

文章编号:1001-9081(2010)06-1651-04

## 基于方法切片及入口依赖的回归测试研究

杜章华<sup>1</sup>, 李建华<sup>1</sup>, 胡江明<sup>1</sup>, 程晓菊<sup>2</sup>

(1. 中南大学 信息科学与工程学院, 长沙 410075; 2. 湖南大学 计算机与通信学院, 长沙 410007)

(duzhanghua@sina.com)

**摘要:**为了提高大型软件回归测试效率,将程序切片思想应用于回归测试用例选择过程,提出了方法切片及入口依赖的概念。切片算法 MethodSlice 以方法为基本单元进行切片,提取软件变化影响点,仅选择那些受源程序修改部分影响的测试用例进行回归测试。实验结果表明:与传统的程序切片相比,方法切片更易于大型软件的回归测试过程中,其实现简单,不易出错,在一定程度上提高了软件回归测试效率。

**关键词:**管理信息系统;依赖图;测试效率;测试用例选择;大型软件

**中图分类号:** TP311.11 **文献标志码:** A

## Method-slicing and entry-dependency based research for regression test

DU Zhang-hua<sup>1</sup>, LI Jian-hua<sup>1</sup>, HU Jiang-ming<sup>1</sup>, CHENG Xiao-ju<sup>2</sup>

(1. School of Information Science and Engineering, Central South University, Changsha Hunan 410075, China;

2. School of Computer and Communication, Hunan University, Changsha Hunan 410007, China)

**Abstract:** In order to improve the performance of regression test for large software, the idea of slicing was applied to the selection of the regression testing cases, and Method-Slicing and entry-dependency were proposed in this paper. Method-slicing algorithm sliced the code with method as the basic unit. Only the test cases influenced by the revised source code would be re-tested. The experimental results demonstrate that the method-slicing dramatically improves the efficiency in regression testing of large software, and reduces the cost of software testing to some extent.

**Key words:** Management Information System (MIS); dependency graph; test efficiency; test case selection; large software

### 0 引言

回归测试需要在软件发生改变时运行,用于确保软件在增加了新功能或作了代码修改后,现有功能仍然继续可用。软件测试过程中,代码的新增及修改非常普遍,对整个软件进行回归测试将使测试费用大幅增加,如何获取待测用例的最小集是软件测试人员面临的一大难题,而运行程序切片思想于回归测试用例的选择上一直是软件测试的研究热点<sup>[1-3]</sup>。程序切片的概念最初由 Weiser 于 1979 年提出<sup>[4]</sup>,它具有简化问题和缩小目标范围的特征,后来 Horwitz 等人提出通过建立系统依赖图来进行切片<sup>[5]</sup>,进而用于测试用例的选择。程序切片经过 30 余年的发展,逐步形成了动态切片、静态切片及面向对象切片等方法,由于其将软件变化所产生的影响限制在一个较小的范围之内,切片思想在软件调试、测试及维护方面得到了广泛应用。

目前现有的回归测试用例选择算法大部分基于源代码,对单行程序进行切片,实现技术复杂,易出错,不宜应用于大型程序的回归测试中。由于程序员修改源代码时具有集中性的特点,即往往只修改程序代码中的某一个方法,特别是当敏捷软件开发模式<sup>[6]</sup>提出以后,这种现象越发普遍。如果基于程序方法进行切片,只找到方法之间的依赖关系,不仅能满足日前软件测试的需要,而且能将其应用于大型程序中。本文为了提高回归测试用例选择的效率,根据方法依赖图计算出

方法所依赖的所有程序入口,用于快速判断测试用例是否有必要重新回归。

### 1 方法切片

当软件的某一方法(函数)被修改后,为了找到受影响的其他方法,本文提出了方法切片的思想,应用方法切片能快速确定受影响的其他方法的最小集,利用此方法最小集在回归测试过程中可以仅对那些执行流经过最小集里的方法的测试用例进行回归,理论上提高了测试效率,更重要的是本思想可以应用于大型软件的回归测试中。

