# 面向对象软件测试引擎的设计和实现*)

马雪英[1,2] 姚 砺[2] 叶澄清[2]

(浙江财经学院信息管理系 杭州310012)[1] (浙江大学计算机科学系 杭州310027)[2]

**摘 要** 为了减少软件测试的工作量,提高软件测试的效率,非常有必要开发计算机辅助的软件测试工具。本文基于已经实现的分别面向C/C++和Visual Basic语言的计算机辅助软件测试工具为基础,介绍了面向对象软件测试引擎的设计和实现。该引擎以中间数据库为核心,使用语言分析器对源程序进行基于块的划分,然后完成代码插装和编译连接,为自动测试模块提供可执行的经过代码插装的程序而不是源代码,从而提高了测试工具的通用性和重用性。

**关键词** 计算机辅助测试,测试引擎,代码插装,中间数据库,面向对象软件测试

## The Design and Implementation of the Object-Oriented Software-Testing Engine

MA Xue-Ying[1,2] YAO Li[2] YE Cheng-Qing[2]

(Department of Information Management, Zhejiang Institute of Finance & Economics, Hangzhou 310012)[1]

(Department of Computer Science, Zhejiang University, Hangzhou 310027)[2]

**Abstract** It is necessary to develop a computer-aided software testing tool to enhance testing efficiency and reduce testing cost. On the basis of two developed testing tools for the Visual Basic and C/C++ language respectively, this paper introduces the design and implementation of the object-oriented software-testing engine. The programs are divided into a sequence of blocks and then instrumented and compiled in this engine by the program analyzer, the test data is saved in the medium database that is the kernel of the engine. The testing engine, which acts as an agency, associates the testing automation module with instrumented executable program rather than the source code and makes the test tool can be reused conveniently by modifying the program analyzer only.

**Keywords** Computer-aided software test, Testing engine, Program instrumentation, Medium database, Object-oriented software-testing

## 1 引言

以软件测试为中心的软件质量保障技术在软件生产实践中得到了迅速的发展,软件测试已成为软件生产中必不可少的质量保障手段。

随着软件工程技术的发展,软件设计规模的增大,软件测试在软件开发过程中的作用显得尤为重要,其复杂性也越来越高。开发计算机辅助软件测试工具的目的就在于将复杂的测试工作自动化或半自动化,以提高测试效率,降低软件开发成本。我们开发的面向程序设计语言(如C、C++、Visual Basic)的软件测试自动化工具,以结构测试为主,服务的对象是了解软件内部编码逻辑的人员,或者说是软件的开发人员,能在单元测试和集成测试的级别上工作。主要的功能有程序结构图的自动生成、控制流图的自动生成和控制流分析自动化、覆盖分析自动化、质量度量自动化、动态跟踪自动化和测试执行自动化。其中,程序结构图及控制流图的自动生成和质量度量自动化通过静态分析实现;覆盖分析自动化和动态跟踪自动化通过动态分析实现;测试执行自动化用来支持回归测试。图1给出了其系统结构。

从图1可以看出,该工具主要分成两大部分:引擎(虚线框内部分)和自动化测试模块。其中引擎是整个系统的核心,是实现一系列自动化的基础,用以支撑整个软件测试工具。为了有效地分析程序的控制流,引擎对传统的流图模型施以约束,使用一种新的、有效的、基于块的程序划分机制,得出相应的流图模型,且这种流图模型对于不同的上下文无关语言显示了惊人的一致[1]。并且在这种约束的基础上,延伸了传统的以程序为基础的测试充分性度量准则,又进一步给出基于这种约束的测试复杂性的质量度量[2]。而本文主要介绍引擎的核心中间数据库的设计,以及对基于块的划分后的程序的插装。虽然动态数据库是在动态分析过程中产生,但是作为中间数据库的一部分,所以也在文中加以介绍。

