浙江理工大学学报(自然科学版),第31卷,第4期,2014年7月
Journal of Zhejiang Sci-Tech University (Natural Sciences)
Vol. 31, No. 4, Jul. 2014

文章编号:1673-3851 (2014) 04-0429-05

测试方法对软件失效数据影响的实验分析

李晓雪,丁佐华,胡觉亮
(浙江理工大学科学计算与软件工程实验室,杭州 310018)

摘　要:失效数据常被用于评估软件的质量、监测和预测软件的运行情况,不同的测试方法对失效数据的影响是研究的重点。采用随机测试、分支覆盖测试和分块覆盖测试这3种不同的测试方法选取测试用例集,运用Daikon动态地获取程序不变量,再从这些不变量中提取失效数据,比较分析哪种方法可以获取更多的失效数据。通过实验得出结论:在3种测试方法中,随机测试方法可以获得更多的失效数据。

关键词:失效数据;测试方法;随机测试;测试用例集;程序不变量
中图分类号:TP311.55　　**文献标志码**:A

0　引　言

软件测试是证明一个软件实现预期目标的最常用方法,其主要的目标有两个,一是对质量或可接受性做出评判,二是发现存在的问题。它利用测试用例来操作软件[1-2],通过测试比较实际输出结果与期望输出结果,从而得到失效数据。在实际的测试中,由于成本的限制,不可能用太多的测试用例对软件进行测试,这就需要采取有效的测试方法,用最少的测试用例发现最多的软件缺陷。目前,软件的测试方法有很多种,不同的方法有不同的特点,有各自的优缺点。在本文中,随机测试方法可以节约成本,但是所选测试用例集不能完全地覆盖程序;分支覆盖测试方法的测试用例集可以覆盖程序的每一个分支,分块覆盖测试方法的测试用例集可以覆盖程序的每个出口和入口,通过这两种方法可以有目的的选取测试用例,使得每个测试用例对程序的覆盖范围都不一样,这样就可以认为测试用例集中每个测试用例都是有效的,而两者的区别在于选取测试用例时前一个测试用例对后一个测试用例的影响不同。

通常情况下,通过测试得到的失效数据是静态的,不能把握软件的整体行为,只能获得软件的部分行为。当测试方法不同或者测试用例集不同时,获得的失效数据就会不一样,为软件质量的评估(如软件可靠性的评估)增添了复杂性[3]。正如Chen M H等[4]提到的,当用不同的测试方法选取测试用例时,得到了不同的失效数据。软件质量的评估一直是软件用户和软件开发人员都十分关心的问题,如何能更准确地评估软件的质量好坏,与软件测试有着密切的关系,而失效数据是软件测试的产物之一。

因为不同的测试方法只能得到程序的不同行为,这些行为之间可能有交叉,但是也有不同的方面。实际上一个软件写好后,它的内部缺陷是固定了的,不会因为测试方法的不同而改变,不同测试方法只能从不同的角度来揭示软件对不同环境的反应。希望能得到某些测试数据可以反映程序的整体行为,进而提出以下方法:选取任一个测试方法,运行测试用例,得到程序不变量,提取失效数据,因为不变量是指程序在运行时保持不变的属性的量,刻画了程序的动态性质,体现了软件的整体行为。这样可以更准确地反应程序的内部行为,从程序的内部行为这一层次去评估它的质量。

收稿日期:2013－12－05
基金项目:国家自然科学基金(61210004)
作者简介:李晓雪,女,山西吕梁人,硕士研究生,主要从事软件测试方面的研究。
通信作者:胡觉亮,E-mail:hujhz@163.com

1 程序不变量

程序不变量常被用于追踪软件缺陷[5]、硬件错误[6]和软件评估[7]。如 Pietrantuono R 等[8]通过程序不变量来监测和预测在线软件可靠性。程序不变量揭示了程序在运行过程中保持不变的状态,这些状态对软件的发展、测试和维护有重要的意义。

