一种软件自动测试成本的估算控制模型

杨善林,阚红星,余本功
(合肥工业大学计算机网络系统研究所,安徽合肥 230009)

摘 要: 根据已有的投资回报(ROI)分析模型,结合 COCOMO 度量结果,在充分考虑回归测试时测试程序维护成本的基础上,提出一种软件自动测试成本估算控制模型. 该模型首先提出了平均维护代价因子的概念,并准确地计算出它的阈值,然后通过这个阈值选择合理的测试方式、控制测试成本,从而为整个测试过程提供动态指导和正确的决策方法.

关键词: 自动测试; 平均维护代价因子; 阈值; 估算/控制成本
中图分类号: TP311.5 **文献标识码**: A **文章编号**: 0372-2112 (2007) 08-1588-04

A Model of Cost Estimation and Control for Software Test Automation

YANG Shan-lin, KAN Hong-xing, YU Ben-gong
(*Institute of Computer Network Systems, Hefei University of Technology, Hefei, Anhui 230009, China*)

Abstract: A model of cost estimation and control for the test automation based on the Return On Investment (ROI) model and COCOMO measurement results is presented. The maintenance cost of regression test is also taken into account. This model proposes a concept of mean maintenance cost factor and precisely predicts its threshold. We can make use of this threshold to find a suitable test mode and control its cost during the testing process in order to provide dynamic guidance and correct strategy for the whole software test automation.

Key words: test automation; mean maintenance cost factor; threshold; estimation/control cost

1 引言

软件测试是保证软件质量的重要环节[1],根据 IEEE 的定义[2],软件测试可分为手工测试和自动测试两大类. 随着软件业迅速发展,自动测试越来越受到人们重视,在某些情况下,自动测试确实有自己优势,如在分布式系统测试[3]、代码覆盖测试[3]、大容量数据测试[4]中,手工测试很难做到完美,甚至根本无法测试. 但我们不能因此而认为自动测试就是解决测试问题的"银弹"(Silver Bullet)[5]. 实际上,即使适合自动测试的软件项目,如果不能恰当地应用自动化测试,其投资的成本会远远高于手工测试. 即在应用自动测试时,必须对自动测试的投资成本进行估算,然后才能决定是否用自动化测试以及如何在合理的范围内控制自动测试的成本.

成本问题一直是自动测试研究的热点之一. John Bible 等人提出通过提取测试用例集最大子集的方法来降低自动测试的成本[6],Douglas D. Lonngren 提出可通过测试程序组件的重用来减少自动测试的成本[7]. 这些方法确实可以降低自动测试的代价,但缺少定量的分析和评估过程.

Douglas Hoffman 对自动测试进行了投资回报(Return On Investment, ROI)分析[8],通过自动测试和手工测试的比较,定量分析了自动测试的投资回报过程,并利用两个实例证明了该模型在实践中的有效性,其分析结果如下:

$$E(n) = \frac{A_a}{A_m} = \frac{V_a + nD_a}{V_m + nD_m} \tag{1}$$

以上各变量以"人月"为单位. 其中 $E(n)$ 是自动测试和手工测试成本的比值; n 表示回归测试次数; A_a, A_m 分别表示自动测试和手工测试的总成本; V_a 表示测试程序的开发代价; V_m 表示手工测试准备的时间支出; D_a 表示自动测试执行完成后的整理工作量; D_m 表示一次手工测试的时间支出. 以上变量除 n 外,都可在测试之前通过不同方式求出, V_a 可用 COCOMO 等模型估算,其余变量可由软件企业历史经验数据得到,因此,这些数据可看作与 n 无关的常量.

式(1)是关于 n 的严格单调减函数,随着 n 的增加, $E(n)$ 在逐渐减小,且其极限值 D_a/D_m 小于 1,也就是说自动测试的成本是在测试程序的多次重复使用中得到回收的.

ROI 分析有精确的量化过程,并指出要小心使用自动测试,但也有不足之处,主要表现在两个方面:(1) ROI 分析模型只考虑了测试程序的设计成本,没有把维护成本作为一个重要因素考虑到模型中去. 其实,随着被测软件的升级和修改,几乎在每一次的回归测试中,测试程序都要进行修改[9]. (2) ROI 分析方法仅仅分析了自

收稿日期:2006-07-31;修回日期:2007-04-16
基金项目:国家自然科学基金(No. 70631003);教育部博士点基金(No. 20040359010)

动测试的投资回报原理,没能给整个自动测试过程提供适时指导,因而不能对自动测试的成本进行有效地控制.

