

文章编号:1001-9081(2006)06-1449-03

软件可靠性测试的加速机理研究

吴玉美, 阮 镰

(北京航空航天大学 工程系统工程系, 北京 100083)

(zzwym@sohu.com)

摘 要:从软件的失效机理出发,对比硬件失效分析了软件失效的特性,结合软件测试的机理,阐述了软件可靠性测试的机理,并在此基础上研究了软件可靠性测试存在加速的可能性,提出了实现加速的思想和方法。

关键词:软件失效; 软件可靠性测试; 加速机理

中图分类号: TP311 **文献标识码:** A

Research on the acceleration principle of software reliability testing

WU Yu-mei, RUAN Lian

(Department of System Engineering, Beijing University of Aeronautics and Astronautics, Beijing 100083, China)

Abstract: The characteristics of software failure were analyzed compared with hardware failure on the basis of software failure principle. The principle of software reliability testing was discussed combined with the principle of software testing. Based on this the underlying acceleration of software reliability testing was studied, and a method of how to implement acceleration was proposed.

Key words: software failure; software reliability testing; acceleration principle

近年来,随着软件的应用范围和规模的迅速扩大,由软件缺陷所引发的产品故障越来越多,甚至引发灾难性事故。在软件的生命周期中,软件测试和软件可靠性测试是保证软件质量,提高软件可靠性的一个十分有效的手段。但由于传统的软件可靠性测试耗时长,资源利用率高,测试代价大在工程实践中很难开展。本文从软件的失效机理出发,阐述了软件测试机理和软件可靠性测试机理,在此基础上分析了软件可靠性测试中潜在的加速机理,并从三个方面提出了提高软件可靠性测试效率实现软件可靠性测试加速的方法。

1 软件的失效机理

1.1 软件失效的基本机理

软件测试和软件可靠性工程研究的关键任务之一是防止软件失效提高软件质量。软件失效主要是因为软件中残留设计开发阶段的失误或错误造成的。由于在软件中残留有错误将导致软件产品的缺陷,使一些功能部件的执行发生偏差。当软件运行在某一特定的条件时,软件缺陷就会引起软件发生故障,使软件运行的内部状态发生意外的改变。这种情况若不能及时得到修正,将致使软件失效,使程序操作背离了程序的需求,最终导致系统全部或部分丧失功能,这就是软件失效的基本机理。

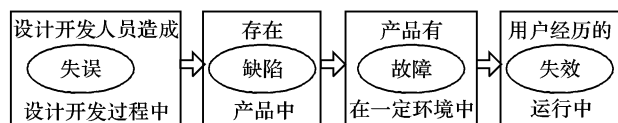


图1 软件失效的机理示意图

软件失效都是由于在软件运行过程中遇到了故障。这些故障就其产生的原因有内在和外在两类。内在原因都是在软

件开发过程中形成且未被排除的潜在缺陷或差错,如有缺陷的、遗漏的或多余的指令或指令集,这些缺陷和差错的来源可能是软件开发者的失误,也可能是恶意逻辑;外在原因都是软件外部给软件提供的各种非期望的条件,这些条件又可分为两种,一种是客观存在于软件外部的系统中的环境异常;另一种是软件运行过程中人员造成的,可能是操作人员的失误,也可能是有人恶意的侵袭。

1.2 软件失效与硬件失效的比较

根据生命周期全过程不同阶段的特点,与硬件失效特性的比较能够更好地了解软件失效的特性见表1。

对软件失效机理和特性的了解可以帮助我们在软件过程的各个阶段中有效地降低软件缺陷的形成和软件失效的发生,特别是在软件测试和软件可靠性测试中可以提高测试效率节约测试成本。

2 软件测试的机理

软件测试是通过检测软件中的缺陷来保证软件质量和可靠性的重要手段,为软件质量的评定提供依据。软件测试涉及过程控制、管理、测试技术、测试环境搭建、工具选择、测试评价等多个方面。虽然软件测试的目的都是尽可能多地发现软件缺陷,但是考虑到被测软件的类型不同,在风险、进度、费用等资源的限制下,需要综合设计测试策略(包括测试计划和测试说明),正确抽象出测试模式,合理选择执行测试的方式(如图2)才能满足特定的测试要求。

