软件测试与软件维护

袁志录

航空航天部第六三一研究所

一九九一年九月

前　言

为了帮助软件工作者掌握软件的测试与维护技术及有关的软件工具，本文从论述软件测试、软件维护的重要性及基本概念出发，进而详细地介绍了软件测试和软件维护的基本方法与过程，并介绍了一组用于软件测试和软件维护的软件工具。由于时间仓促，编者水平有限，文中难免出现差错，热诚希望广大读者与软件界同行们批评指正。

编　者

一九九一年九月于西安

1 软件测试

1.1 软件测试的必要性和重要性

在软件开发的各个阶段中都包含了人们大量的创造性劳动，软件产品是人们复杂脑力劳动的成果。由于人们思维的局限性及软件开发过程的复杂性，决定了软件产品不可避免地会含有不少错误和缺点。另外，由于软件产品是一种抽象的产品，其错误和缺点就更不容易被发现。因此，软件测试在软件开发过程中是必不可少的，其重要程度无论怎样强调都不算过份。同时还必须看到，软件测试是一件非常复杂的工作，必须花足够的力量才能做好。

关于软件错误带来的危害，根据系统的大小及其重要程度也有所不同，严重的会导致整个系统失败，例如法国的一个气象卫星的坠毁；美国第一次发射金星火箭失败（由于FORTRAN程序中一个DO语句有错导致失败）；还有Mariner-I宇宙飞船发射的失败，这些都是由于软件的错误造成的。还有一个例子是说APOLLO宇航飞船发射中由于软件中一个逗号误改为一个句号导致控制失效。由上述例子可以看出，如果不及时查出和排除软件产品中潜在的错误所带来的危害程度是多么严重。

另外，据统计资料表明，在软件开发阶段，测试费用约占总经费的50% 左右。更有甚者，美国APOLLO登月计划中，大约80%的钱用于系统的测试中。 由此可见，软件测试是何等的重要。

1.2 软件测试的概念

1.2.1 软件测试

软件测试从广义来说不仅仅是程序的测试，而应该是贯彻整个软件开发过程的始终，深入到各个环节的一个不断发现软件中存在错误的过程。从统计资料表明，在软件的分析与设计阶段发生错误的比例占60% 左右，而编码阶段发生错误的比例占40%左右。

软件测试按传统思想只理解为对程序的测试。这是因为对分析和设计阶段的测试在方法和工具方面都还不够成熟，而且其发展也比较晚一些。因此，以下论述仍以程序测试为主要对象。

1.2.2 成功的测试

一个成功的测试应该是指发现了一个或多个以前未被发现的错误。而未发现任何错误的测试认为只是一次失败的测试。因此，对一个软件进行测试时，如果测试成功次数越多，说明软件中存在的问题越多，测试失败的次数越多可能说明软件中存在的问题越少。