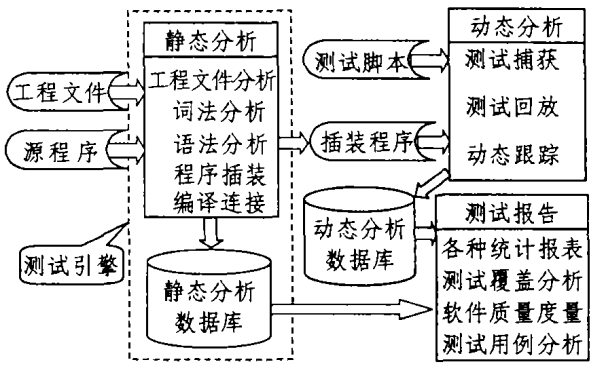


图1 测试工具系统结构

## 2 引擎工作流程

引擎是整个系统的核心,是实现一系列自动化生成工作的基础,用来支撑整个软件测试环境。其工作流程如图2所示。

从图中可以看出,首先由源程序分析器对面向对象源程

*)本课题得到国家自然科学基金资助(项目编号60073027)。马雪英 博士研究生,主要研究方向软件测试,数据库设计,网络安全。姚 砺 博士,主要研究方向为软件测试,网络安全。叶澄清 教授,博士生导师,主要研究方向为软件测试、高性能及智能机系统、多机并行处理系统、多媒体计算机技术。

序进行词法、语法分析，它相当于一个编译程序的前端，但它的语义动作是为了实现测试的目的而执行的静态数据的获取和动态程序的插装，而不象一般高级语言编译程序的语义动作，是由计算机执行目标代码来完成算法所表述的功能。分析器在语法分析的同时，对源程序作静态分析和程序插装，再把插装后的程序编译连接成可执行代码。静态分析就是从源程序中提取必要的数据（比如：函数名、类名、行号等信息），同时对源程序的控制流进行分析，为源程序进行基于块的程序划分，建立基于块的流图模型，然后把得到的数据都保存到一定格式的专为测试设计的静态分析数据库中，这些数据文件用来实现图表自动生成和质量度量自动化。程序插装就是要往源程序的特定位置中插入我们自己定义的代码，当插装后的可执行程序运行时就可以获取动态数据，并生成动态分析数据库。插装后的可执行程序为实现覆盖分析自动化和动态跟踪自动化作好了准备。

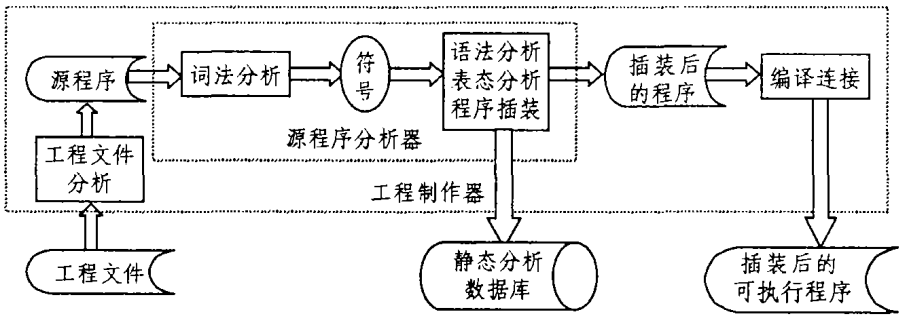


图2　引擎的工作流程

基于块的划分机制以及基于块的流图模型，从根本上保证了对多种上下文无关语言的测试语义的一致性。基于这种模型的优越性，我们扩展了传统的以程序为基础的软件测试充分性度量准则和复杂度量准则。

静态分析按测试的要求生成静态数据库文件和插装后的源代码，并为后续的动态分析过程提供支持。在整个测试环境中，只有静态分析模块是直接与源程序的语言相关的。而动态分析与测试报告模块则与源程序无关，它们之间的唯一联系便是通过中间数据库文件（静态数据库文件和动态数据库文件）。静态分析模块实际上起到了中介的作用，它使测试环境的动态分析过程只与数据文件相关，而与源程序的语言无关。