被测软件的用途决定软件的类型,不同类型的软件其缺陷种类和分布是不同的;另外,软件的需求规范决定了软件测试的类型。针对不同类型的测试和不同种类的缺陷应用的测试技术也不同。例如,对于实时性要求较高的航空电子系统

收稿日期:2005-12-01;修订日期:2006-02-21

作者简介:吴玉美(1980-),女(满族),辽宁沈阳人,博士研究生,主要研究方向:软件可靠性工程、软件可靠性测试;阮镰(1938-),男,上海人,教授,博士生导师,主要研究方向:软件可靠性工程。

软件(简称航电软件),通常根据需求要进行性能测试,且航电软件一般的缺陷多集中在任务调度和时序方面,因此要采用插桩的侵入式测试方法或采用嵌入式软件的仿真测试方法。

表1 软件失效与硬件失效特性的比较

软件	硬件
失效主要在设计 and 开发过程中引入	除设计过程外,生产过程对产品影响也很大,使用和维护均需加强控制
通过严格的测试,查出缺陷加以排除,为此要精心设计测试用例	通过检验,排除缺陷,为此要适当建立环境应力条件
冗余设计往往不会提高可靠性,反而增加了复杂度,降低了可靠性	相同的部件之间是自然独立的,其适当的冗余可提高可靠性
使用过程中出现失效后必须修改原软件以解决问题;若修改时未带来新的缺陷,软件可靠性就会增长	使用过程中出现故障后,不对原产品进行修改,只需更换或修复实效的部件,使产品恢复良好状态,可靠性一般不会提高
一处修改可能影响其他处,修改时必须考虑完整一致性	维修一处一般不会给它处造成影响
失效有传播性,相同版本的软件都有相同的缺陷	故障没有传播性,故障因老化和耗损造成(不存在设计和生产缺陷情况)
失效强度随故障的排除而下降	失效率变化类似浴盆曲线
可靠性参数估计无物理基础	可靠性参数估计有物理基础
软件的可靠性基本不受外界环境的影响	可靠性受温度、湿度、振动等环境因素的影响
模块间的接口只是一种概念,是不可见的	硬件的接口是可见的
逻辑产品,标准化程度差	物理产品,标准化程度高

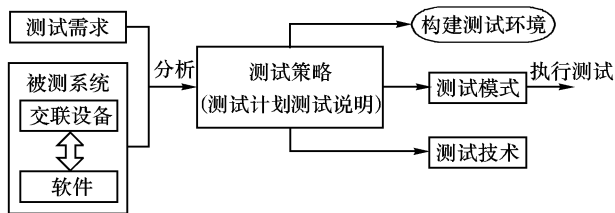


图2 软件测试机理简图

3 软件可靠性测试的机理

软件可靠性测试是指验证软件是否达到可靠性要求,评估软件的可靠性水平和提高软件可靠性而对软件进行的测试,它一方面继承了软件测试的思想,另一方面又有自身的特点,特别是在测试机理上与软件测试是不同的。当前有两种主要的软件可靠性测试方法,一种是基于使用模型的统计测试方法^[1,2];另外一种是基于操作剖面的可靠性测试方法^[3]。这两种方法的基本思想都是基于统计学的基本原理,即根据软件实际使用情况的统计规律的描述,对软件进行随机测试。

3.1 基于使用模型的统计测试

基于使用模型的统计测试是建立软件的使用模型,根据使用模型产生测试用例(软件所有可能输入的一个子集)和度量软件可靠性的过程。使用模型是指系统使用中所有可能

的情形及其发生的概率。使用模型可由许多形式(包括马尔可夫模型和形式化的语法)来表示,目前使用得最广且理论上研究最多的是马尔可夫模型,用马尔可夫过程来描述软件的使用方式,任何下一个发生的事件只和当前的状态有关,不涉及历史信息。因此软件的使用模型可表示为有穷状态、离散参数的马尔可夫链。图3是一个简单的基于马尔可夫链的使用模型。