1.2.3 完成测试的标准

在有关资料中提到如下两条标准可供参考：

·当所有测试情况都已执行而且不能再发现错误，也就是说测试最后是失败了，此时可以停止测试；但是，应该清醒的知道，此时只是没有再发现错误，而不是错误已经全部查出。

·按测试计划进度表，测试结束时间已到，则测试停止。

但在实际应用中，可推荐如下的方法：

·在指定的测试方法失败之后，即可停止测试；

·在规定时间内，对一定规模的模块、程序(例如1000行的程序)要查出一定数量的错误来(如查出10个错误)，可停止测试；

·画出测试时间与查出错误数量的关系图，如趋上升趋势，不可停止测试，如趋下降趋势，可停止测试；

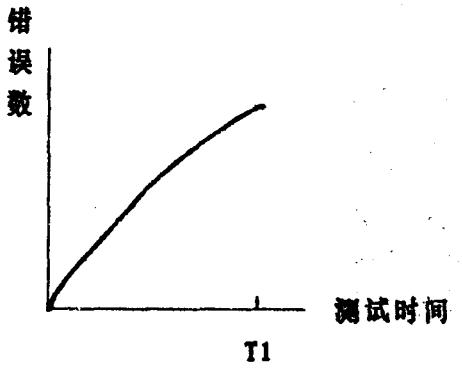

图 1

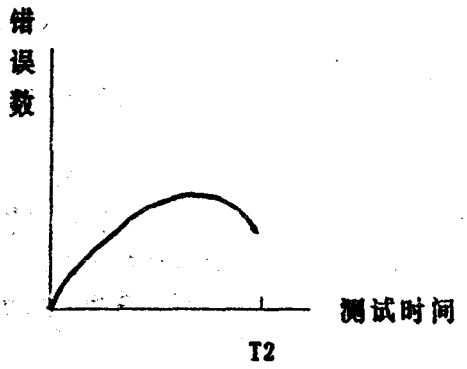

图 2

1.3 测试的目标

测试目标可分为三个方面：

·预防错误发生；

·通过系统的方法发现程序中的错误；

·应提供良好的错误诊断信息，以便改正错误。

显然，如果能预防发生错误，这将是最令人满意的。为此，应该把预防错误发生的工作渗透到整个软件开发的各个阶段和环节中。例如，在分析阶段、设计阶段如果能认真贯彻测试的思想，那么就能较早地预防和排除大部分错误。但是，要完全做到预防错误发生是不可能的。因此，人们还必须完成测试的第二个目标，也就是利用系统的方法检验程序，以便发现程序中的错误。但是，发现程序有错，并不一定知道错误的性质、原因和位置。因此，测试的第三个目标就是为解决这个问题提供必要的信息，以利于程序排错。

1.4 测试方法

测试方法一般分为静态分析和动态测试两种。动态测试又可分为结构测试(也称白盒测试法)和功能测试(也称黑盒测试法)两类。

1.4.1 程序的静态分析

静态分析不涉及程序的实际执行，而是以人工的方法对程序进行分析和测试。经验表明，大约30%～70%的逻辑设计错误和编码错误可以通过静态分析检查出来，所以静态分析方法是一种卓有成效的测试方法。

·桌前检查

通常由程序员反复阅读程序编码和流程图，对照模块功能说明、算法及语法规

定检测程序的语法和逻辑错误。也可以设计少量测试实例，由人来模拟计算机单步执行并考察执行的结果，从中发现错误。由于程序员熟悉程序的逻辑结构和自身的设计风格，可以节省很多检查时间。但由于人的盲目的自信心和对设计要求误解的延续性，其效果不甚理想。

·代码会审

代码会审是由一组人通过阅读、讨论和争议对程序进行静态分析的过程。会审小组由组长、若干名有关程序设计和测试的专家以及程序的作者组成。会审小组在充分阅读待审程序文本、控制流程图及有关要求、规范等文件基础上，召开程序审查会，程序作者逐句讲解程序的逻辑，并开展热烈的讨论甚至争议，以揭示程序错误的关键所在。实践证明，程序作者在讲解过程中能发现许多自己原来没有发现的错误，而讨论和争议则进一步促进了问题的暴露。例如，对某个局部性小问题修改方法的讨论，可能发现与之有牵连的涉及到模块的功能说明、模块间接口和系统总体结构的大问题，导致对需求定义的重定义，重设计和重验证，大大提高了软件的质量。

对会审小组每个成员应准备一份常见错误的清单，对照它进行检查，将会大大提高会审的效率。

常见错误大致有如下几种类型：

a.数据引用错误：如引用了未赋值的变量或数组；数组元素下标越界或非整数值；引用类型与说明不一致；……。

b.数据说明错误：如忘记对某些变量的显式说明；变量初值错误；变量的类型、长度、存贮类别说明不一致；……。

c.计算错误：如不同类型变量混合计算；赋值长度大于被赋值变量长度；表达式中间结果或最后结果上溢出或下溢出；……。

d.比较错误：如不同变量（如布尔量和整型量）的比较；比较运算符的误解（如将“至少”理解成“大于”）；运算符优先次序错误；……。

e.控制流错误：如多路分支(如CASE语句)标引变量值大于可能的分支数；循环不终止或循环不执行；程序包含死循环；循环次数多一或少一；复合语句括号不配对；……。

f.接口错误：如程序调用中形参与实参在个数、类型、范围、单位、顺序方面不匹配；模块间传递的参数在类型、范围、单位定义不一致；全程变量前后定义和属性不一致；……。

g.输入/输出错误：如文件的打开与关闭是否配对； 打开语句中属性理是否正确；是否正确判断和处理了文件结束标志；输入/输出语句与格式语句失配； 是否有输入/输出错误处理；输出文字信息拼写是否正确； 同时打开的文件数目是否超出限制；……。