为了使测试者能够在一定成本范围内动态控制和调整自动测试的过程,不盲目地投资自动测试,本文提出一种自动测试成本估算控制模型,该模型建立在 ROI 分析模型和软件企业历史经验数据基础上,利用 COCOMO 度量结果,并充分考虑了测试程序维护成本,为测试者提供一个测试方式选择、测试成本估算控制的有效方法.

2　自动测试成本的估算控制

2.1　含有维护成本的估算模型

软件维护是指在保留现有运行软件主要功能不变的同时对其进行修改的过程[10]. 因此,软件维护是软件生命周期中非常消耗时间和费用的昂贵阶段[11,12].

根据 COCOMO 软件维护量的估算原理[10],软件维护的规模和开发规模成正向关系,即软件的规模越大,开发的时间越长,维护的成本也越大.

令测试程序总的开发成本为 V_a,第 i 次维护的成本为 C_i,则有

$$C_i = \alpha_i \times V_a, \quad 0 < \alpha_i < 1, i = 2,3,4,\cdots,n \qquad (2)$$

正常情况下,维护的成本不会超过开发成本[13],所以 $0 < \alpha_i < 1$. 这里称 α_i 为第 i 次维护代价调整因子.

测试程序设计好后,假定测试程序运行了 n 次,则需要进行 $n-1$ 次的维护(第一次运行时不需要维护). 设 C 为 $n-1$ 次维护的总成本,根据式(2)有

$$C = \sum_{i=2}^{n} C_i = V_a \sum_{i=2}^{n} \alpha_i \qquad (3)$$

设 $\bar{\alpha}$ 为平均维护代价因子,则式(3)可改写为

$$C = V_a \sum_{i=2}^{n} \alpha_i = \bar{\alpha}(n-1) V_a \qquad (4)$$

综上分析,如果测试程序进行了 n 次测试,设 $E(n)$ 为自动测试总成本和手工测试总成本的比值,则根据上面的公式有

$$E(n) = \frac{V_a + nD_a + \bar{\alpha}(n-1)V_a}{V_m + nD_m}, \quad D_a < V_a, D_m < V_m \qquad (5)$$

式(5)就是含有维护过程的自动测试成本估算控制方程.

2.2　测试的成本分析

根据 ROI 分析和式(5)可得 $n=1$ 时,$E(1) = \frac{V_a + D_a}{V_m + D_m} > 1$. 求 $E(n)$ 的导数

$$E'(n) = \frac{(D_a + \bar{\alpha}V_a)(V_m + nD_m) - D_m[(V_a - \bar{\alpha}V_a) + n(D_a + \bar{\alpha}V_a)]}{(V_m + nD_m)^2} \qquad (6)$$

由式(6)可得:当 $\bar{\alpha} \geq \frac{D_m V_a - V_m D_a}{D_m V_a + V_a V_m}$ 时,$E(n)$ 是单调增函数;当 $\bar{\alpha} < \frac{D_m V_a - V_m D_a}{D_m V_a + V_a V_m}$ 时,$E(n)$ 是单调减函数. 由于 $E(1) > 1$,所以要使自动测试的成本得到回收,$E(n)$ 必须是单调减函数,故可得平均维护代价因子的取值范围为:

$$\bar{\alpha} < \frac{D_m V_a - V_m D_a}{D_m V_a + V_a V_m} < 1 \qquad (7)$$

另一方面,即使 $E(n)$ 是单调减函数,如果它的最小值不小于 1 的话,自动测试的成本仍然得不到回收. 所以 $E(n)$ 的最小值即其极限值应小于 1. 根据 $\lim_{n\to\infty} E(n) = \frac{D_a + \bar{\alpha}V_a}{D_m} < 1$,可得

$$\bar{\alpha} < \frac{D_m - D_a}{V_a} < 1 \qquad (8)$$

当 $D_a < V_a, D_m < V_m$ 时,$\frac{D_m V_a - V_m D_a}{D_m V_a + V_a V_m}$ 大于 $\frac{D_m - D_a}{V_a}$,所以要使自动测试成本得到回收,平均维护代价因子 $\bar{\alpha}$ 必须满足 $\bar{\alpha} < \frac{D_m - D_a}{V_a}$,这里称 $\frac{D_m - D_a}{V_a}$ 为 $\bar{\alpha}$ 的阈值. 在自动化测试中,如果 $\bar{\alpha}$ 超过这个阈值,则从投资回报的角度来说,自动化测试是没有意义的.

图 1、图 2、图 3 详细的说明了 $E(n)$、$\bar{\alpha}$ 和 n 三者之间的关系,设 $\alpha_1 = \frac{D_m V_a - V_m D_a}{D_m V_a + V_a V_m}$, $\alpha_2 = \frac{D_m - D_a}{V_a}$.