1.1 如何获取程序不变量

目前有很多种提取程序不变量的工具,常用的有 Daikon[7]和 DIDUCE[5]。本文采用 Daikon 作为提取不变量的工具。

Daikon 作为动态不变量提取工具,利用程序执行来发现可能的不变式[9-10]。它通过修改目标程序来跟踪感兴趣的域,通过一组测试套件来运行修改以后的程序,从而利用跟踪得来的结果推导不变式。这些推导出来的不变式也可以作为形式化的规格说明插入到代码中作为注释,帮助提高程序的可理解性。

下面通过一个计算器例子(图 1)来说明如何得到程序不变量。

a) 生成测试用例集。用原先已有的测试用例生成技术[10]来生成一个测试用例集,该技术将输入域分成若干个等价类,再从每一个等价类中选取测试用例。为了更好的提取不变量,还采用 Gupta N[11]提出的方法进一步提炼测试用例集。在计算器例子中,测试用例集包括 80 个测试用例。

Code(C1)

```
JButton b[]=new JButton[16];
String
s[]={"0","1","2","3","4","5","6","7","8","9",
      "+","-","*","/","C","="},
……
public void actionPerformed(ActionEvent e)
{     ……
  if(flag==10)   m2=m1+m2;
  else if(flag==11)   m2=m2-m1;
  else if(flag==12)   m2=m2*m1;
  else fi(flag==13)   m2=m2/m1;
  else m2=m1;
  ……
}
……
```

图 1 计算器例子

b) 定义不变量类型。Daikon 不变量类型有很多种,如常数 $x=a$、非零值 $x\neq 0$、取值范围 $a\leqslant x\leqslant b$、线性关系 $y=ax+b$、大小关系 $x\leqslant y$、函数关系 $x=fn(y)$、包含关系 $x\in y$ 等,其中 a,b 是常数,x,y 是变量。如果还需要更多的不变量类型,可以自定义。

c) 运行程序和测试用例得到轨迹文件。

d) 得到程序不变量。在该例子中得到的不变量如图 2。通过上述例子得到 12 个程序不变量,例如第二个不变量'*size*(*this*. *b*[])==*size*(*this*. *s*[])'是指数组 *b* 的长度等于数组 *s* 的长度,第五个不变量 *this*. *b* ! =*null* 指参数 *b* 的取值不能为空。

Invariants(T1)

1) JB:::OBJECT
2) size(this. b[])==size(this. s[])
3) this. m1>=0
4) this. flag>=0
5) this. b! =null
6) this. b[] contains no nulls and has only one value, of length 16
7) this. b[] elements! =null
8) this. s[] contains no nulls and has only one value, of length 16
9) size(this. b[])==16
10) this. s. getClass()==java. lang. String[]. class
11) this. m1<=this. m2
12) this. flag<size(this. b[])-1

图 2 通过原程序得到的不变量

1.2 如何得到不正确的不变量

本节描述如何得到不正确的程序不变量,即失效数据。所谓不正确的不变量指的是实际输出与期望输出不同的不变量。原程序 C1 改为程序 C2,如图 3 所示,即将 *JButton b*[]=*new JButton*[16]中的 16 改为 17,同时调换 *String s*[]={"0","1","2","3","4","5","6","7","8","9","+","-","*","/","C","="}中'+'和'-'的位置。得到新的不变量集 T2,如图 4。通过对比发现 1)、3)、4)、5)、7)、10)、11)、12)与图 2 中一样,而 2)、6)、8)、9)却发生了变化。如原先的 2)是 *size*(*this*. *b*[])==*size*(*this*. *s*[]),意为数组 *b* 的长度等于数组 *s* 的长度,而改变后的是 *size*(*this*. *b*[])-1==*size*(*this*. *s*[]),意为数组 *b* 的长度减 1 才等于数组 *s* 的长度。这样,就认为 2)、6)、8)、9)是失效数据。

```
Code(C2)
JButton b[]=new JButton[17];
String
s[]={"0","1","2","3","4","5","6","7","8","9",
     "-","+","*","/","C","="},
……
public void actionPerformed(ActionEvent e)
{    ……
  if(flag==10)  m2=m1+m2;
  else if(flag==11)  m2=m2-m1;
  else if(flag==12)  m2=m2*m1;
  else fi(flag==13)  m2=m2/m1;
  else m2=m1;
  ……
}
……
```