h.其它错误：如变量定义了而未被引用；程序或模块功能错误；……。

会审检查可以避免程序员的盲目自信和对功能要求误解的延续性，检查的实效要比桌前检查更好。

·步行检查

步行检查是类似于代码会审的静态测试方法，不同的是预先为被测试程序选

择了一些有代表性的测试用例和数据，并要通过计算得到相应的预期结果。检查小组人员扮演计算机角色，让测试用例沿程序的逻辑运行一遍，随时记录程序的踪迹供讨论和分析使用。人们借助于测试用例的执行状态，对程序的逻辑和功能提出各种疑问和责难，开展热烈的讨论和争议。实践证明，通过讨论和争议所能发现的问题，要比测试用例直接发现的问题更多、更重要。

步行检查着重于从流程的角度来考察程序，借助于流程图来进行数据流和控制流的分析。通常以方框表示程序段，以有向边表示控制路径。利用测试用例的执行，可以分析程序的行为，对执行结果及执行路径进行检查。

静态分析主要用于检查逻辑设计和编码错误，但在理论和实践上还存在局限性，所以还必须进行动态测试。但是，人们正在努力研制各种静态分析自动化系统，尽可能使其与编译系统的语法分析、代码优化等相结合，以提高静态分析的速度和实效。

1.4.2 程序的动态测试

对程序进行动态测试的基本思想是，将程序视为函数，取函数定义域中每一个元素作为输入，实际运行程序，判断执行结果是否全部包括在函数的值域之内，用来检查程序的正确性、可靠性和有效性。应用穷举测试法是保证软件质量的唯一方法。但是它是一种不切合实际的方法。因为应用穷举法要作到完全测试，其测试的次数是非常之大的，甚至达到不可完成的地步。因此，我们必须将穷举法转化为可行的测试过程，在目前尚未找到可靠有效的算法之前，我们的讨论将集中在如何从函数定义域合理地、有效地选取具有代表性的子集，这些子集尽可能覆盖整个定义域而又能使测试付之实施。

·结构测试法

结构测试法也称白盒法或逻辑覆盖法。它是根据对软件内部逻辑结构的分析，选取测试数据集(即测试用例)。而测试数据集对程序逻辑覆盖的程度决定了测试完全性的程度。它是用来检查程序内部逻辑和执行路径的。

关于常用的覆盖标准有如下几种：

a.语句覆盖法：语句覆盖法是用足够多的测试用例，使程序的每一个语句至少执行一次，以增加发现错误的可能性；

b.判定覆盖法：判定覆盖法又称为分支覆盖法，是用足够多的测试用例，使程序中每个判别的每个分支至少执行一次。对于只有一个入口的程序，满足判定覆盖必然满足语句覆盖，因为所有语句都处于每个分支的路径上。对于多个入口的程序，除了判定覆盖外，还需要增加一些语句覆盖测试；

c.条件覆盖法：条件覆盖是用足够多的测试用例，使每一个判别条件的每种结果至少取到一次。例如，判别条件I>2，需要对I<2，I=2，I>2三种结果各执行一次。所以，条件覆盖是比判定覆盖更强的覆盖标准；

d.判定/条件覆盖：它是既可以使每个判别条件的一切可能结果至少取到一次，又可以使每个判别的每条支路至少通过一次。因此，由判定/条件覆盖产生的测试数据必然覆盖了语句覆盖、判定覆盖、条件覆盖，从而获得更好地实效；

e.多重条件覆盖：它是既使每个判定的每条支路至少通过一次，又使所有判别

条件可能结果的每种组合至少出现一次。显然，它是比前面几种覆盖更强的覆盖标准。

·功能测试法

功能测试法又称黑盒法。它根据对程序功能的分析，推演出由函数定义域中有代表性的元素组成测试数据集。这些数据应包括对程序是有效的和无效的输入；极端的、正常的和特殊的数据元素(可以有简单的，也可以有复杂的数据结构)。它是将程序看成是不透明的，不管内部结构及路径，只管它的整体功能。即只管输入与输出数据的转换是否按功能要求进行的。