根据的取值范围,$E(n)$ 的极限值 $\frac{D_a + \bar{\alpha}V_a}{D_m}$ 可能小于 1,也可能大于 1,甚至大于它的初值 $E(1)$. 当 $\bar{\alpha} \geq \alpha_1$ 时,$E(n)$ 是单调增函数,即随着测试次数 n 的增加,自动测试的成本越来越大,其成本永远得不到回收;当

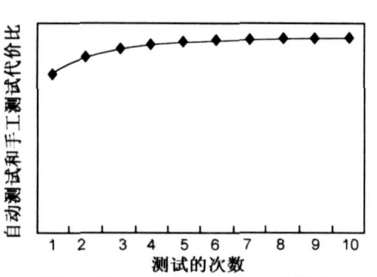

图 1　$\bar{\alpha} \geq \alpha_1$ 时 $E(n)$ 和 n 的关系

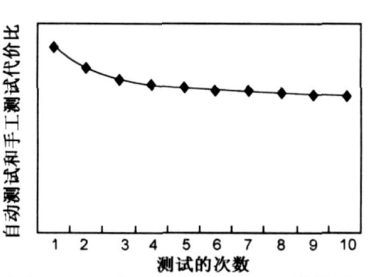

图 2　$\alpha_2 \leq \bar{\alpha} < \alpha_1$ 时 $E(n)$ 和 n 的关系

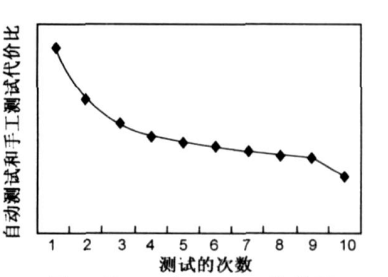

图 3　$\bar{\alpha} < \alpha_1$ 时 $E(n)$ 和 n 的关系

$\alpha_2 \leq \bar{\alpha} < \alpha_1$ 时，$E(n)$ 是单调减函数，即随着测试次数 n 的增加，自动测试的成本会越来越小，但即使减到最小，自动测试的成本仍然大于相应的手工测试；当 $\bar{\alpha} < \alpha_1$ 时，$E(n)$ 是减函数，且随着测试次数 n 的增加，自动测试的成本最终会小于相应的手工测试，即自动测试的成本会得到有效地回收.

2.3 平均维护代价调整因子 $\bar{\alpha}$

在测试程序维护中，不论受什么因素影响，平均维护代价因子 $\bar{\alpha}$ 必须要小于它的阈值，否则自动测试成本就得不到回收. 根据 COCOMO 维护模型，测试程序维护工作量计算方法为[13]

$$C = ACT \times \prod_{j=1}^{15}(VAF_j) \times V_a \tag{9}$$

其中 C 表示测试程序维护的成本；ACT 表示测试程序在每次维护中代码所发生的变化——增加或修改；V_a 是上面所说的测试程序开发的支出；VAF(Value Adjust Factor) 表示与维护成本相关的 15 个成本调整系数，每个成本调整系数有五个级别，每个级别都对应一个具体的数据. 表 1 就是 15 个成本调整系数及其相关值[13].

表 1 软件维护工作量调整系数

成本调整系数	等级					
	很低	低	标准	高	很高	极高
产品属性						
RELY 所测软件的可靠性	1.35	1.15	1.00	0.98	1.10	
DATA 数据库规模		0.94	1.00	1.08	1.16	
CPLX 产品复杂性	0.70	0.85	1.00	1.15	1.30	1.65
计算机属性						
TIME 执行时间约束			1.00	1.11	1.30	1.66
STOR 主存储器约束			1.00	1.06	1.21	1.56
VIRT 虚拟机的易变性		0.87	1.00	1.15	1.30	
TURN 计算机周转时间		0.87	1.00	1.07	1.15	
人员属性						
ACAP 分析员能力	1.46	1.19	1.00	0.86	0.71	
AEXP 应用经验	1.29	1.13	1.00	0.91	0.82	
PCAP 程序员能力	1.42	1.17	1.00	0.86	0.70	
VEXP 虚拟机经验	1.21	1.10	1.00	0.90		
LEXP 编程语言经验	1.14	1.07	1.00	0.95		
项目属性						
MODP 现代编程规范	1.24	1.10	1.00	0.91	0.82	
TOOL 软件工具的使用	1.24	1.10	1.00	0.91	0.83	
SCED 所需的开发进度	1.23	1.08	1.00	1.04	1.10	