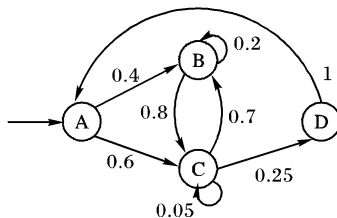


图3 基于马尔可夫链的使用模型

测试用例是从初态开始经过若干个中间状态到达终态的状态和边的序列。产生测试用例时,从初态开始,在每一个状态都生成一个0~1间的随机数,根据这个数选择这个状态的一条出边,转移到下一个状态,直到到达终态。这样产生的测试用例是随机的,符合用户的使用。

3.2 基于操作剖面的可靠性测试

软件的可靠性是由用户的使用决定的,用户的使用方式对于软件可靠性测试和可靠性评估是十分重要的,操作剖面即是对用户使用软件方式的描述,定量地刻画了用户对软件的使用情况,即用户使用软件的统计规律。操作剖面的开发采用自顶向下逐层细化的方法,开发过程如图4所示。

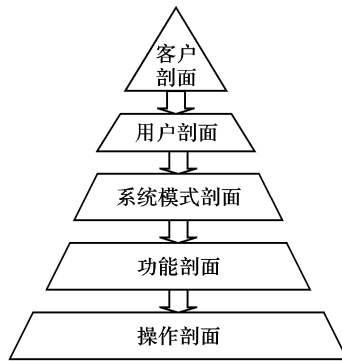


图4 操作剖面的开发

将操作剖面 $\{OP_i | OP_i = \langle O_i, P_i \rangle, i = 1, 2, \dots, N\}$ 中所有操作发生的概率 P_i 求前 j 项和 S_j (O_i 表示操作, P_i 表示操作发生的概率), $S_j = \sum_{i=1}^j P_i$ 形成一个数列 $\{S_j\}$,其中 $j = 1, 2, \dots, N$, N 为软件操作剖面中操作总数,规定 $S_0 = 0$,并有 $S_1 = P_1, S_N = 1.0, S_j - S_{j-1} = P_j$,这里操作相互独立。

根据操作剖面生成测试数据的具体步骤如下:

- 1) 抽取操作:任给一个随机数, $\eta \in (0, 1.0)$ 观察 η 落在哪个区间,若 η 满足 $S_{j-1} < \eta \leq S_j$,则该随机数 η 与 P_j 这个概率值对应,那么这次随机抽到的操作为 O_j ;
 - 2) 确定操作中每个输入变量将取到的具体值:连续型的输入变量要在其取值区间内依概率密度函数抽样;离散型的输入变量要在其可能取值的集合内依概率分布抽样;
 - 3) 通过对操作和各输入变量取值两个步骤的抽样,就生成了一个测试数据。
- 不断重复上述步骤,直到生成所需数量的测试数据为止。

4 软件可靠性测试的加速机理

在工程实践中由于时间进度和资源的限制,采用软件可靠性测试方案对软件(特别是可靠性要求比较高的软件)进行测试所需的时间周期长,测试费用高,常常无法满足工程型号的进度和经费要求。另外,对软件可靠性测试的过程本身进行分析得到:

1) 无论是基于使用模型的统计测试还是基于操作剖面的可靠性测试,都是通过定义和描述使用分配概率然后进行随机抽样得到测试数据,对于概率高的事件或操作抽到的概率也高,从而导致一部分事件或操作总是重复进行,而发生概率较小的事件或操作很少甚至不被抽到。

2) 软件可靠性测试是基于使用的测试,但是发现软件缺陷也应该是其目的之一,而且根据莫非定律在实际使用过程中一些小概率事件是会发生的,一旦发生其失效后果很可能是严重的,对可靠性的影响也是关键的。再者,软件可靠性测试在概率分配中没有考虑到功能模块的复杂度,这样的模块发生失效的可能性大对软件的可靠性也是有贡献的,而发生概率高的操作涉及到的软件模块可能复杂度较低,这样反复抽取和测试会提高测试的费用降低测试的效率;

