基于组件的软件测试时间分配策略研究*

朱 嫄, 高建华

(上海师范大学数理信息学院计算机系 上海: 200234)

jhgao@shtu.edu.cn

摘要: 本文提出了三种基于组件的软件测试时间分配的策略。这些策略根据各组件不同的系统参数，对于两种受限情况（给定总的失效率、给定总的测试时间）进行了分析，给出了各种情况下的软件测试时间分配策略。对于分配基于组件的软件测试时间是一种可行的方法。

关键字: 组件，测试，分配策略

1 引言

随着软件业的迅速发展，软件的规模也不断扩大，复杂度也不断增加。为了解决软件规模和复杂度所带来的问题，进一步提高软件复用度，基于组件的程序设计成为软件设计和开发的一个新趋势。基于组件的程序设计是在面向对象技术的基础上，进一步实现了更高层次的软件重用性和异质体系结构中的互操作性，为提高软件质量、尽可能地缩短软件开发周期提供了新的途径。

但是，基于组件的软件与传统的软件有着诸多不同，如开发过程、开发的侧重点、可靠性估计、测试的方法等。虽然在这些方面已有一些研究成果，但是还有很多问题有待于解决，比如基于组件的软件测试问题。通常，对于一个基于组件的软件测试会有各种不同的限制，主要有以下两种（为了充分地在限制条件内，分配测试时间，①、②两式通常取等号。）:

•情况 1: 给定总的失效率，即

$$\sum_{i=1}^{n} \lambda_i' \le \lambda_{tot} \quad i=1, 2, \ldots, n \qquad ①$$

•情况 2: 给定总的测试时间，即

$$\sum_{i=1}^{n} t_i \le t_{tot} \quad i=1, 2, \ldots, n \qquad ②$$

但是，对于每一个组件没有一个具体限制，这就需要测试人员根据给定的限制条件给出一个合适的分配策略。针对情况 1，应该给出失效率 λ 的合理分配策略，由于分配失效率的目的在于得到一个预计的测试停止时间，所以问题归结于测试时间的分配；而对于情况 2，也应当给出各个组件测试时间 t 的分配方案。最终实现使用尽可能少的时间，达到较高的软件可靠性（满足软件总的限制）。

在基于组件的软件系统中，各组件的使用时间率、重要程度、复杂度、初始失效率、错误衰减率等参数均可以作为分配测试时间的依据。它们的定义分别如下：

•组件的使用时间率是指某一组件的使用时间与整个软件系统使用时间的比率。

•组件的重要程度是指组件在发生失效后，对整个系统的影响程度。

•组件的复杂度是各组件间的相对逻辑复杂程度。

* 本文得到上海市曙光计划项目（2002SG53）及上海市高校科技发展基金重点项目（CL200222）的资助

作者简介：朱嫄（1979 年），女，硕士研究生，主要研究软件可靠性设计理论与方法、软件测试技术；

高建华（1963 年），男，工学博士，教授，主要研究软件可靠性理论与设计、软件开发环境与开发技术、数据安全与计算机安全、网络测试、LSI/VLSI 测试等领域。

•组件的初始失效率是程序未经测试时的预计失效率。

•组件的错误衰减率是随着时间的增加，失效率减小的快慢程度。

本文针对上述问题，根据测试中的限制和各组件的已知参数的不同，研究了三种不同的分配测试时间的策略。

2 根据不同的系统参数情况，使用不同的策略

2.1 按使用时间率来分配

1）给定总的失效率

这是最为简单的一种分配方法，假设一个串联系统给定总的失效率 λ_{tot}、并已知了各个组件的估计使用时间率 ρ_i（$\rho_i=d_i/T$，d_i 是第 i 个组件的估计使用时间,T 是整个系统的使用时间。 当然也可以以各组件的调用次数作为近似的分配依据），则我们根据各个组件的估计使用时间率 ρ_i 来分配 λ_{tot}，使每个组件具有自己必须达到的失效率 λ_i，从而确定何时停止测试 t_i。