根据表 1，15 个成本调整系数由四大类组成，分别为产品属性，计算机属性，人员属性和项目属性. 在 $n-1$ 次的维护中，每次维护的产品属性、人员属性以及项目属性中的编程规范应该是一样的. 因此 15 个成本调整系数实际上只有 6 个可能会在每一次的维护中发生改变. 根据这一特性，可以把 15 个成本调整系数分为两大类，分别为不变调整系数和可变调整系数，在每次维护中不改变的 9 个系数称不变调整系数，其余的为可变调整系数. 因此，每次维护的调整系数实际上就是不变调整系数和可变调整系数的乘积. 为了便于计算，用方阵 A 表示 6 个可变调整系数形成的二维表，$a_{i,j}(1 \leq i \leq 6, 1 \leq j \leq 6)$ 是方阵中的元素，表示某一个可变调整系数的具体值.

令 ACT_k 为第 k 次维护中代码所发生的变化，γ_k 表示第 k 次维护中 ACT_k 和 6 个不稳定调整系数的乘积，则 γ_k 可表示为

$$\gamma_k = ACT_k \prod_{i=1}^{6} a_{i,j}, \quad j \in \{1,2,3,4,5,6\} \tag{10}$$

这里可称 γ_k 为不稳定因子.

同样，令 β 为 9 个不变调整系数的乘积，并称 β 为稳定因子，α_k 为第 k 次维护代价调整因子，则 α_k 可表示为

$$\alpha_k = \beta\gamma_k, \quad k = 2,3,\cdots,n \tag{11}$$

由式(10)，(11) 可知，通过调整 β 和 γ_k 的值，可对整个自动测试成本进行跟踪控制，把 α_k 控制在 $\bar{\alpha}$ 阈值范围之内，使自动测试的成本能得到有效回收. 通常有两种方法调整 α_k 的值，其一是在不同测试阶段调整 β 和 γ_k 的值. 如在某一特定测试阶段，β 和 γ_k 可能有确定值，不能调整，这时如 β 和 γ_k 的乘积大于 $\bar{\alpha}$ 阈值，可以通过其他测试阶段来调整 β 和 γ_k 的值，使它们的乘积平均小于 $\bar{\alpha}$ 阈值；其二是在各因子之间进行调整，通过 β 和 γ_k 之间互相调整，使得两者乘积小于 $\bar{\alpha}$ 阈值.

但是 β 和 γ_k 的调整范围是有限度的，它受到特定项目和特定测试阶段的影响，不可能任意调整. 例如对一个确定的测试项目来说，根据式(11) 估算出 α_k 一定大于 $\bar{\alpha}$ 阈值，那么从投资回报的角度来考虑，最好选用手工测试. 因此 $\bar{\alpha}$ 阈值是选择自动测试还是手工测试的依据，也是控制整个测试阶段成本的依据，指导整个测试的过程.

3 估算模型的应用

某个测试项目，利用 COCOMO 等模型估算出 $V_a = 56$ 人月，利用测试的历史数据得到 $V_m = 6$ 人月，$D_a = 0.1$ 人月，$D_m = 15$ 人月，n 表示测试执行的次数.

首先计算平均维护代价因子 $\bar{\alpha}$ 的阈值. 根据式(8)，$\bar{\alpha}$ 的阈值为 0.2674.

假设该测试项目是对一个应用程序的 GUI 进行测试，根据测试经验，估计其平均代码的改变量为 30 %，即 $ACT = 0.3$. 根据式(11) 和表 1，对每个调整系数取它的平均值然后相乘，可得乘积为 2.78，进一步可得平均维护代价因子 $\bar{\alpha} = 0.834$，远大于它的阈值 0.2674，因此在这种情况下，该测试项目不宜用自动测试.

如果该项目代码的修改量约为 10 %，即 $ACT = 0.1$，

每个调整系数也取它的平均值,那么可得平均维护代价因子 $\bar{\alpha}=0.278$,与它的阈值 0.2674 相差较小,可以用调整的办法使它降低到阈值以下,因此可考虑用自动测试.

当然,以上计算并不是说平均代码改变量为 30 % 就一定不能用自动测试,修改量为 10 % 就一定能用自动化测试,因为上面的调整系数取的是平均值. 实际上,如果平均代码改变量为 30 % 甚至更大,而 15 个调整系数在某种环境下都取较小的值,这时它们乘积就可能小于 $\bar{\alpha}$ 的阈值,因此可以使用自动测试. 同样,平均代码改变量为 10 % 也可能不适合用自动测试.