3) 软件测试过程中的数据和信息也没有在可靠性测试中得到使用,这无疑也会造成大量测试的重复。

因此,加速软件可靠性测试既有工程上的需求,并且从软件可靠性测试机理上也可以分析得到其存在加速的可能性和方法。

4.1 增加环境严酷度

这里的“环境”不同于硬件中的环境,是指“恶劣”输入或输入组合,不是指温度、湿度和振动等,因为软件失效与这些因素无关,在实际环境中进行测试时,通常将包含被测软件的系统作为一个整体(软件不能脱离交联系统),此时对外部环境因素的分析就显得尤为重要,如界面异常交互,人员过操作等因素都可能导致软件环境严酷度的增加。

对于软件自身,实现环境严酷度的方法主要是:1) 改变正常输入值;2) 变换输入类型;3) 变换输入时序。这样能够提高软件输入的严酷度在一定程度上加快软件缺陷的暴露,达到了压缩测试时间、减少测试量的目的。

这种思想来源于软件的异常测试,但是在软件可靠性测试中需要加入概率的分配,而难点则在于如何将恶劣环境下得到的测试结果转换到标准使用环境下进行计算得到可靠性评估结果。

4.2 提高输入强度

提高输入的频度即强度,被认为是在测试量一定情况下减少测试时间的有效方法。在这一过程中会用到加速因子,关于加速因子的定义和确定方法有许多种,它对于提高输入强度、减少测试时间、加快暴露软件的缺陷提高软件的可靠性是非常重要的,目前的实现方法主要有以下两种:

1) N_f/N_e 定义为加速因子, N_f 为实际测试中单位时间内执行的测试用例数, N_e 为正常使用条件下单位时间内执行的测试用例数。由于软件系统的统计失效数据受许多因素的复杂影响,且对于软件测试后的软件其失效发生的可能性大大降低,因此在定义加速因子,计算软件可靠性上可借鉴硬件中的可靠性加速测试理论^[4]。

假设失效出现的时刻服从泊松分布:

$$P_k(t) = \frac{(\lambda t)^k}{k!} e^{-\lambda t}$$

加速后失效强度用 $\lambda(B)$ 表示,则:

$$P_k(t, B) = \frac{(\lambda(B)t)^k}{k!} e^{-\lambda(B)t}$$

令 $\lambda(B) = \lambda_0 + \gamma(B - B_0)^r$, 定义加速因子为: $K = \frac{t}{x(t)}$ $= \frac{\lambda(B)}{\lambda(B_0)}$, 可以看出,式中 γ, r 和 λ_0 都需要实验确定,且与被测软件的类型和结构有关。

2) 加速因子的确定还可以应用以下的思想(见图5):分析和总结出软件过程中影响软件可靠性的因素;借鉴信息融合中的思想和方法,给出加速因子的定义式;利用优化理论进行灵敏度分析,定性地分析得到对某一类软件的可靠性影响较大的因素;利用模糊理论和综合评判法,对这些因素进行综合分析和评价,从而进一步得到加速因子,该因子为一个向量(数组) $A = \{a_1, a_2, \dots, a_n\} (i = 1, \dots, n)$, i 表示第 i 个因素, a_i 表示考虑到第 i 个因素时的加速因子中的一个加速元素。

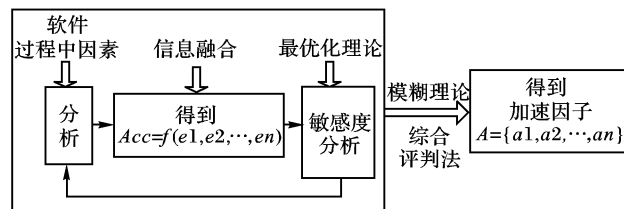


图5 确定加速因子示意图

4.3 基于灰盒分析制定可靠性测试方案

一般认为软件可靠性测试是在测试阶段后期进行的面向使用的黑盒测试,它忽视了软件自身的特征和软件测试阶段的过程信息,这对于时间和资源都是一种浪费。软件的这些内部结构和特征信息对设计测试用例有效地发现软件缺陷是非常有用的,同时软件测试阶段的测试信息也可以减少可靠性测试中的测试工作量:

