(p,5),(q,4),(j,4)}	{(p,5),(q,4),(j,4)}	{(i,3)}	{(i,3)}

从表 2—表 4 中可以看到,由于第三次迭代的结果与第二次相同,迭次过程停止,完成到达每个基本块入口和出口的定值信息的计算。

4.2　利用 DU(Define-Use)集合进行部分软件错误的检测

DU 集合存储的是能够到达程序中某变量的引用点的所有定值信息的集合。对于变量的某个引用点而言,如果该引用点的 DU 集合为空,则表示该变量存在未定值引用的问题。

根据第 2 节介绍的到达定值数据流分析方法可以看到,对于每一个引用点来讲,包含两个 DU 集合:$MUST-DU$ 集合和 $MAY-DU$ 集合。下面给出了 $MUST-DU$ 集合和 $MAY-DU$ 集合的计算方法。

如果在基本块 B 中,某变量 A 的引用点之前有 A 的定值点 d,且 d 为 A 所定之值能到达点 u,则 A 在 u 点的 $MUST-UD$ 集合为 $\{d\}$,$MAY-UD$ 集合为 $\emptyset$;

如果基本块 B 中,某变量 A 的引用点 u 之前没有 A 的定值点,则包含在 $MUST-IN[B]$ 中的全部 A 的定值点均可以到达 u,故 $MUST-IN[B]$ 中这些 A 的定值点组成了 A 在 u 点的 $MUST-UD$ 集合;包含在 $MAY-IN[B]$ 中的全部 A 的定值点可能到达 u,故 $MAY-IN[B]$ 中这些 A 的定值点做成了 A 在 u 点的 $MAY-UD$ 集合。

按照引用定值链的计算方法,对于图 2 所示的程序而言,该程序中的基本块 4 中的 p 的引用点的定值引用集合为:

$$MUST-DU(p)=\{(p,2),(p,5)\},MAY-DU(p)=\emptyset$$

由于在 p 的 $MUST-DU$ 集合中存在一个定值点,该定值点所在的基本块的拓扑排序位于基本块 4 之后,在该引用点存在着未赋值引用错误。

基本块 4 中的 i 的引用点的定值引用集合为:

$$MUST-DU(i)=\emptyset,MAY-DU(i)=\{(i,3)\}$$

由于该引用点的 $MUST-DU$ 集合为空而 $MAY-DU$ 非空,意味着不存在肯定能到达变量 i 的引用点的定值信息,即关于变量 i 的定值信息可能不能到达 i 的引用点,因此,该引用点可能存在着未定指引的错误。

4.3　到达各基本块的活跃信息的计算

要进行到达各基本块的变量的活跃信息的计算,首先需要根据基本块的特点,进行 $DEF[B]$ 集合和 $USE[B]$ 集合的计算。表 5 给出了各基本块中的变量的定值和引用的基本信息。

表 5　各基本块的基本信息

基本块	该基本块的后继	DEF[B]	USE[B]
1	2,3	Ø	Ø
2	4	p	Ø
3	4	{i}	Ø
4	5	{q,j}	{p,i}
5	1	Ø	Ø

根据第 2 节、第 3 节中的活跃变量计算的方法,对图 2 所示的程序,按照拓扑序列 1,2,3,4,5 的顺序,对到达各基本块的入口和出口的活跃变量信息的计算结果如表 6 所列。

表 6　到达各基本块入口、出口的变量的活跃信息

	基本块	MUST-OUTL [B]	MAY-OUTL [B]	MUST-INL [B]	MAY-INL [B]
第一次迭代	5	Ø	Ø	Ø	Ø
	4	Ø	Ø	{p,i}	Ø
	3	{p,i}	Ø	{P}	Ø
	2	{p,i}	Ø	{i}	Ø
	1	Ø	{p,i}	Ø	{p,i}
第二次迭代	5	Ø	{p,i}	Ø	{i}
	4	Ø	{i}	{p,i}	{i}
	3	{p,i}	Ø	{p}	Ø
	2	{p,i}	Ø	{i}	Ø
	1	Ø	{p,i}	Ø	{p,i}
第三次迭代	5	Ø	{p,i}	Ø	{i}
	4	Ø	{i}	{p,i}	{i}
	3	{p,i}	Ø	{p}	Ø
	2	{p,i}	Ø	{i}	Ø
	1	Ø	{p,i}	Ø	{p,i}

从表 6 可以看到,经过 3 次迭代之后,每个集合不再发生变化,因此,第三次迭代的结果代表了到达每个基本块入口和出口的变量的活跃信息。

4.4　利用变量的活跃信息进行部分软件错误的检测

对基本块中的定值语句而言,如果在某个基本块中进行了变量的定值操作,而该变量在该基本块的出口不活跃,意味着该定值操作是一个无用定值。

根据表 4 的结果,检查每个基本块中的定值语句,可以发现,对于图 2 所示的程序中的第 4 个基本块而言,该基本块中存在关于变量 j 的定值操作,而该变量既不属于 $MUST-OUTL[4]$ 也不属于 $MAY-OUTL[4]$,即该变量在基本块的出口是不活跃的,因此该定值操作是一个无用定值。

结束语　应用程序中部分软件故障的产生与变量的状态有关，为检测程序中的这些错误，需要对应用程序进行数据流分析以确定变量之间的定值与引用关系。数据流分析可以通过程序代码的静态分析进行，或者通过程序的动态运行得到。由于程序动态运行不能覆盖程序的所有分支，会对完整性分析产生影响，因此，本文提出了静态的数据流分析方法。该方法从程序的控制结构出发，充分考虑了程序中分支结构对数据流的影响，可以准确反映程序动态运行中的数据流状态，为软件故障的检测提供有力的支持。