同样项目,如果用 ROI 分析模型来分析,可得当 $n>3$ 时, $E(n)<1$,这说明无论如何修改测试程序代码,当它被重复执行 4 次时,自动测试的成本就可得到回收,这显然不现实.

图 4 显示了 $\bar{\alpha}$ 在阈值范围内,$\bar{\alpha}$ 与成本回收、测试次数三者之间的关系. 从图中可以看出 $\bar{\alpha}$ 越小成本回收的速度越快. 当 $\bar{\alpha}=0$ 时,本模型即为 ROI 分析模型,测试程序只要执行 4 次,成本即可得到回收,因此 ROI 分析模型只是本模型的一个特例.

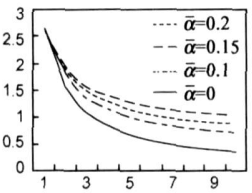

图 4　在阈值范围内 $\bar{\alpha}$ 的取值与成本回收的关系

4　结论

由 ROI 分析模型可知,一次自动测试的成本要远大于一次手工测试的成本,自动测试成本是在反复回归测试中得到回收的,因此需要反复测试的软件产品才适合用自动测试,而那些测试次数少的定制型项目则不适合自动测试. 但适合自动测试的项目,如果不能对其成本进行有效估算和控制,反复回归测试后其成本仍然可能高于手工测试.

虽然 ROI 分析模型被作者证明在实践中是有效的,但没有考虑回归测试时测试程序的维护成本,所以难免有些误差. 本文在 ROI 分析模型基础上,结合 CO-COMO 度量结果,提出一种自动测试成本估算控制模型. 该模型在充分考虑回归测试时测试程序维护成本的同时,提出了平均维护代价因子的概念,并可用简单的方法计算出它的阈值,通过阈值选择测试方式和控制整个自动测试的过程,从而使整个测试成本控制在可以有效回收的范围内.

本模型是建立在历史数据基础上,因此适合有测试经验的组织,对没有测试经验的组织只是提供一种方法指导. 同现有的方法相比,本模型更接近测试活动的实际.

参考文献:

[1] Salem Ahmed M, Rekab Kamel, Whittaker James A. Prediction of software failures through logistic regression[J]. Information and Software Technology, 2004, 46(12): 781 - 789.

[2] 829-1998, IEEE Standard for Software Test Documentation [S].

[3] Keith Stobie. Too darned big to test[J]. Queue, 2005, 3(1): 30 - 37.

[4] Stefan Berner, Roland Weber, Rudolf K Keller. Observations and lessons learned from automated testing[A]. Proceedings of the 27th International Conference on Software Engineering [C]. Missouri, USA: IEEE Press, 2005. 571 - 579.

[5] Brooks, F P Jr. No silver bullet essence and accidents of software engineering[J]. Computer, 1987, 20(4): 10 - 19.

[6] John Bible, Gregg Rothermel, David S Rosenblum. A comparative study of coarse-and fine-grained safe regression test-selection techniques[J]. ACM Transactions on Software Engineering and Methodology, 2001, 10(2): 149 - 183.

[7] Douglas D Lonngren. Reducing the Cost of Test Through Reuse [DB/OL]. http://www.serendipsys.com/ReduceCost.pdf. 2005-05-17.

[8] Douglas Hoffman. Cost Benefits Ananlysis of Test Automation [DB/OL]. http://www.softwarequalitymethods.com/Papers/Star99%20model%20Paper.pdf. 1999-04-16.

[9] Skoglund Mats, Runeson Per. A case study on regression test suite maintenance in system evolution[A]. Proceedings-20th IEEE International Conference on Software Maintenance[C]. Chicago Illinois, USA: IEEE Press, 2004. 438 - 442.

[10] Boehm B W. Software Engineering Economics[M]. Upper Saddle River, NJ: Prentice-Hall, 1981.

[11] Martin J, Mcclure C. Software Maintenance: The Problem and Its Solutions[M]. Englewood Cliffs NJ: Prentice-Hall, 1983. 23 - 24.

[12] Stephen Schach 著, 袁兆山, 等译. Software Engineering with JAVA[M]. 北京: 机械工业出版社, 1999. 95 - 101.

[13] Barry W Boehm 著, 李师贤, 等译. 软件工程经济学[M]. 北京: 机械工业出版社, 2004. 86 - 95.

作者简介:

杨善林　男, 1948 年生于安徽怀宁, 教授, 博士生导师. 主要研究方向为管理信息系统、智能决策支持系统和知识发现.
E-mail: slyang@mail.hf.ah.cn

阚红星　男, 1972 年生于安徽肥东, 博士生, 主要研究方向为软件工程, 自动化测试, 决策分析.