图 3　修改后程序 C2

```
Invariants(T2)
1) JB:::OBJECT
2) size(this.b[])==size(this.s[])
3) this.m1>=0
4) this.flag>=0
5) this.b! =null
6) this.b[] contains no nulls and has only one value, of
   length 17
7) this.b[] elements! =null
8) this.s[] contains no nulls and has only one value, of
   length 17
9) size(this.b[])==17
10) this.s.getClass()==java.lang.String[].class
11) this.m1<=this.m2
12) this.flag<size(this.b[])-1
```

图 4　通过程序 C2 得到的程序不变量

2　实验设置

2.1　实验来源

采用的实验程序来源于 the Software-artifact Infrastructure Repository[12]。The Siemens set 包含 7 个 C 程序,如表 1,它们常用于错误定位和软件可靠性计算。

表 1 描述了实验所需要的 7 个程序。第一列是程序名称。第二列是该程序的错误版本,每个错误版本只包含一个错误,而在本实验中,需要把所有的错误整合在一起,构成一个包含多个错误的版本。第三列是每个程序包含的测试用例个数。第四列是程序的描述,即主要实现的功能。

表 1　Siemens set

程序	错误版本	测试用例	描述
print_tokens	7	4 130	词汇分析
Print_tokens2	10	4 115	词汇分析
Replace	32	5 542	模式识别
Schedule	9	2 650	模式识别
schedule2	10	2 710	模式识别
Tcas	41	1 608	高度分析
tot_info	23	1 052	信息度量

2.2　测试方法

测试方法有很多种,从 3 种不同角度分类为[13]。

a) 从是否需要执行被测软件的角度可以分为静态测试和动态测试。

静态测试方法是不利用计算机运行被测程序,而通过其他手段达到测试目的的方法。相反,动态测试方法是指计算机真正运行被测程序的方法,需要输入测试用例,判断输出结果是否满足要求,以此来预测程序的可靠性、正确性及有效性。

b) 从软件测试用例设计方法的角度可以分为黑盒测试和白盒测试。

黑盒测试方法是一种从用户角度出发的测试,也称其为功能测试。当用这种方法测试时,被测程序就相当于一个黑盒,忽略了程序内部的结构,只需要关心程序的输入和输出之间的关系即可。白盒测试方法是基于程序内部的结构来测试的,检查内部的操作是否按照规定执行,软件的各个部分能否得到充分的利用。

c) 从软件测试的策略和过程的角度可以分为单元测试、集成测试、确认测试、系统测试和验收测试。

单元测试是软件测试的最小的单位,它是针对系统每个单元的测试。它确保每个模块能够正常的工作。集成测试是对已测试过的模块进行组装,其目的主要是检验程序的结构问题。确认测试是检验开发的软件是否满足所有需求和所有功能的最后的手段。系统测试是检测软件与系统其它部分的协调性。验收测试主要从用户的角度着手,是保证产品质量的最后一关。

本文采用 3 种不同的测试方法从测试用例集中选取测试用例,这 3 种方法分别是:随机测试方法、基于分块覆盖的测试方法和基于分支覆盖的测试方法。

随机测试[14]即从程序的输入域中随机的选取一个测试用例。每一个测试用例都有其被选中的概率。在实验中,根据预先定义的发生概论,从程序的

规格说明书中随机的选取至少一个功能进行随机测试以及随机的选取一个合适的测试用例来对这个说明书进行测试。

只有一个入口和一个出口的顺序语句称为一个块[4]。分块覆盖测试方法要求至少有一个测试用例覆盖一个块和所有的测试用例要覆盖所有的程序块。首先根据程序的规格说明书构建一个测试用例集 C,如果测试用例集 C 不能覆盖所有的块,那么选取至少一个未被覆盖的程序块来决定新的测试用例集,然后重新执行新的测试用例集,直到所有的程序块被覆盖。