**定义 1** 静态切片。静态切片技术是对源程序代码(非运行状态)进行分析,析别出其数据流和控制流,获取语句间的最小依赖集的过程。该技术对程序的输入不作任何假设,所做的分析完全以程序的静态信息为依据,使用该技术的工作量较大,实现困难,不适合对大型程序进行语句切片。

**定义 2** 方法依赖图。一个方法依赖图  $G$  是一个有序二元组,记作  $G = (F, E)$ ,简称依赖图。其中,  $F = \{f_1, f_2, \dots, f_n\}$  是一个有限非空集,其元素是  $G$  的节点,即依赖图中的方法实体;  $E$  是  $G$  的边集,其元素是  $G$  的一条边,表示一个方法依赖关系,是一有向线段,其起点表示依赖主体方法,终点指向被依赖的客体方法,即主体方法依赖客体方法。方法依赖图表示了整个软件系统中方法之间的一种使用与被使用、调用与被调用的关系。

收稿日期:2009-11-20;修回日期:2010-03-08。

**作者简介:**杜章华(1981-),男,湖南长沙人,硕士,主要研究方向:软件测试、软件工程; 李建华(1963-),男,湖南长沙人,教授,博士,主要研究方向:分布式计算、软件工程; 胡江明(1981-),男,湖南长沙人,硕士研究生,主要研究方向:软件测试、XML 数据库; 程晓菊(1982-),女,湖南长沙人,硕士研究生,主要研究方向:嵌入式软件测试、回归测试。

**推论1** 若方法A依赖方法B,方法B依赖方法C,则方法A依赖方法C。

**推论2** 在方法依赖图中不存在环。

**证明** 由定义可知,方法依赖图是方法间的一种调用与被调用的关系,而程序语言(如C、C++、Java等)自身的特点不允许方法间的循环调用,故方法依赖图中不存在环。证毕。

方法切片是将Weiser的程序切片思想<sup>[2]</sup>应用于程序的方法层上,是种静态切片方法,用程序方法代替具体的程序语句进行切片,方法切片的结果为软件方法依赖图的一个子图,而不是程序具体语句的依赖图。

**定义3** 路。在方法依赖图中若存在 $l$ 条边( $l \geq 1$ )的序列 $\{A_0, A_1\}, \{A_1, A_2\}, \dots, \{A_{l-1}, A_l\}$ ,则称节点 $A_0$ 到 $A_l$ 存在路,其中 $A_0$ 称为路的始点, $A_l$ 称为路的终点。

**定义4** 方法切片依赖图。方法切片依赖图 $G'$ 是一个有序二元组,记作 $G' = (F', E')$ ,其中 $G' \subseteq G, F' \subseteq F, E' \subseteq E$ ( $G, F, E$ 的定义参见定义2),满足在 $G'$ 中不存在两条始终点相同的路,且 $G'$ 中的路为 $G$ 中同始终点最短的路。

**定义5** 方法切片。基于方法切片就是找出能够代表因被修改方法(本文称为切片基准方法)的改变在整个系统中所产生影响的方法的最小集过程,此最小集的方法依赖图即为被修改方法对应的方法切片依赖图。

下面给出一个简单的方法切片实例。假设某软件系统中存在如图1所示的方法依赖关系,方法4的改变将引起方法1、2、3、5、6、7、8、9的改变,即方法1、2、3、5、6、7、8、9的改变为方法4的改变所产生的影响,但由于方法4对方法2、3、7、8的影响,最终还是传递给方法1和方法6、9了,故关于方法4的切片就是方法1、5、6与9,即切片基准方法(简称基准方法)——方法4的修改将导致方法1、5、6与9发生变化,且方法1、5、6与9的变化足以代表方法4的改变在整个软件回归测试中所产生的影响。方法4的切片依赖如图2所示。