## 3　中间数据库设计

尽管面向对象语言的种类繁多，但其采用的面向对象技术都是一致的，如继承、封装、多态等，差别只在于实现的程度。如，C++实现了多继承、覆盖、重载等 OO 概念，而 Java 则实现了单继承、覆盖、重载等 OO 概念。因此，为了实现设计与代码的复用，我们根据软件结构理论的观点，在设计时采用了以中间数据库为核心的软件结构。

中间数据库是软件测试环境的核心，我们设计中间数据库的指导思想是“存储方便、数据详尽、易于扩充”。

### 3.1　静态数据库结构

静态数据库的信息主要用于软件质量的度量和对源代码的静态测试分析。

我们为每一个源程序文件创建一个静态数据文件，用来记录该源程序的信息。为避免同一字符串重复保存，便于管理，我们将所有字符串都统一保存在一起，用 Hash 法管理。所以在静态数据库文件中本该保存字符串的域保存的都不是字符串本身，而是用来检索该字符串的索引。

静态数据库的内容包括：源文件的结构信息、类的定义信息、方法的定义信息、源文件的块划分信息。静态数据库的结构参见图3。

（1）*源文件的结构信息*　为每一个源文件创建一个文件结构体（参见图3中的 file_node0结构），存放诸如：源文件名、动态数据库的存放路径、源文件引用的文件（适用 C/C++这类有包含关系的语言）、源文件中定义的方法、源文件中定义的类、源文件中定义的静态和非静态全局变量、源文件对应的块划分序列等信息，所有源文件的文件结构体用链表依次串成一条链。

（2）*类的定义信息*　为每一个类声明建立一个类结构体（参见图3中的 class_def0结构），存放包括类名、定义该类的源文件、定义该类的方法、该类的外层类、该类的基类、该类的友元类、该类的私有成员变量、该类的公有成员变量等等关于类的信息，同样所有的类结构体用链表依次串成一条链。

（3）*方法的定义信息*　为每一个方法定义创建一个方法结构体（参见图3中的 func_def0结构），存放包括方法名、方法的返回类型、方法的类型、定义该方法的源文件、该方法的第一个程序块在动态文件中的序号、方法体的第一个程序块、方法体的最后一个程序块等等有关方法的信息，所有方法的结构体也用链表依次串成一条链。“方法的类型”包括以下八类及它们的相互组合：宏函数、非类成员函数、类成员函数、虚函数、内联（inline）函数、重载（overloading）函数、静态（static）函数和纯虚函数等。

另外，有些数据域要填入函数名，但是这些函数名并不一定是源程序中实际的函数名称，而是对其加以变化，避免命名重复冲突。比如对于重载的两个函数：int *foo*()和 int *foo*(int *n*)，如果都以这两个函数的实际名字来表示，那就会发生重复，所以要加以变化，比如分别变为：*foo_v* 和 *foo_n* 等。

（4）*源文件的块划分信息*　对面向对象软件的控制流结构重新进行了基于块（Block）的划分，源程序划分为统一的段和节点的排列，并将这种划分序列信息存入静态数据库文件，供后续的动态分析过程使用。该信息包括：块的类型、块所对应的记录点的序号、块在源程序里的起始行号等信息。源文件中的每个块都对应一个块结构体（参见图3中的 block_node0结构），每个源文件的所有块用链表依次串成一条链，对应于整个程序。

### 3.2　动态数据库的内容

动态数据库文件用于统计测试覆盖率、分析执行路径/控制流、统计模块执行的时间和执行频率、分析测试用例效率、测试用例与代码的响应、最小化测试用例生成以及其他动态性能分析等。

因为在动态分析中，对程序执行过程的跟踪，是基于对源程序的结构经重新划分而得到的段和节点的执行顺序，而段和节点的顺序只与源程序的逻辑结构有关，而与源程序使用的具体语言无关。所以，动态数据库文件的内容与具体的面向对象语言无关。