功能测试通常包含有如下几种方法：

a.等价分类法：等价分类法根据程序的I/O特性，将程序的定义域划分为有限个等价区段，使得从每个区段内抽取的代表性数据进行的测试等价于该区段内任何数据的测试。对于每个输入条件存在着对程序有效输入的有效等价类，和对程序错误输入的无效等价类。例如，某整数X的值域由条件a<x<b限定，则所有(a+1)和(b-1)之间的整数构成了有效等价类；而任何小于a和大于b 的整数构成了二个无效等价类。等价类的划分取决于程序的功能要求、定义域和测试人员对问题的理解力和创造力。随着软件开发的进展，系统的总功能在各个阶段上自顶向下不断细化和求精，从而为相应级上的等价类划分和功能测试用例的选择提供了重要的指导信息。当确定了等价类之后，选取尽可能多的测试用例覆盖所有有效等价类，然后选取覆盖所有无效等价类用例。

b.边值分析法：这是一种根据I/O等价类边界上或紧靠边界的条件，选择测试用例的更有效的方法。例如，某整数X的值域为a<x<b，则有效等价类的边值为 a+1和b-1，而无效等价类的边值为a和b。另外，除了考察输入等价类外，还需要考察输出等价类来选取测试用例。例如，函数SIN(X)在特定区间上的输出边界值为1，0和-1，其相应的输入值也应成为测试的目标。如果能正确地选取边界值，可以检查出许多对随意选取的实例所不易检查出的错误。边值分析法产生的测试用例更全面、更具有代表性，发现错误的机率更高，但也更复杂。

c.因果图法：因果图法是采用逻辑图的形式来表达功能说明中输入条件的各种组合和输出的关系，帮助选择高效率的测试用例。由于因果图结构非常复杂，将因果图转换成判定表的过程也极为烦琐，使用起来不太方便，这里不作详细介绍了。

d.错误猜测法：错误猜测法是基于人们的经验和直觉，猜测可能发生的某些错误类型，并依此去选择测试用例以便发现这些错误。例如，零输入和对应零输出的输入值；开平方或对数运算中的负输入值；空的数据结构如空表、空栈、零矩阵；栈的上溢和下溢等，这些都是容易出错的情况。由于不存在错误猜测的规则，所以只能举例说明。

1.4.3 测试的策略与过程

·测试策略

各种数据测试用例的设计方法均各自提供了一组独特的、有益的测试用例，但没有一种方法能单独提供全部用例。也就是说任何一种测试方法都不是十全十美的，不可能通过一种方法将全部错误都查出来。只能说是某种方法易于查出某些错误，

某种方法可能在某些方面优越一些。因此，通常是将各种方法联合使用，较为理想的方法是首先实施功能测试(黑盒测试)，然后查出功能测试中没有执行的路径，最后通过结构测试(白盒测试)将未执行的路径补充测完。在这里我们提出下面的策略供大家参考：

a.如果功能说明中含有输入条件组合，则可以从因果图法开始；

b.然后用边值分析法对I/O等价类边值进行分析，产生一组补充的测试用例；

c.判别I/O的有效和无效等价类，进一步补充测试用例；

d.利用错误猜测法补充测试用例；

e.对照已有测试用例查看程序逻辑，再用足够多的测试用例覆盖各种覆盖条件。

数据测试是一种有价值的、重要的测试手段，其关键在于测试数据集的产生。但是测试数据量很大，构造复杂，而且无法获得覆盖所有错误的测试用例集，因此也就不能最终判定程序的正确性。所以测试过程的种种努力只是向着无错误程序更靠近一步。

·测试过程

测试过程一般按照“单元测试”、“集成测试”、“功能测试”、“系统测试”的顺序进行。

a.单元测试：单元测试是对构成系统的最小单位 —— 模块进行测试。单元测试的目的在于发现模块的逻辑、接口与该模块的外部规格（模块的功能、输入、输出以及对外部的影响）之间的差异。一般先进行模块功能测试，然后进行模块逻辑测试。单元测试还需要特殊的驱动模块和模拟模块，前者是向被测试模块传送测试数据和检查结果的，后者是被测试模块调用其它模块时，对该模块进行模拟的。

b.集成测试(亦称整体测试或统一测试)：在单元测试之后，如果要将模块集成为一个系统可能会出现新的问题，例如，跨越模块接口的数据是否会丢失；一个模块是否会破坏另一个模块的功能；子功能的组合是否能达到预期要求的主功能；全程数据结构是否有问题；以及单个模块的误差通过集成放大是否会达到不能接受的程度等。因此，必须在模块集成时进行集成测试，发现并消除模块联接中出现的问题，最终构成所要求的软件结构。