众所周知，软件可靠性的公式为：

$$R(t)=e^{-\lambda t} \qquad ③$$

对于一个串联系统，按调用时间分配后的软件系统失效率可以表示为

$$\lambda_{tot}=\sum_{i=1}^{n}\rho_i\lambda_i \qquad i=1，2，\ldots，n，其中\ \lambda_i'=\rho_i\lambda_i \qquad ④$$

由于失效率与运行时间成反比，根据时间分配失效率的公式为

$$\lambda_i=\frac{\rho_j}{\rho_i}\lambda_j \qquad i,j=1，2，\ldots，n \qquad ⑤$$

根据公式①、④、⑤，就可以求得每个组件的失效率 λ_i。若已知各组件各自的目标软件可靠性，则根据公式③，就可以近似地求出每个组件的测试时间 t_i。

例 1 一个基于组件的软件系统，给定的 λ_{tot} 为 0.01/天，组件 1 和组件 2 的调用时间率分别为 0.6、0.4。并已知组件 1 和组件 2 的目标可靠性均为 0.92，其结构见图 1：

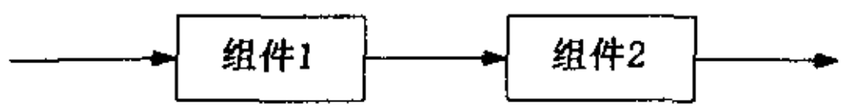

图 1 一个串联的基于组件的软件系统

根据公式①、④、⑤，得：

$$\begin{cases}0.6\lambda_1+0.4\lambda_2=0.01\\ 0.6\lambda_1=0.4\lambda_2\end{cases}$$

求解得 $\lambda_1\approx0.0083$/天，$\lambda_2=0.0125$/天，λ_1、λ_2 为两个组件分别应该达到的失效率。根据公式③，得 $t_1\approx10.0058$ 天,$t_2=6.6705$ 天，由于测试期间内 λ_i 可能是变化的值，所以这个测试时间仅仅是一个近似值。

2）给定总的测试时间

若一个系统给定总的测试时间 t_{tot}、各个组件的估计使用时间率 ρ_i。显而易见，使用时间长的组件应该分配多一点的测试时间，以确保整个软件系统的可靠性，因此得出了公式⑥。使用公式②、⑥即可求

出各个组件所分配到的测试时间。

$$t_i = \frac{\rho_i}{\rho_j} t_j \quad i,j=1，2，\ldots，n \qquad ⑥$$

2.2 按调用情况、复杂度、重要程度来分配

1）给定总的失效率

在给定系统总的失效率 λ_{tot} 和已知各组件实际使用时间 ρ_i 的前提下，如果还知道了各组件在整个系统中的重要程度 U_i 以及各组件的复杂度 C_i。我们就使用这些参数给出更为合理的分配策略。

（1）各组件估计使用时间率 ρ_i：在 2.1 节中我们已知，对于使用率高的组件，我们应该增加它的测试时间，以保证该组件有较低的失效率。

（2）各组件的重要度 U_i：各组件的重要度 U_i 是指组件在发生失效后，对整个系统的影响程度。如果 i 组件的失效会引起系统的失效或严重后果，则令 U_i 为 1；若该组件的失效只引起系统较小的损失，则 $0 \le U_i \le 1$。对于重要度高的组件，应该使其达到较低的失效率。

（3）各组件的复杂度 C_i：软件越复杂，其发生错误的可能性就越大[2]。所以对于复杂度大的组件，应该分配较大的失效率，当然，如果该组件的重要度高、使用的时间也多，那么就应该分配较少的失效率。

根据上述参数的分析，可以给出

$$\lambda_i / \lambda_j = \frac{C_i}{U_i \rho_i} / \frac{C_j}{U_j \rho_j} \quad i,j=1，2，\ldots，n \qquad ⑦$$