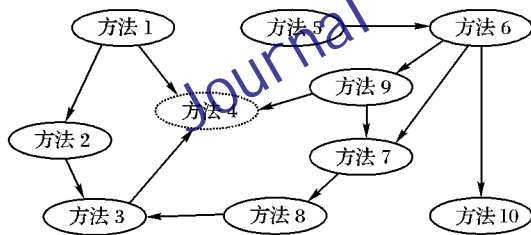


图1 方法依赖

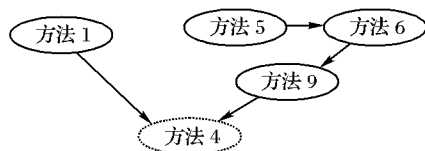


图2 方法切片依赖

**定理1** 如果方法A依赖方法B,则方法B的切片依赖图方法A的切片依赖图并上其至方法B的最短路径。

**证明** 由定义知,方法切片依赖图是方法改变时引起其他方法改变的最小集。由题设及定义,方法A的切片依赖图方法A改变时引起变化的最小方法集,假设B的切片依赖图为 $G'_B$ , $G'_B$ 包含A的切片依赖图 $G'_A$ 与A至B非最短路径 $P$ ,而A至B的最短路径为 $P_{\min}$ ,即 $P > P_{\min}$ ,则 $G'_A \cup P > G'_A \cup P_{\min}$ ,故 $G'_A \cup P$ 为非最小集,与定义矛盾。

以下给出方法切片算法。

算法:MethodSlice( $G, M$ )。

输入:方法依赖图 $G$ ,切片方法 $M$ 。

输出:方法切片依赖图 $G'$ 。

begin:

将依赖图 $G$ 反转,即出边转为入边,入边转为出边,得到图 $G_{\text{temp}}$ ;

在 $G_{\text{temp}}$ 中找到方法 $M$ 至可达叶子节点的最短路径;

将所有找到的最短路径并入依赖图 $G'$ 中;

反转依赖图 $G'$ ;

返回结果 $G'$ ;

end

## 2 入口依赖

在软件回归测试中,对大量的测试用例进行全部回归,代价非常昂贵,有时甚至不可取,如何快速确定待回归的测试,是大型软件回归测试的一个难点。本文为了提出确定待回归测试用例的速度,提出了入口依赖的概念,根据程序入口快速确定测试用例是否需要回归。

**定义6** 入口。入口有外部入口与内部入口两种类型,在方法切片依赖图中,入度为0的节点即为切片基准方法的外部入口,如在图1的方法切片依赖图中,方法1与方法5为方法4的外部入口。在方法切片依赖图中,位于外部节点至切片基准方法的路上的方法,如果在方法依赖图中有一条不经过基准方法的路,则此方法节点为切片基准方法的内部入口。例如,在图2中,方法6、9位于外部入口——方法5与切片基准方法——方法4的路上,但在图1中,仅方法6有一条不经过方法4的路(方法9—方法10),故方法6为方法4的内部入口,而方法9就不是方法4的内部入口。

方法切片基准方法的入口,包括外部入口和内部入口,实质上是找出了外部节点至基准方法的所有关键节点,在软件回归测试中,利用这些关键节点可以快速确定某一个测试用例是否经过被修改的方法,从而决定其是否需要重新回归。

**定义7** 入口依赖。如果存在以下方法依赖关系:方法A依赖方法B,且方法B为切片基准方法,方法A为方法B的入口,则称方法B依赖入口A。

**定义8** 入口依赖图。入口依赖图 $G''$ 是一个有序二元组,记作 $G'' = (F'', E'')$ ,其中 $G'' \subseteq G', F'' \subseteq F', E'' \subseteq E'$ ( $G', F', E'$ 的定义参见定义4),且在 $G''$ 中只允许由切片基准方法与其对应的入口组成。图1中方法4的入口依赖图如图3所示。

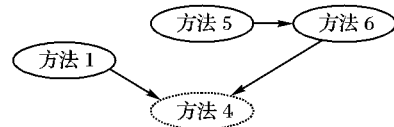


图3 入口依赖

入口依赖图表示了切片基准方法与其入口之间的对应关系,在软件回归测试过程中,如果对每个方法进行路径的确认,将在一定程序上增加算法的时间复杂度。引入入口依赖图后,在保持原有依赖关系的基础上,降低了测试用例执行路径的确认次数。

## 3 回归测试用例选择

在文献[7]中,对测试用例的选择是通过修改前后的程序进行对比,找到所有受影响的测试路径,进而进行测试用例的选择。在程序不太复杂的情况下,例如软件单元测试中,具有不错的可行性,但是随着程序复杂度的提高,特别是在大型软件系统中,该方法的复杂度增加得极快,实现起来困难