集成过程通常采用增殖方式集成，它是逐步将一个个模块联接在一起构成一个更大的模块，然后对新构成的大模块进行测试，直到最后将所有模块都集成在一起，构成一个软件系统，并对它进行测试。

关于增殖方式集成可分为以下三种方式：

·自顶向下增殖方式，这种方式是从根结点模块开始，沿控制层下移，逐步增殖和测试；

·自底向上增殖方式：这种方式从最底层模块开始，沿控制层逐层上移，逐步增殖和测试；

·混合增殖方式：实际测试中很少采用“纯粹”自顶向下或自底向上的增殖方式，而是将二种增殖方式混合使用。

c.功能测试：功能测试的目的在于发现程序与功能规格说明书(规定外部功能的规格说明书)的不一致之处。此处功能测试主要指将所有模块集成在一起进行的功能测试。在有的资料上将此类测试合并到集成测试中。有的资料上将这一步骤称

为有效性测试。

d.系统测试：系统测试是把软件作为整个计算机系统的一个元素，与硬件、外设等其它的系统元素结合在一起，对该计算机系统进行一系列的整体测试和功能测试。其目的是为了发现程序与产品目标的不一致之处。系统测试以基本规格说明书（规定产品目标的规格书）为依据，进行以下几方面的测试：

·可靠性测试——如果程序的目标中有可靠性描述，按要求进行测试（如平均无故障时间要求）；

·可用性测试——如出错信息、响应方式、平均排故时间等测试；

·性能测试——如响应时间、吞吐量的测试；

·强度测试——如对多用户系统，对用户上限情况进行测试；

·容量测试——如对大量的数据处理、大程序的运行等的测试；

·存贮器测试——如缓冲区大小的测试；

·安全性测试——如对程序保密性的测试；

·配置测试——如对最大、最小硬件配置环境的测试；

·兼容性或转换测试——检查兼容性及系统之间的可转换性；

·恢复测试——如出错后恢复功能的测试；

·可维护性测试——有关维护工具、维护过程方面的测试；

·资料测试——测试用户资料的正确性、准确性。

除上述测试之外，还有其它方面的测试，此处不再列举。上述所列情况供大家在使用中参考。

1.5 测试工具

由于测试对软件开发是至关重要的，但人工测试又特别容易出错，故测试过程的自动化是发展的方向。目前国内外已有许多自动测试工具被用来对程序进行静态分析和动态测试。

下面以VAX SET 为例，介绍一下可用于软件测试的一些工具。VAX set 是VAX/VMS操作系统下运行的一组软件工具，可用于支持软件的开发及维护阶段的各项工作。其中，SCA主要用于静态分析，也以用于帮助设计动态测试用例及分析错误影响范围；PCA及DTM主要用于动态测试。

1.5.1 源代码分析程序(SCA)

1.5.1.1 SCA的基本思想

SCA是一个静态分析工具。首先，它需要依靠编译程序来建立各个程序中各种信息之间的关系文件。然后，SCA依据这些关系文件进行查询和分析，使用户可以更容易地理解一个应用系统的组织结构和更快地确定系统中错误的位置及其影响的范围。SCA的查询和分析功能是对整个应用系统进行的，在软件的开发和维护阶段，SCA都是一个强有力的工具。

1.5.1.2 SCA的组织结构

SCA由两部分组成。第一部分是交叉查询功能。交叉查询可以提供应用系统中

任意一个符号或一类符号的定义位置及其各种引用的位置。由此可以更好地了解对一个符号作修改时所产生的影响和更快地确定某一个错误所影响的范围。这个功能在理解一个程序和改正程序错误的过程中都是很有用的。第二部分是静态分析功能。静态分析主要是提供程序之间的调用关系图和检查程序间调用的一致性。调用关系图可以显示一个程序调用了哪些程序以及它被哪些程序调用了，这对于理解一个应用系统的结构是极为有用的。程序调用一致性检查主要是检查程序调用中传递参数的个数、类型、顺序以及返回值类型等方面的错误，这些错误在编译时一般是不检查的。