这样，利用公式④、⑦，就可以在限定失效率的情况下，合理地分配各个组件的失效率，从而再分配各个组件的测试时间。

2）给定总的时间

如果系统给定总的测试时间 t_{tot}，并且同样已知参数 ρ_i、C_i、U_i，我们同样可以使用这些参数给出合理的分配策略。

（1）各组件估计使用时间率 ρ_i：对于使用率高的组件，我们当然应该增加它的测试时间。

（2）各组件的重要度 U_i：对于重要度高的组件，应该花费更多的时间进行测试，确保其可靠性。

（3）各组件的复杂度 C_i：软件越复杂，其发生错误的可能性就越大。对于复杂的组件,我们应该分配更多的时间进行测试。

由此得出分配公式⑧，如下：

$$t_i / t_j = \rho_i U_i C_i / \rho_j U_j C_j \quad i,j=1，2，\ldots，n \qquad ⑧$$

2.3 根据初始失效率、错误衰减率来分配

1）给定总的失效率

在现代大型的基于组件的软件开发中，各个组件通常不是由一个开发商开发的，而是由多个开发商开发的，更多的是使用现成商用组件即 COTS（Commercial Off-The Shelf components）[3]。这样的 COTS 组件，必须要有详细的说明，这些说明中提供的参数同样可以用来进行测试时间的分配。

比如各个组件已知了初始失效率 $\lambda_{i,0}$ 以及失效衰减率 μ_i，则根据指数可靠性模型理论[4]，每个组件的失效率为：

$$\lambda_i = \lambda_{i,0} e^{-\mu_i t_i} \quad i=1，2，\ldots，n \qquad ⑨$$

由公式⑨可知，在相同的时间下，软件的失效衰减率 μ_i 越大，失效率 λ_i 减少得就越快，若初始失效率 $\lambda_{i,0}$ 越小，失效率 λ_i 就越小，其关系图见图 2、3。因此，给出了公式⑩这样的分配策略。由于本策略考虑到了初始失效率（initial failure rate in C_j at time 0）和错误衰减率（failure decay parameter for C_j during testing）这两个参数，简称 IFR-FD 策略。

$$\lambda_i / \lambda_j = \lambda_{i,0}\mu_j / \lambda_{j,0}\mu_i \quad i,j=1，2，\ldots，n \qquad ⑩$$

例 2 一个软件仍由 3 个组件组成，已知 λ_{tot}=0.003/天，$\lambda_{1,0}$=0.03/天，$\lambda_{2,0}=\lambda_{3,0}$=0.045/天，$\mu_1=\mu_2=3,\mu_3=1$，求 λ_i 以及 t_i。

同时使用 IFR-FD 策略、平均分配的策略（即 $\lambda_1=\lambda_2=\lambda_3$），以及文献[4]中的最优策略解决该问题，三种策略的比较见表 1。