1) 结合覆盖率信息将其作为生成可靠性测试用例的约束条件和评价测试质量的可测量的标准,从而降低那些使用概率高但是已经得到测试的操作的概率;

2) 考虑程序结构的复杂性,将软件的复杂度作为先验信息制定可靠性测试的操作剖面;

3) 基于对程序控制流和数据流的分析定义得到可测程序子集,对源代码进行改变由此得到的程序子集在语义上与源程序等价但代码量减小并且是独立可测的,从而可以减少单个测试用例的执行时间;

4) 对于相似软件产品,特别是基于构件的软件产品由于其功能模块很多得到了复用产品本身有一定的继承性,且开发环境相同,开发过程遵循一定的模式,则在可靠性测试过程中,只对“新的使用”进行测试,可以极大地减少开发测试剖面的工作量和可靠性测试费用;

5) 对由操作剖面得到的操作序列按照失效行为的一致性进行分类得到不同的操作序列类,不同的操作序列类具有不同的失效行为,这样在测试过程中,每个操作序列类只需要一个操作序列来代表,测试效率会得到提高;

6) 定义“执行概率”,即通过概率的分配决定如果一旦该操作被抽取到那么它需要被测试(在实际中运行)的可能性,该概率的分配是基于对测试过程中信息的捕捉和分析得到(一部分在软件测试中频繁测过的功能就可以分配一个较低的概率),这样通过二次抽样决定第一次抽到的功能 n 是否被执行从而能够以较高的置信度达到在可靠性指标满足的基

(下转第1462页)

第3辆车: $D1 \rightarrow 91(1-2) \rightarrow 11(2-1) \rightarrow 51(2) \rightarrow 92(1-2) \rightarrow 12(2-1) \rightarrow 52(2) \rightarrow D1$

第4辆车: $D1 \rightarrow 61(2-3) \rightarrow 62(2-3) \rightarrow 41(1-2) \rightarrow 42(1-2) \rightarrow D1$

表2 车库及各服务点间距离(km)

d_{ij}	D1	D2	11	21	31	41	51	61	71	81	91	101	12	22	32	42	52	62	72	82	92	102
D1	0	83	118	38	87	127	183	12	27	60	13	53	170	54	154	99	103	187	122	34	196	197
D2	116	0	144	81	84	127	53	37	104	176	31	29	163	85	198	156	84	25	46	86	126	100
11	38	69	0	42	13	61	2	99	171	59	199	26	167	111	143	104	62	93	92	137	192	27
21	25	86	118	0	195	130	95	78	17	161	85	182	111	191	144	149	81	142	179	97	138	57
31	46	131	100	5	0	91	70	185	12	103	163	135	186	6	112	125	5	0	50	86	58	154
41	138	131	36	66	91	0	127	74	159	160	183	99	33	38	190	115	189	189	152	15	93	70
51	200	50	182	56	133	150	0	70	186	121	26	182	188	12	36	87	162	105	94	37	39	177
61	118	2	23	133	78	1	89	0	172	32	101	94	194	167	18	48	50	90	95	69	45	144
71	196	146	42	112	152	126	84	31	0	9	109	64	27	149	168	91	122	82	147	104	79	193
81	74	40	10	78	51	175	96	192	135	0	6	200	161	167	94	42	167	143	80	63	96	15
91	162	65	105	161	188	143	52	195	115	184	0	145	97	16	112	199	116	44	131	190	168	10
101	111	183	47	86	31	177	169	150	161	68	136	0	1	11	37	20	156	159	123	45	175	54
12	187	81	74	20	189	57	103	109	17	177	132	47	0	192	96	45	96	90	53	110	10	66
22	179	189	6	122	18	19	198	34	144	52	81	62	168	0	73	86	47	3	129	189	187	94
32	84	83	156	15	158	159	89	173	172	109	68	163	46	137	0	4	101	112	200	175	81	141
42	184	59	133	26	107	48	145	125	158	109	87	165	118	168	141	0	33	74	21	185	167	22
52	156	150	44	6	75	172	127	5	45	23	101	127	120	67	165	178	0	128	56	18	62	62
62	81	150	82	13	159	33	125	114	83	44	62	146	163	158	64	33	27	0	147	21	7	160
72	130	40	189	171	94	135	184	195	188	136	147	68	19	47	160	13	26	125	0	150	186	25
82	86	142	132	157	163	180	128	50	200	35	147	34	200	190	145	97	90	122	170	0	134	176
92	45	168	24	43	168	135	14	124	169	36	7	129	28	71	15	27	173	3	15	28	0	190
102	37	21	56	50	132	74	196	118	165	164	140	169	44	95	149	27	26	78	133	174	69	0