按分支覆盖准则进行测试是指[15],设计若干测试用例,运行被测程序,使得程序中每个判断的取真分支和取假分支至少经历一次,及判断的真假值均曾被满足。在该试验中,采用 gcov 来得到程序的覆盖信息[16]。

3 实验分析

通过 3 种不同的方法,选取测试用例集 T1(随机测试方法)、T2(基于分块覆盖测试方法)、T3(基于分支覆盖测试方法)。分别运行正确版本程序 C 与 T1、C 与 T2、C 与 T3 得到程序不变量 TI1、TI2 和 TI3,运行错误版本程序 F 与 T1、F 与 T2、F 与 T3 得到程序不变量 FI1、FI2 和 FI3。比较 TI1 与 FI1、TI2 与 FI2 和 TI3 与 FI3 分别得到失效数据 F1、F2 和 F3。

表 2 是 3 种方法得到的 7 个程序的不变量。Random表示随机选取测试用例得到的程序不变量,

表 2 3 种方法得到的失效数据与程序不变量

程序	Random			Block			Branch		
	F1	TI1	Percentage/%	F2	TI2	Percentage/%	F3	TI3	Percentage/%
print_tokens	972	16 854	5.77	951	16 704	5.69	947	16 738	5.66
print_tokens2	1 329	15 971	8.32	1 307	15 864	8.24	1 315	15 021	8.75
replace	2 317	17 583	13.18	2 264	17 438	12.98	2 013	17 328	11.62
schedule	1 372	13 026	10.53	1 342	12 460	10.77	947	13 064	7.25
schedule2	1 528	14 980	10.20	1 497	14 734	10.16	1 294	14 875	8.70
tcas	2 124	20 148	10.54	1 983	19 947	9.94	1 864	20 183	9.26
tot_info	1 865	19 637	9.50	1 744	18 923	9.22	1 721	19 084	9.02

Block 表示基于分块覆盖选取测试用例得到的程序不变量,Branch 表示基于分支覆盖选取测试用例得到的程序不变量。$Fi(i=1,2,3)$表示失效数据,$TIi(i=1,2,3)$表示总的程序不变量,Percentage 表示失效数据占总的程序不变量的百分比。

观察表 2:

程序 print_tokens:5.77%(Random)>5.69%(Block)>5.66%(Branch);

程序 print_tokens2:8.75%(Branch)>8.32%(Random)>8.24%(Block);

程序 replace:13.18%(Random)>12.98%(Block)>11.62%(Branch);

程序 schedule:10.77%(Block)>10.53%(Random)>7.25%(Branch);

程序 schedule2:10.20%(Random)>10.16%(Block)>8.70%(Branch);

程序 tcas:10.54%(Random)>9.94%(Block)>9.26%(Branch);

程序 tot_info:9.50%(Random)>9.22%(Block)>9.02%(Branch)。

在上述结论中,除了 print_tokens2 和 schedule 的结论外,其余的均表明随机测试方法可以选取较多的失效数据。而在 print_tokens2 的结论中,3 种方法得到的失效数据占比相差较少,在 schedule 的结论中,随机和分块覆盖方法可以得到比分支覆盖较多的失效数据占比。另外,通过计算 3 种方法发现失效数据百分比的期望 random 是 9.72%、Block 是 9.57%和 Branch 是 8.6%。这样,可以得出结论,通过随机选取测试用例的方法会得到较多的失效数据。

4 结 论

本文从程序不变量的角度来评价软件测试方法。通过实验说明随机测试能够提取更多的失效数据。从程序不变量的角度来分析不同的测试方法对失效数据的影响,可用于错误定位的研究,有利于评估软件的质量好坏,可以更准确的监测和评估软件的运行情况。实验中用到的程序属于序列化程序,在程序运行的某个时间点只可能有一个 BUG 出现,静态失效数据和动态失效数据都可能揭示该 BUG。对于并发性程序,在同一个时间点有可能同时出现两个或者多个 BUG,而静态的失效数据最多