大,易出错,因此在面对复杂的软件系统时可行性较小。

通过对软件系统中的方法进行切片,而不是程序语句切片,大大降低了回归测试用例选择算法的复杂性,使基于源代码的软件回归测试算法应用于大型复杂软件系统中成为可能。而入口依赖的提出更进一步减少了测试用例路径的确认次数,提高了算法效率。以下给出基于方法切片与入口依赖的回归测试用例选择算法。

算法:  $GetAffectedPath(M, G, P)$ 。

输入:切片基准方法  $M$  及其对应的入口依赖图  $G''$ , 等待确认的测试用例对应的测试路径  $P$ 。

输出:  $true(P$  需要重新回归)/ $false(P$  不需要重新回归)。

```
begin
    执行路径  $P$ ;
    if  $P$  不经过  $G$  的外部入口;
        return false;
    end if
    找到  $P$  在  $G$  中对应的外部入口, 记为  $Entry_p$ ;
    for  $Entry_p$  至  $M$  的每个内部入口  $InsideEntry_p$ ;
        if  $P$  不经过  $G$  的  $InsideEntry_p$ ;
            return false;
        end if
    end for
    return true;
end
```

算法分析:

某大型系统有  $n$  个测试用例需要进行确认是否需要重新回归, 而此系统有  $M$  个用户界面入口(外部入口), 这  $n$  个测试用例均匀地分布在这  $M$  个入口上。不失一般性, 假设被修改的切片依赖图为  $G$ ,  $G$  有  $m$  ( $m \leq M$ ) 个外部入口, 每条外部入口至基准方法的路上有  $j$  ( $j > 1$ ) 个内部入口, 测试用例在  $j$  个内部入口之间也均匀分布, 则在外部入口需要进行的确认次数为  $n$ 。在某条路径上的第  $i$  ( $j \geq i \geq 1$ ) 个内部入口处需要进行的确认次数为  $n \times \frac{1}{M} \left(\frac{1}{j}\right)^{i-1}$ , 故  $j$  个内部入口共需  $\sum_{i=1}^j \left(n \times \frac{1}{M} \left(\frac{1}{j}\right)^{i-1}\right)$  次确认, 这  $m$  条路径共需要  $\sum_{i=1}^j \left(n \times \frac{1}{M} \left(\frac{1}{j}\right)^{i-1}\right)$  次确认。则针对基准方法判断这  $n$  个测试用例是否需要重新回归共需确认次数为:

$$Confirm_{total} = n \left[ 1 + \frac{m}{M} \sum_{i=1}^j \left(\frac{1}{j}\right)^{i-1} \right] \quad (1)$$

在式(1)中, 由于  $\sum_{i=1}^j \left(\frac{1}{j}\right)^{i-1} \leq \frac{3}{2}$  (证明略), 故  $Confirm_{total} \leq n \left[ 1 + \frac{3m}{2M} \right]$ 。

又因为在大部分软件系统中, 特别是Web系统, 往往给

用户提供独立功能的外部入口, 内部方法之间的交叉不复杂,

即式(1)中的  $m$  往往远远小于  $M$ , 至少可以保证  $\frac{m}{M} \leq \frac{2}{3}$  成立, 则:

$$Confirm_{total} \leq 2n \quad (2)$$

由式(2)可以得知, 判别  $n$  个测试用例是否需要重新进行回归测试时, 对其执行流最多进行  $2n$  次确认, 且几乎有一半以上的测试用例可在程序界面入口处即可作出判断, 算法效率较高。

## 4 回归测试一般步骤

基于方法切片及入口依赖的回归测试一般步骤如下:

- 1) 识别出源程序中被修改的方法集合;
- 2) 由新修改的源程序计算出方法依赖图;
- 3) 根据方法依赖图计算出每个被修改方法的切片依赖图, 由切片依赖图在源程序中设置路径确认标识;
- 4) 根据算法  $GetAffectedPath$  识别出所有需要进行回归测试的测试用例;
- 5) 利用识别出的测试用例进行回归测试。