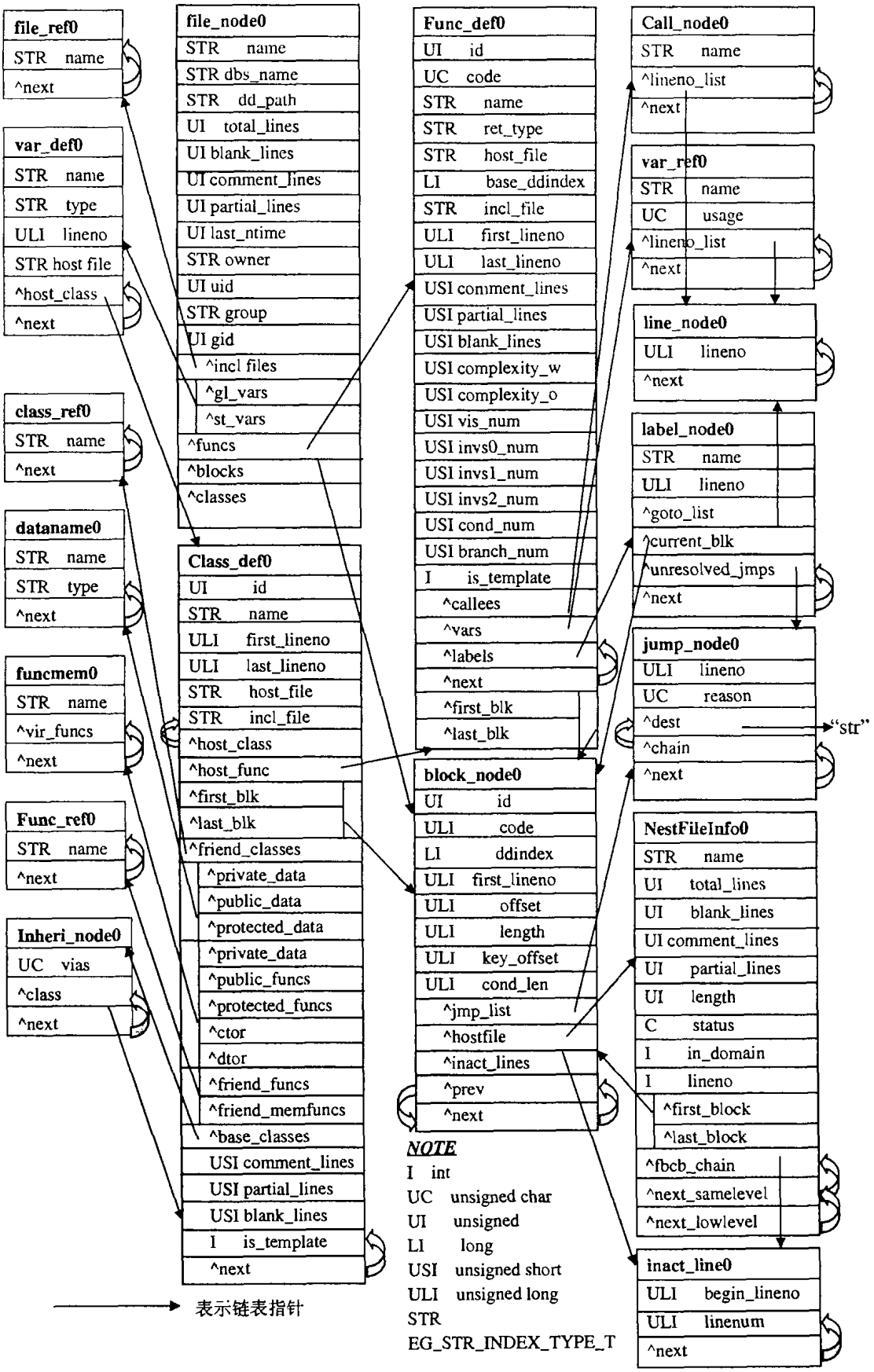


图3 静态数据库结构示意图

动态数据库文件只收集与被测软件运行时相关的信息，如：程序中各个段、条件等在每次测试中的执行次数；程序中的每个条件、判断在测试执行过程中的取值(true/false)；每个测试用例测试了哪些段、方法、类等。