揭示其中的一个 BUG,本文提出的方法有可能揭示两个或者多个 BUG。所以接下来的工作是:a) 验证该方法是否适用于并发程序;b) 去掉不正确的不变量之间的相关性,分析测试方法对失效数据的影响;c) 比较更多的测试方法,分析不同测试方法对软件失效数据的影响;d) 利用本实验得到的失效数据去评估软件的质量。

参考文献:

[1] Rapps S, Weyuker E J. Selecting software test data using data flow information[J]. IEEE Transactions on Software Engineering, 1985 (4): 367-375.

[2] 刘继华, 陈　策. 软件测试技术的研究进展[J]. 微计算机信息, 2012, 10(28): 16-20.

[3] 陈　明. 软件测试[M]. 北京: 机械工业出版社, 2011.

[4] Chen M H, Mathur A P, Rego V J. Effect of testing techniques on software reliability estimates obtained using a time-domain model[J]. IEEE Transactions on Reliability, 1995, 44(1): 97-103.

[5] Hangal S, Lam M S. Tracking down software bugs using automatic anomaly detection[C]//Proceedings of the 24th International Conference on Software Engineering. ACM, 2002: 291-301.

[6] Sahoo S K, Li M L, Ramachandran P, et al. Using likely program invariants to detect hardware errors [C]//Dependable Systems and Networks with FTCS and DCC. IEEE, 2008: 70-79.

[7] Ernst M D, Cockrell J, Griswold W G, et al. Dynamically discovering likely program invariantts to support program evolution[J]. IEEE Transactions on Software Engineering, 2001, 27(2): 99-123.

[8] Pietrantuono R, Russo S, Trivedi K S. Online monitoring of software system reliability [C]//Dependable Computing Conference (EDCC). IEEE, 2010: 209-218.

[9] Ernst M D, Perkins J H, Guo P J, et al. The Daikon system for dynamic detection of likely invariants[J]. Science of Computer Programming, 2007, 69(1): 35-45.

[10] Ding Zuo-hua, Zhang Kao, Hu Jue-liang. A rigorous approach towards test case generation[J]. Information Sciences, 2008, 178(21): 4057-4079.

[11] Gupta N. Generating test data for dynamically discovering likely program invariants[C]//Proceedings of ICSE 2003 Workshop on Dynamic Analysis. 2003: 21-24.

[12] Do H, Elbaum S, Rothermel G. Supporting controlled experimentation with testing techniques: an infrastructure and its potential impact[J]. Empirical Software Engineering, 2005, 10(4): 405-435.

[13] 朱少明. 软件测试方法和技术[M]. 北京: 清华大学出版社, 2010.

[14] 李海峰, 马　琳. 软件测试[M]. 3 版. 北京: 人民邮电出版社, 2011.

[15] 张晓明, 黄　琳. 软件测试的艺术[M]. 3 版. 北京: 机械工业出版社, 2012.

[16] 王立新. 软件测试数据的高效生成及测试方法研究[D]. 上海: 东华大学, 2010.

Empirical Study on Effects of Testing Methods on Software Failure Data

LI Xiao-xue, DING Zuo-hua, HU Jue-liang

(Lab of Intelligent Computing and Software Engineering, Zhejiang Sci-Tech University, Hangzhou 310018, China)

Abstract: Failure data are often used to evaluate the quality of software, monitor and predict the operation situation of software. The effect of different testing methods on failure data is the focus of the current studies. Firstly, this paper adopts three different testing methods: random testing, branch-coverage testing and block-coverage testing to select test case set, respectively. Secondly, program invariants for each method are obtained dynamically by Daikon tool, and the failure data are then extracted from program invariants. Finally, the number of failure data gotten from each method is compared. It comes to the conclusion that random testing can get the most failure data among the three methods.

Key words: failure data; testing methods; random testing; test case set; program invariant

(责任编辑: 陈和榜)