1.5.1.3 SCA的作用

SCA是一个交互的、多语言的源代码交互查询和静态分析工具。它提供了源程序编码的详细信息，它可以用于理解一个大型的软件系统。

1.5.1.4 SCA的功能

a.定位名字及名字的出现(指各种用法)。

b.询问一个指定的名字组或部分名字组，使用通配字符是允许的。

c.对指定特性的一个询问(如程序名、变量名或源文件名)。

d.对特定出现(如一个符号的初始声明、一个符号的读写出现、或一个文件的出现)的询问。

e.显示一个程序的调用关系。

f.分析程序调用的一致性。

1.5.1.5 SCA的使用方法

(1) SCA分析数据文件的建立

为了生成详细的源文件分析数据并加载到一个SCA库中，SCA依赖于VMS 各个编译程序。

如下的DCL命令行可产生一个分析数据文件

```
$ Language/ANALYSIS_DATA[=file-spec][/…]source-file[,…]
```

其中，/ANALYSIS_DATA要求指定的编译程序生成一个带有缺省类型.ANA 的源程序信息的输出文件。除非你指定了它的目录，这个.ANA文件出现在你的当前缺省的目录中。若未指定file-spec，则分析数据文件与源文件同名。language是指各语言的编译命令。

(2) 建立SCA环境的其本步骤：

a.在当前目录下建立一个库目录

例如 $ CREATE/DIR [.SCALIB]

b.建立一个SCA库

将前面建立的库目录建成一个SCA库

例如 $ SCA CREATE LIBRARY [.SCALIB]

c.应用编译程序建立一个分析数据文件(如对C语言程序)

例如 $ CC/ANALYSIS_DATA C1.C, C2.C

d.加载分析数据文件到SCA库中

例如 $SCA LOAD C1.ANA,C2.ANA

e.激活SCA库并询问信息

例如 $SCA SET LIBRARY [.SCALIB]

$SCA FIND *PRINTF

(3) 请求SCA

SCA可以随着LSE被请求作为一个集成工具，也可以在DCL级作为一个单独的工具被请求。

作为一个集成工具，LSE支持一个扩充的命令语言，它包含了所有的SCA 命令和定向命令。SCA命令的发出是用同LSE命令一样的方法，其格式如下：

LSE > Command [Parameter][/qualifier …]

为了在DCL级单独请求SCA，命令格式如下：

$SCA command [parameter][/qualifier …]

或者

$SCA

SCA > command [parameter][/qualifier …]

在SCA >提示符之后可以打入任意一个SCA命令，并通过按RETURN或ENTER键来执行它。EXIT可以结束一个SCA子系统会话，并返回到DCL级。也可以按CTRL/Z结束一个子系统会话。

(4) 交叉查询的概念

SCA的交叉查询特点允许应用一个FINT命令询问一个SCA库，查找与源程序有关的符号或文件信息。

可以询问SCA库查找关于name、item、occurrence的信息。其含义分别解释如下：

a.一个name是一系列字符，它标识了在一个源程序中的一个符号或一个文件。不管级别的情况。

b.一个item是在一个源程序中一个符号(如一个变量、常数、标识符、过程等）或一个文件的状态。带有级别的信息。

c.一个occurrence是在一个源程序中一个符号或一个文件的用法。

对于一个给定的符号名，在SCA库中可能有多种级别（如一个过程名或一个变量名），利用出现的不同类型可以进一步定义这个符号的用法(如声明或引用)。

例如假定名字ABC有两种有关的符号级别，一个是过程名和一个是变量名，每一个符号有如下的出现类型：

——过程ABC有一个基本声明、多个外部声明、和对过程的调用。

——变量名ABC有一个声明、读引用(在那里变量得到值）和写引用（在那里变量的值被修改)。

(5) 定向概念

当在LSE内部第一次发出一个FIND或者一个CHECK命令时，开始了一个询问会话。在此期间，询问和源程序的缓冲区同时被建立，并且SCA的询问命令和LSE的定向命令结合在一起允许作如下事情：

a.逐步穿过一个询问缓冲区的显示；

b.从一个询问缓冲区存取一个源程序缓冲区；

c.从一个源程序缓冲区发出一个询问；

d.操作多个询问会话。

(6) 静态分析的概念

SCA静态分析的特点允许检查程序结构的一致性。这些命令可以在LSE 内部使用，或者在DCL级使用。命令如下：

```
CHECK  CALLS
VIEW  CALL_TREE
```

—— 显示程序调用关系

命令VIEW CALL_TREE显示程序调用关系。不像FIND和CHECK命令，这个命令不能用在一个询问会话的上下文中。命令格式如下：

```
VIEW  CALL_TREE [/qualifier …] routine-name-expr[,…]
```