回归测试过程中, 当程序部分修改后, 如何快速确定需要重新回归的测试用例集是项繁琐的工作, 程序切片在单元测试且功能不太复杂的情况下, 可用性较好, 但其不能应用在大中型软件系统中, 为了提高大型软件系统回归测试效率, 本文引入方法切片及入口依赖的思想, 理论上可以大幅减少测试用例路径确认的次数, 提高测试效率。

## 5 实验与分析

为了确认方法切片在回归测试过程中的有效性, 实验对4个不同的Web系统进行回归测试时将语句切片及方法切片的有效性进行比较, 4个不同的系统规模由小到大, 处理逻辑变由简单到复杂, 实验结果如表1所示。由表1可知: 1) 随着系统规模的不断扩大, 语句切片的结果节点数(代表了实现的难易程度)迅速增加, 对于代码超过5000行的中大型应用系统来说, 几乎不可实现语句的切片, 语句切片只运用在单位测试及微小型系统中; 相反, 方向切片却能很好地运用于在中大型系统中, 实现起来较简单, 不易出错。2) 语句切片决定回归测试用例时进行的路径确认次数较方法切片多, 几乎是方法切片的2~3倍, 由此可知, 方法切片较语句切片更有效, 更易于用于中大型软件的回归测试中。然而, 由表可知, 利用方法切片得到的回归测试用例数较实现需要回归的测试用例数偏多, 这说明方法切片确定的回归测试用例精度有待进一步提高。

表1 实验结果对比

软件系统名称	代码行数	逻辑判断数	界面数	测试用例总数	修改部分	切片结果节点数		回归测试确认次数		实际需要回归的测试例数	回归测试用例数	
						语句切片	方法切片	语句切片	方法切片		语句切片	方法切片
加减乘除计算器	105	8	1	20	加法器	105	5	50	25	3	3	5
简单学生管理器	500	25	3	50	修改字段	100	27	200	70	10	10	15
论文文献管理器	5000	200	10	110	查询	几乎不可实现	50	几乎不可实现	150	35	几乎不可实现	50
教学管理系统	10000	210	20	200	选课	不可实现	75	不可实现	300	40	不可实现	55



为了确定方法切片算法效率跟软件规模的关系,对各种规模的管理信息系统切片时所消耗的时间与空间进行对比,对软件内部处理逻辑复杂度相近的4款规模不同的软件进行实验,实验结果如图4~5所示。由实验结果可知,方法切片的时间与空间复杂度与处理逻辑复杂度相近的软件的规模呈近似的线性关系,其所消耗的存储空间跟软件的方法数成正比。

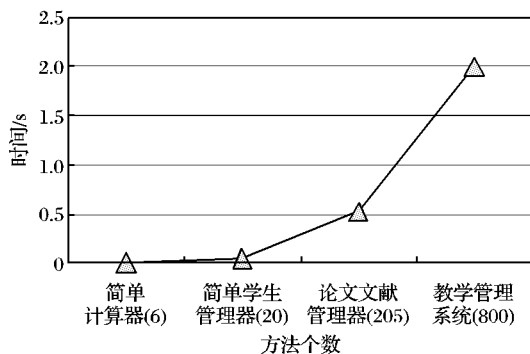


图4 方法切片时间复杂度比较

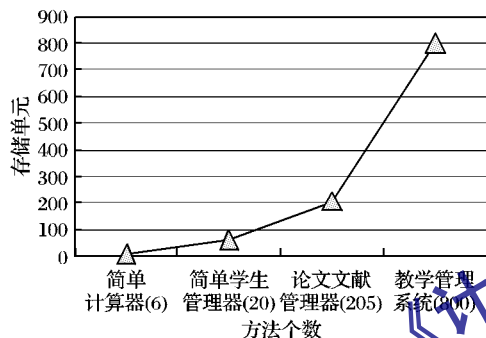


图5 方法切片空间复杂度比较

## 6 结语

通过将程序切片的思想应用在软件的方法上,提出了方法切片及入口依赖的概念,降低了大型软件回归测试过程中待回归测试用例选择算法的复杂度,实用性强。算法极大减少了回归测试用例执行路径的确认次数,提高了大型软件回归测试的效率。下一步的研究重点是如何将方法切片与程序切片相结合,在没有大幅增加算法复杂度的前提下,提高回归测试用例选择的精确度。