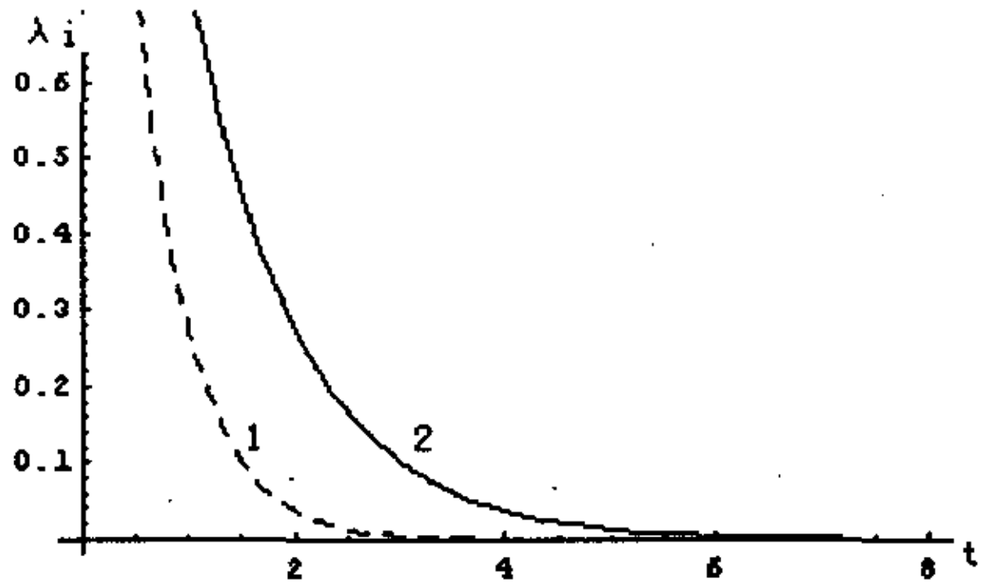

图 2 $\mu 1>\mu 2, \lambda_{1,0}=\lambda_{2,0}$ 的情况下，λ_1 、λ_2 的变化情况

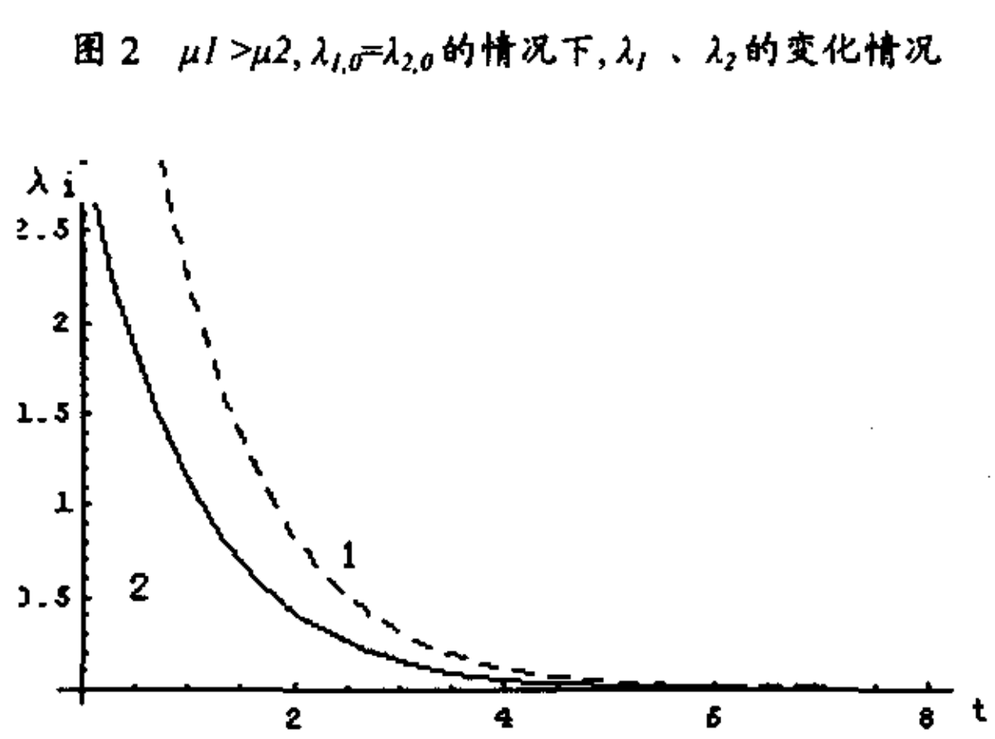

图 3 $\lambda_{1,0}>\lambda_{2,0}, \mu_1=\mu_2$ 的情况下，λ_1 、λ_2 的变化情况

表 1 IFR-FD 策略、平均分配策略、最优策略的比较

参数	IFR-FD 策略	平均分配策略	最优策略
λ_i	λ_1=0.000632/天， λ_2=0.000947/天， λ_3=0.001421/天	λ_1=0.001/天， λ_2=0.001/天， λ_3=0.001/天	λ_1=0.0006/天， λ_2=0.0006/天， λ_3=0.0018/天
t_i	t_1=1.2867 天， t_2=1.2870 天， t_1=3.4553 天	t_1=1.1337 天， t_2=1.2689 天， t_3=3.8066 天	t_1=1.3040 天， t_2=1.4392 天， t_3=3.2189 天
t_{tot}	6.0290 天	6.2092 天	5.9621 天

可见，在达到相同的失效率的情况下，使用 IFR-FD 策略花的测试时间较少。它的效果介于最优策略（相差 0.0669 天）和平均分配策略（相差 0.1802）之间，并且算法比最优策略简单。