这个命令可以显示库中所有那些程序，它们调用了指定的程序，或者被指定的程序调用。

—— 检查程序调用的一致性

CHECK CALLS命令检查指定程序关于这个库内所有程序接口的一致性。命令格式如下：

```
CHECK  CALLS  [/quaifier …] routine-name-expr[, …]
```

该命令发出，可以得到一些诊断信息，显示在屏幕上方窗口，下部窗口可显示相应的源程序。

例如，

```
LSE > CHECK CALLS EXPAND_STRING
LSE > GOTO SOURCE
LSE > NEXT OCCURRENCE
LSE > GOTO SOURCE
```

1.5.2 性能及覆盖分析程序(PCA)

1.5.2.1 PCA的基本思想

由于软件产品具有一定的抽象性，因此，仅仅根据一个软件执行的结果并不能完全显示出它的动态性能。但是，有时候人们往往还需要对软件的动态性能有一个更深入细致的了解。例如，当希望一个软件能够运行得更快时，就必须了解该软件中各个程序单元运行时占用的机器时间，从中找出占用机器时间最多的程序单元，然后加以改造和优化，即可达到加快运行速度的目的。又如当对一个软件进行结构测试时，就应当知道对某一组指定的测试数据，该软件的执行路径是什么。PCA 就是一个分析动态性能的工具。它较好地解决了包括上述问题在内的动态性能分析问题。

1.5.2.2 PCA的组织结构

PCA由两部分组成。第一部分是收集程序。它对一个正在运行的用户程序收集关于动态性能和测试范围的数据，并把这些数据记录在一个性能数据文件中。这些

数据包括程序计数器抽样数据、页面错误数据、系统服务程序调用数据、输入/输出调用数据、精确的执行计数数据、测试路径范围数据。第二部分是分析程序。它负责读入性能数据文件，并进行分析和处理这些数据，从而产生性能和路径范围的方框图和表格。根据这些方框图或表格就可以了解关于动态性能及执行路径范围方面的信息。

1.5.2.3 PCA的作用

PCA可用来分析应用系统在运行期间的性能。它是一个动态特性分析工具。根据系统的动态性能，可以更有效地对程序进行改造和优化。根据覆盖路径分析，可以改进系统结构的性能，并可以帮助设计更好的测试实例，获得更好的测试结果。

1.5.2.4 PCA的功能

a.确定应用系统在执行中各个程序单元占用的机器时间和其它性能问题。应用这些信息，可以修改模块使它运行的更快。

b.利用给定的一组测试数据测试应用系统哪些部分是执行了或者是没有执行，来分析测试的范围。即给出测试路径。

1.5.2.5 PCA的使用方法

(1) 性能和覆盖分析程序由两个部分组成

a.收集程序

这个部分首先运行，它对一个运行的程序收集性能和测试范围数据，并把这些数据记录在一个性能数据文件上。

b.分析程序

这个部分读性能数据文件并处理这些数据，从而产生性能和测试范围的方框图和表。

(2) 使用收集程序

要想使用收集程序，需要按以下方法编译、连接和运行应用程序，假定程序为PRIMES.FOR，首先应用

```
$ FORTRAN/DEBUG  PRIMES.FOR
```

编译源程序，其中/DEBUG限定词使编译程序在目标模块中建立一个调试符号表(DST)，这个目标模块中包含了为了说明在程序中的位置所需要的所有符号和行号信息。然后应用

```
$ LINK/DEBUG=SYS $ LIBRARY : PCA $ OBJ.OBJ  PRIMES.OBJ
```

连接程序，其中PCA$OBJ.OBJ是收集程序的启动模块，然后应用

```
$ RUN  PRIMES.EXE
```

运行程序，并开始性能和范围数据收集。

当程序开始运行时，由收集程序进行控制，显示一个标题信息，并提示输入收集程序的命令，形式如下：

```
PCAC >
```

它是收集程序的提示符，表示可在该提示符之后打入收集程序的命令。

关于数据收集，一般可按如下步骤进行，首先应用

```
PCAC > SET  DATAFILE  filename
```

来指定一个包含所收集的数据的性能数据文件名，如果不使用该命令指定性能数据文件名，按照缺省值取程序名作为性能数据文件名，其类型为.PCA，其次，打入命令来说明要收集什么样的性能和范围数据，关于指定收集数据类型的命令包括：

```
SET  PC_SAMPLING
SET  PAGE_FAULTS
SET  SERVICES
SET  IO_SERVICES
SET  COUNTERS
SET  COVERAGE
SET  STACK_PCS
SET  EVENT
```