### 参考文献:

- [1] LI ZHENG, HARMAN M, HIERONS R M. Search algorithms for regression test case prioritization [J]. IEEE Transactions on Software Engineering, 2007, 33(4): 225-237.
- [2] ROTHERMEL G, UNTCH R, CHU C, et al. Prioritizing test cases for regression testing [J]. IEEE Transactions on Software Engineering, 2001, 27(10): 929-948.
- [3] ROTHERMEL G, UNTCH R, CHU C, et al. Test case prioritization: An empirical study [C]// Proceedings of the International Conference on Software Maintenance. Washington, DC: IEEE, 1999: 179-188.
- [4] WEISER M. Program slicing: Formal, psychological and practical investigations of an automatic program abstraction method [D]. Ann Arbor, Michigan: University of Michigan, 1979.
- [5] HORWITZ S, REPS T, BINKLY D. Interprocedural slicing using dependence graphs [J]. ACM Transactions on Programming Languages and Systems, 1990, 12(1): 26-60.
- [6] MARTIN R C. Agile software development principles, patterns, and practices [M]. [S. l.]: Pearson Education, 2003: 10-12.
- [7] 刘凯枫. 回归测试选择技术研究[D]. 长沙: 湖南大学, 2004.

(上接第1644页)

## 5 结语

本文基于软件再生理论,设计并实现了一个分布式自适应性能监控系统,通过对计算系统运行时资源的采集、分析和控制来保证系统的高性能。

监控系统采用C/S模式,可根据服务端性能实现对多个监控节点的监控。监控系统的特色在于基于自组织映射网络对数据的分析,实现对监控端监控参数的自适应调节,以保证监控端的轻量级、低负载,尽可能地降低监控系统本身对被监控系统的影响。不过,对自组织网络训练过程中会占用监控系统服务端较多的计算资源,对自组织网络训练算法的改进将进一步完善监控系统,这也是继续研究的方向。

监控系统提供多种数学模型对性能变化进行分析和预测,但只设计了决策方法支持系统级的基于时间的再生策略。在后续研究和软件开发过程中,将设计更有效的决策方法,为更多的抗衰策略提供决策支持。

### 参考文献:

- [1] HUANG Y, KINTALA C, KOLETTIS N, et al. Software rejuvenation: Analysis, module and applications [C]// IEEE International Symposium on Fault Tolerant Computing. Washington, DC: IEEE Computer Society, 1995: 381-390.
- [2] CASTELLI V, HARPER R E, HEIDELBERGER P, et al. Proactive management of software aging [J]. IBM Journal of Research &

Development, 2001, 45(2): 311-332.

- [3] VAIDYANATHAN K, TRIVEDI K S. A comprehensive model for software rejuvenation [J]. IEEE Transactions on Dependable and Secure Computing, 2005, 2(2): 124-137.
- [4] VAIDYANATHAN K, SELVAMUTHU D, TRIVEDI K S. Analysis of inspection-based preventive maintenance in operational software systems [C]// International Symposium on Reliable Distributed Systems. Washington, DC: IEEE Computer Society, 2002: 286-295.
- [5] SHI R, YOU J, SUN Y Q, et al. The relationship research between usage of resource and performance of computer system [C]// International Workshop on Computer Science and Engineering. Washington, DC: IEEE Computer Society, 2009: 451-456.
- [6] XU J, YOU J, ZHANG K. A neural-wavelet based methodology for software aging forecasting [C]// Proceedings of International Conference on Systems, Man and Cybernetics. Washington, DC: IEEE Computer Society, 2005: 137-145.
- [7] AVRITZER A, BONDI A, GROTTKE M, et al. Performance assurance via software rejuvenation: monitoring, statistics and algorithms [C]// Proceedings of International Conference on Dependable Systems and Networks. Washington, DC: IEEE Computer Society, 2006: 435-444.
- [8] LIU Y S, WEI C C, GAO S W. Research on software reliability modeling based on Stochastic Petri Nets and module level software rejuvenation [J]. ICIC Express Letters, 2009, 3(3): 607-613.