2）给定总的时间

如果系统给定总的测试时间，我们也知道了同情况 1）的参数。经过分析可得，软件的失效衰减率 μ_i 越大，失效率 λ_i 减少得就越快，可以分配较少的测试时间 t_i；若初始失效率 $\lambda_{i,0}$ 越小，也可以分配较少的测试时间 t_i。，可得如下分配公式。

$$t_i / t_j = \lambda_{i,0}\mu_j / \lambda_{j,0}\mu_i \quad i,j=1，2，\ldots，n \tag{11}$$

2．4 多应用系统的分配方法

在上述章节中，讨论的三种情况均是多个组件被一个应用程序所调用的情况。如果是一个多应用系统，则应该加入更多的参数进行考虑。如有一个系统，它由三个应用程序 A_1、A_2、A_3 组成，又共有四个组件 C_1、C_2、C_3、C_4。三个应用的调用情况为：

•A_1 使用 C_1、C_2

•A_2 使用 C_2、C_3、C_4

•A_3 使用 C_1、C_2、C_3、C_4

所以，既要考虑每个应用中各组件的使用情况，还要考虑每个应用间各组件的使用情况。对于这样的多应用系统，只要在上述策略中加入各应用之间各个组件的使用情况的参数，就可以使用相类似的方法来进行分配。如组件 C_2 同时被 A_1、A_2、A_3 三个应用所调用，那么，应该保证 C_2 有较低的失效率，应该分配较多的测试时间 t。类似地，组件 C_2 只被 A_2 所用，所以可以分配较少的测试时间。当然，各个应用在整个系统中的重要程度、使用时间等参数也可以作为分配的参数及依据。

3 总结

在本文中，我们讨论了几种分配测试时间的方法，根据已知的参数的不同，使用不同的分配方法。这些分配策略均在满足一定测试条件的基础（给定总的失效率或给定总的测试时间）上，优化测试时间的分配。这些方法在实际运用过程中，是简单可行的。

但是，仍有一些问题尚未解决。比如，上述方法均要在知道足够多的参数后，方可使用这些方法。由于基于组件的软件开发还属于起步阶段，各个组件的说明可能还不规范、细致，可能会导致无法使用这些分配策略。又如，由于各个组件通常由不同的团队进行开发，所以每个团队所给出的组件说明指标（参数）可能不一致，如何对这样的组件进行测试时间的分配，也有待于我们进一步考虑。

基于组件的软件为软件开发带来的好处是显而易见的，虽然现在对这种软件的测试策略还没有传统的软件那么完善，但随着基于组件的软件开发设计的发展和规范化，相应的、更好的测试策略会出现，从而进一步缩短测试时间，提高软件可靠性。

参考文献：

[1]黄锡滋,软件可靠性、安全性与质量保证[M],电子工业出版社，2002 年 10 月, pp.136-137, 206-207.

[2]Lui Sha, Using Simplicity to Control Complexity [J], IEEE SOFTWARE, Jul./Aug. 2001,pp.20-28

[3]Dick Harmlet, Dave Mason, Denise Woit, Theory of Software Reliability Based on Components [C],23rd International Conference on Software Engineering, (ICSE'01), 2001,pp.361-370

[4]Michael R.Lyu, Optimal Allocation of Test Resources for Software Reliability Growth Modeling in Software Development [J], IEEE TRANSACTIONS ON RELIABILITY,VOL.51,NO.2,JUNE 2002

A Study for the Strategy to Distribute Testing Time of Component-Based Software

ZHU Yuan GAO Jianhua

(Department of Computer Science and Technology, Shanghai Normal University, Shanghai 200234)

Email: jhgao@shtu.edu.cn

Abstract: This paper presents three strategies to distribute testing time of Component-Based Software. According to the differential system parameters, these strategies analyze two types of limited condition, that is a problem with fixed target failure rate or a problem with fixed debugging time, and bring out the distributive strategy for each situation. These strategies are plausible in distributing testing time of component-based software.

Keywords: component, testing, distributive strategy