每个被测源文件都对应四种动态数据库文件：DDH 文件、DDC 文件、DD 文件和 DS 文件。下面具体介绍这些动态数据库文件的组成结构。

(1)DDH 文件的结构 DDH 文件保存每次运行被测程序时，被测程序的记录点个数，本次运行被测程序的时间，以及各个记录点对应的程序块是否运行过和执行时间的信息。

(2)DDC 文件的结构　DDC 文件用于依次记录源程序中每一个条件/判断在被测程序运行过程中的取值情况，包括源程序中包含的条件(Condition)的数目，源程序中包含的判断(Decision)的数目，以及源程序中每一个条件在被测程序运行过程中的取值情况。对于"$a$ && ($b$||$c$)"这样一个表达式，我们把整个表达式"$a$ && ($b$||$c$)"叫做一个判断(Decision)，其中的每一个子表达式叫做一个条件(Condition)。

(3)DD 文件的结构　DD 文件用来记录插装程序运行时各记录点的累计运行次数、最后一次运行次数，以及时间等其他信息。以 binary bit map 格式存储。

(4)DS 文件的结构　DS 文件记录用来帮助显示测试覆盖率的字符串。每个字符串之间用换行符(0x0A，'\n')间隔。DS 中保存的字符串是在语法分析过程中根据作图需要生成的，而不是直接从源文件中拷贝的。例如，对于类声明语句，对应的字符串是："class 关键字"+"类名"+"说明继承情况的字符串"。

## 4　程序插装

我们统计动态数据的原理是：对源程序进行插装(即插入一段代码或函数)，利用插装代码收集需要的程序运行时的动态统计数据。语法分析器识别分析的源程序的语法，在我们定义的插桩点上插入代码，并生成各种动态数据库文件：DD 文件、DDH 文件、DDC 文件和 DS 文件(见3.2节介绍)。

### 4.1　基于动态链接库的插装

传统的插装技术是把与插装操作有关的函数(包括对相关文件的操作、插入的代码等)做成一个 Lib，然后在源程序的必要位置插入调用这些函数的代码，在编译连接插桩后的源程序时，再把这个 Lib 也连接进去。为了将插装代码所收集的动态数据从内存更新到数据文件中，必须在被测程序所有的结束语句(如 C 的 exit 语句)前插入输出函数，从而在源程序正常运行结束之前，把内存的动态数据更新到数据文件中。这种方法的一个致命的弱点，就是当插桩后的被测程序运行崩溃或用户强行中断时，无法将插桩代码所收集的动态数据从内存更新到动态数据库中。

为此，我们采用了动态链接库技术来实现动态数据的收集和更新工作，即将与动态数据库操作有关的函数做成一个 dll 文件，然后在 VB 源程序的必要位置插入调用 dll 中相应函数的代码。插装代码收集到的动态数据也保存在动态链接库的地址空间，而不是被测程序的地址空间。

这样，一方面，因为动态链接库与调用它的应用程序分离，被测程序即使崩溃也不会破坏已收集到的动态数据，从而我们可以准确地定位程序崩溃时的出错位置。另一方面，在 Windows 操作系统中，调用动态连接库的程序或线程退出运行时(包括非正常退出的情况，如用户按 ALT+F4强行中断程序的运行，或被测程序崩溃)，会释放对动态连接库的引用，在动态连接库中可以获知这一事件，进行退出时用户指定的操作。

我们设计的动态链接库程序的作用是：初始化节点、构造链表，形成被测程序运行所需要的数据结构；首次调用动态连接库时，挂上被测程序退出时要执行的函数；检查用来保存统计数据的文件存在与否；将内存中的统计数据写入文件。