参 考 文 献

[1] Hardekopf B, Lin C. Flow-sensitive pointer analysis for millions of lines of code[C]// International Symposium on Code Generation and Optimization, 2011. Chamonix, France, IEEE, 2011: 289-298

[2] Kooli M, Bosio A, Benoit P, et al. Software testing and software fault injection: Design & Technology of Integrated Systems in Nanoscale Era (DTIS), 2015[C]// Naples. IEEE, 2015: 1-6

[3] Yu Hong-tao, Xue Jing-ling, Huo Wei, et al. Level by level: making flow-and context-sensitive pointer analysis scalable for millions of lines of code[C]// Proceedings of the 8th annual IEEE/ACM International Symposium on Code Generation and Optimization, 2010. New York, USA: 2010: 218-229

[4] Li L, Cifuentes C, Keynes N. Precise and scalable context-sensitive pointer analysis via value flow graph[J]. ACM Sigplan Notices, 2013, 48(11): 85-96

[5] Denaro G, PezzAl M, Vivanti M. On the right objectives of data flow testing[C]// Proceedings of the IEEE International Conference on Software Testing, Verification, and Validation. 2014: 71-80

[6] Sui Y, Ye S, Xue J, et al. Making context-sensitive inclusion-based pointer analysis practical for compilers using parameterised summarisation[J]. Software Practice & Experience, 2014, 44(12): 1485-1510

[7] Nair S, Jetley R, Nair A, et al. A static code analysis tool for control system software[C]// IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering. 2015: 459-463

[8] Khan M E. Different approaches to white box testing technique for finding errors[J]. International Journal of Software Engineering and Its Applications, 2011, 5(3): 1-11

[9] Khan S A, Nadeem A. A tool for data flow testing using evolutionary approaches (etodf)[C]// International Conference on Emerging Technologies (ICET). 2013: 1-6

[10] 严蔚敏，吴伟民. 数据结构[M]. 北京：清华大学出版社，2011：180-183

(上接第 479 页)

[8] Bird C, Gourley A, Devanbu P, et al. Open Borders ? Immigration in Open Source Projects[C]// International Conference on Software Engineering Workshops. 2007: 6

[9] Beecham S, Baddoo N, Hall T. Motivation in Software Engineering: A systematic literature review[J]. Information and Software Technology. 2008, 50(9): 860-878

[10] Krogh G, Spaeth S, Lakhani K R. Community, joining, and specialization in open source software innovation: a case study[J]. Research Policy, 2003, 32(7): 1217-1241

[11] Fronza I, Sillitti A, Succi G. An interpretation of the results of the analysis of pair programming during novices integration in a team[C]// Procs of the 3rd Int'l Symposium on Empirical Softw. Eng. and Measurement. IEEE Computer Society, 2009: 225-235

[12] Herraiz I, Robles G, Amor J J. The processes of joining in global distributed software projects[C]// Proceedings of the Int'l Workshop on Global Softw. Dev. for the Practitioner. 2006: 27-33

[13] Dagenais B, Ossher H. Moving into a new software project landscape[C]// Proceedings of Int'l Conf. on Softw. Eng. (ICSE). 2010: 275-284

[14] Ducheneaut N. Socialization in an Open Source Software Community: A Socio-Technical Analysis[J]. Computer Supported Cooperative Work, 2005, 14(4): 323-368

[15] Hahn J, Moon J Y, Zhang C. Emergence of New Project Teams from Open Source Software Developer Networks ?: Impact of Prior Collaboration Ties[J]. Information Systems Research, 2008, 19(3): 369-391

[16] Toral S L, Martínez-Torres M R, Barrero F. Analysis of virtual communities supporting OSS projects using social network analysis[J]. Information and Software Technology, 2010, 52(3): 296-303

[17] Crowston K, Wiggins A, Howison J. Analyzing Leadership Dynamics in Distributed Group Communication[C]// Proceedings of the Annual Hawaii Int'l Conf' on System Sciences(HICSS). 2010: 1-10

[18] Hossain L, Zhu D. Social networks and coordination performance of distributed software development teams[J]. The Journal of High Technology Management Research, 2009, 20(1): 52-61

[19] Datta S, Sindhgatta R, Sengupta B. Evolution of Developer Collaboration on the Jazz Platform: A Study of a Large Scale Agile Project[C]// Proc. of the 4th India Softw. Eng. Conf. (ISEC). 2011: 21-30

[20] Huang S K, Liu K M. Mining Version Histories to Verify the Learning Process of Legitimate Peripheral Participants[C]// The 2nd Int'l Workshop on Mining Softw. Repos(MSR). 2005: 1-5

[21] Allaho M Y, Lee W C. Analyzing the Social Ties and Structure of Contributors in Open Source Software Community[C]// Proceedings of Int'l Conf. on Advances in Social Networks Analysis and Mining. 2013: 56-60

[22] Howison J, Conklin M, Crowston K. FLOSSmole: A collaborative repository for FLOSS research data and analyses[J]. International Journal of Information Technology and Web Engineering, 2006, 1(3): 17-26

[23] He P, Li B, Huang Y. Applying Centrality Measures to the Behavior Analysis of Developers in Open Source Software Community[C]// International Conference on Cloud and Green Computing (CGC). 2012: 418-423

[24] Albert R, Barabási A L. Statistical mechanics of complex networks[J]. Reviews of Modern Physics, 2002, 74(1): 47-97