关于上述命令的含义及用法均可通过

```
PCAC > HELP
```

得到联机帮助。

例如为了收集程序计数器(PC)抽样数据应打入如下命令

```
PCAC > SET  PC_SAMPLING
```

如果没有打入任意一个上述的SET命令，当你打入运行命令GO时，按缺省值将得到程序计数器(PC)抽样数据。也可以使用SHOW ALL命令显示已经用过哪些收集数据命令，同时也显示性能数据文件的名字及语言的设置。最后，应该应用GO命令告诉收集程序去启动程序，如

```
PCAC > GO
```

此时，按照当前的数据收集设置去收集数据，并将这些数据记录到性能数据文件中，之后关闭这个数据文件。

在发出GO命令之后，收集程序不返回到收集程序的提示符，而是程序运行到结束并返回一个DCL级的提示符($)。

在SET DATAFILE、SET PC_SAMPLING及GO命令之中，只有GO是必须的，其余二者均有缺省值可以代替。

(3) 使用分析程序

分析程序读由收集程序产生的性能数据文件，并应用这些数据产生方框图、表格及其它报告，帮助评价程序的性能。

a.请求分析程序

为了请求分析程序，发出一个PCA命令并指定由收集程序建立的性能数据文件的名字。例如，为了应用由前面收集程序运行产生的性能数据文件PRIMES.PCA，打入命令

```
$ PCA  PRIMES
  VAX performance and Coverage Analyzer Version 1.1-1
PCAA >
```

其中，提示符PCAA >告诉你分析程序已准备好接收分析程序的命令。

b.建立方框图和表

用来以图表方式描述程序性能和覆盖特性的基本命令是 PLOT 和 TABULATE。PLOT命令以方框图方式提供信息，而命令TABULATE以数字列表方式显示相同的数据。两个命令接收同样的限定词和参数，并应用相同的数据显示按排。

例如，为了在程序中按照每一个模块将程序计数器抽样数据画图，PLOT命令应该使用的节点说明是PROGRAM_ADDRESS BY MODULE，整个命令句子是

```
PCAA > PLOT/PC_SAMPLING PROGRAM_ADDRESS BY MODULE
```

可能在表与方框图之中用户更喜欢一个表，应用TABULATE代替PLOT并应用同样的限定词和参数，如

```
PCAA > TABULATE/PC_SAMPLING PROGRAM_ADDRESS BY MODULE
```

即可将前面以方框图显示的数据用表的形式显示出来。(输出表省略)

c.打印和保存分析程序的输出

PRINT和FILE命令对最后打入的PLOT和TABULATE命令起作用。 应用PRINT命令得到当前使用的方框图或表的一份硬拷贝。 分析程序仅允许PRINT 命令带有/NOTIFY限定词。如可用命令

```
PCAA > PRINT
```

应用FILE命令可放置当前全部画图或列表的文本，包括摘要页在内，到一个指定的文件中，例如

```
PCAA > FILE file-name
```

为了积累若干个画图和表到一个单个的文本文件中， 可应用/APPEND限定词到FILE命令上去。FILE命令不接受其它限定词。

d.结束分析程序

利用打入EXIT或按CTRL/Z可以结束分析程序，返回到DCL级($)，

```
PCAA > EXIT
$
```

e.得到联机帮助

```
PCAA > HELP
```

可以显示分析程序的命令。

1.5.3 测试管理程序(DTM)

1.5.3.1 DTM的基本思想

在软件开发及维护过程中，经常进行软件测试工作。 DTM为帮助用户建立一个适当的测试环境并使测试在控制条件下进行提供了一个工具，测试和评价测试结果均可用自动化方法完成。 DTM用了回归测试的概念。首先为一个测试建立一个期望结果。然后，在运行这个测试时，将当前的测试结果与期望结果进行比较。如果发现当前结果与期望结果不一致，则表示软件可能有错误。一般来说，在一个软件开发过程中，应该使以前的测试结果能够重现，如果不能重现，则表明软件的修改可能存在错误。如果能证实错误存在，则正在测试的软件被称为出现了“回归”(倒退)，此时应该停止当前的开发，设法改正存在的错误。