### 4.2　插装工作流程

(1)分析 VB 工程文件(*.vbp)，获得所有的 VB 源文件；

(2)对于每一个 VB 源程序文件

①分析源程序结构，生成 DD 文件、DDH 文件、DDC 文件和 DS 文件；

②分析源程序结构，生成插装信息链表 InstrChain；

插装信息链表的数据结构如下：

```
struct InsPoint {
  int SrcPos;              // 代码插入的位置
  int Style;               // 插桩代码的类型
  int Rno;                 // 插桩代码统计的覆盖测试记录点
  int Cno;                 // 插桩代码统计的条件测试记录点
  int Lno;                 // 伪局部变量后缀
  struct InsPoint *Next;
  };
```

③替换插装信息链表 InstrChain 中的伪局部变量后缀；

比较重要的特殊字符有：@R，代表程序的逻辑计数器的位置，根据插装点在源程序的位置依次递增排列，每个插装点中的@R 值是一样的；@C，代表程序中判断的序号，每个判断的@C 值是相同的。根据插装点在源程序的位置依次递增排列；@L，代表伪局部变量的后缀，在插装程序的运行中定义的伪局部变量用来记忆程序的状态。

④根据插装信息链表插桩源程序。

(3)添加公用模块_isa.bas，用于声明全局变量和对 DLL 中函数的引用；

(4)修改工程文件，将_isa.bas 添加到工程文件中。

**结论**　以中间数据库为核心的设计测试引擎，有以下几个优点：

满足了 OO 语言不断发展的需要。基于流图模型分析的中间数据库设计，可以方便地使测试环境支持多种面向对象上下文无关语言。中间数据库的结构全面而又稳定，我们只需要重新改写静态分析的词法分析和语法分析部分代码，甚至连静态数据文件的结构也不需作任何改动，就能支持新的面向对象语言程序的测试，后续的动态分析和测试报告模块也无需改动。

易于系统扩充测试度量标准。随着面向对象技术的发展，会有新的测试度量标准问世，可能会需要增加一些新的度量。中间数据库将语言分析过程与度量计算过程独立开来，一般情况下，如果静态分析和动态分析提供的信息能满足测试度量的需要，那只需修改测试报告模块中相关的度量计算和显示代码即可。

实现增量测试。由于利用数据库分别存放测试软件各个模块的分析结果，这样当用户修改或添加某个模块时，便不需要对整个软件重新进行测试，我们只需对修改或添加的模块进行分析，并根据分析结果，或修改该模块对应的分数据库的内容，或插入该模块对应的分数据库，这样可以大大提高系统的工作效率。

另外，本文研究的插装技术不仅能定位程序运行时的出错位置，而且对被测系统的运行性能影响小。我们曾做过一个试验，以一段播放声音文件的程序作为被测系统，用 Rational 公司的 Pure 系列工具测试时，声音时断时续，而我们的产品测试时，声音是连续的。

## 参 考 文 献


1　杨建军，陈卫东，叶澄清，潘云鹤. 面向上下文无关语言的测试工具的设计和实现. 计算机研究和发展，2000，37(11)：1375～1382

2　马雪英，陈卫东，杨建军，叶澄清. 基于块的测试充分性度量准则及其测试复杂性度量. 计算机科学，2000，29(5)：141～143

3　孙霆. 软件测试工具的研究和实现：[硕士学位论文]. 杭州：浙江大学，1999

4　Shaw M，Garlan D. Software Architecture. Englewood Cliffs，NJ：Prentice-Hall，1996

5　姚砺. 面向对象软件测试研究：[博士学位论文]. 杭州：浙江大学，2002

6　郑人杰. 计算机软件测试技术[M]. 北京：清华大学出版社，1992

7　尤晋元，史美林. Windows 操作系统原理. 北京：机械工业出版社，2001