4 结语

本文从多车库、多货物类型和满载三个方面对一般PDPTW问题进行了扩展,以适应实际应用环境的需要,基于问题本身的NP完全性质,本文提出一种能有效解决该问题的遗传算法。通过实验分析表明:该算法能有效解决复杂PDPTW问题,并取得较好的优化效果。

参考文献:

- [1] 郭伟. 通信网和运输网资源优化问题研究[D]. 上海: 上海交通大学, 2001.
- [2] 贾永基, 谷寒雨, 席裕庚. 求解PDPTW问题的一种快速禁忌搜索算法[J]. 控制与决策, 2004, 19(1): 57-60.
- [3] 潘正君, 康立山, 陈毓屏. 演化计算[M]. 北京: 清华大学出版社, 2000.

(上接第1451页)

基础上减少测试用例的数量,一定程度上减少测试量的目标,见图6。

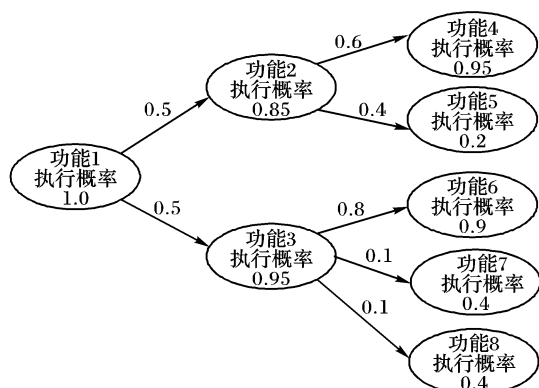


图6 “使用概率+执行概率”的操作剖面描述示意图

上述六点从软件自身、测试过程和软件可靠性测试特点等方面提出了软件可靠性测试加速的可能性和可采取的方法,但是还未能从概率意义上解决定量比较方法的效率问题,在方法的可操作性上有待加强,在工程方面也需进一步的应用验证。

5 结语

本文对软件可靠性测试存在的问题特别是效率问题进行了分析,从增加环境严酷度,提高输入强度和基于灰盒分析制定可靠性测试方案三个方面提出了解决的方法,但对于不同类型的软件还需要更多的实验验证,同时在测试结果的转化处理方面需要更合理的算法支持,且随之而来的测试效率的计算和比较也是未来要进一步研究的内容。

参考文献:

- [1] LINGER RC, MILLS HD. A case study in cleanroom software engineering: the IBM COBOL Structuring Facility[A]. Computer Software and Applications Conference[C]. 1988. 10-17.
- [2] WHITTAKER JA, POORE JH. Statistical testing for cleanroom software engineering[A]. Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences[C]. 1992. 428-436.
- [3] MUSA JD. Operational profiles in software reliability engineering[J]. IEEE Software. 1993, 10(2): 14-32.
- [4] SMAGIN VA. On the physical principle underlying speed-up of software reliability testing[J]. Automatic Control and Computer Sciences, 2003, 37(2): 59-64.