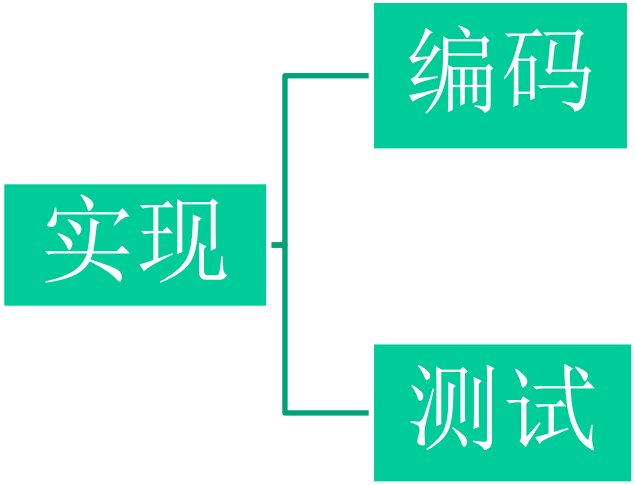
第6章 实现



编码就是把软件设计结果翻译成用某种程序设计语言书写的程序，是对设计的进一步具体化。

程序的质量主要取决于软件设计的质量。软件**测试**是保证软件质量的关键步骤，是对软件规格说明、设计和编码的最后复审。

主要内容

6.1 编码

6.2 软件测试基础

6.3 单元测试

6.4 集成测试

6.5 确认测试

6.6 白盒测试技术

6.7 黑盒测试技术

6.8 调试

6.9 软件可靠性

主要内容

6.1 编码

6.1.1. 选择程序设计语言

程序设计语言是人和计算机通信的最基本的工具，会影响人的思维和解题方式，影响人和计算机通信的方式和质量，影响其他人阅读和理解程序的难易程度。

选择适宜的程序设计语言的原因：

- 根据设计去完成编码时，困难最少；
- 可以减少需要的程序测试量；
- 可以得到更容易阅读和更容易维护的程序。

6.1 编码

高级语言优于汇编语言：

- 汇编语言编码需要把软件设计翻译成机器操作的序列，既困难又容易出差错；
- 高级语言写程序比用汇编语言写程序生产率可以提高好几倍；
- 用高级语言写的程序容易阅读、容易测试、容易调试、容易维护。

6.1 编码

6.1.2. 编码风格

源程序代码的逻辑简明清晰、易读易懂是好程序的一个重要标准，为了做到这一点，应该遵循下述规则。

1. 程序内部的文档

所谓程序内部的文档包括恰当的标识符、适当的注解和程序的视觉组织等。

- 标识符：含义鲜明的名字、缩写规则一致、为名字加注解；
- 注解：正确性，简要描述模块的功能、主要算法、接口特点、重要数据以及开发简史或解释包含这段代码的必要性；
- 视觉组织：适当的阶梯形式使程序的层次结构清晰明显。

6.1 编码

2. 数据说明

数据说明的原则：

- 数据说明的次序应该标准化；
- 当多个变量名在一个语句中说明时，应该按字母顺序排列这些变量；
- 如果设计时使用了一个复杂的数据结构，则应该用注解说明用程序设计语言实现这个数据结构的方法和特点。

6.1 编码

3. 语句构造

下述语句构造的原则有助于使语句简单明了：

- 不要为了节省空间而把多个语句写在同一行；
- 尽量避免复杂的条件测试；
- 尽量减少对“非”条件的测试；
- 避免大量使用循环嵌套和条件嵌套；
- 利用括号使逻辑表达式或算术表达式的运算次序清晰直观。

6.1 编码

4. 输入输出

在设计和编写程序时需考虑有关输入输出风格的规则：

- 对所有输入数据都进行检验；
- 检查输入项重要组合的合法性；
- 保持输入格式简单；
- 使用数据结束标记，不要要求用户指定数据的数目；
- 明确提示交互式输入的请求，详细说明可用的选择或边界数值；
- 程序设计语言对格式有严格要求时，应保持输入格式一致；
- 设计良好的输出报表；
- 给所有输出数据加标志。

6.1 编码

5. 效率

程序运行时间

写程序的风格会对程序的执行速度和存储器要求产生影响，应遵循的规则如下：

- 写程序之前先简化算术的和逻辑的表达式；
- 仔细研究嵌套的循环，以确定是否有语句可以从内层往外移；
- 尽量避免使用多维数组；
- 尽量避免使用指针和复杂的表；
- 使用执行时间短的算术运算；
- 不要混合使用不同的数据类型；
- 尽量使用整数运算和布尔表达式。

主要内容

6.1 编码

6.2 软件测试基础

6.3 单元测试

6.4 集成测试

6.5 确认测试

6.6 白盒测试技术

6.7 黑盒测试技术

6.8 调试

6.9 软件可靠性

6.2 软件测试基础

6.2.1. 软件测试的目标

测试的定义是“为了发现程序中的错误而执行程序的过程”。应该认识到测试决不能证明程序是正确的。即使经过了最严格的测试之后，仍然可能还有没被发现的错误潜藏在程序中。另外，在综合测试阶段通常由其他人员组成测试小组来完成测试工作。

6.2 软件测试基础

6.2.2. 软件测试准则

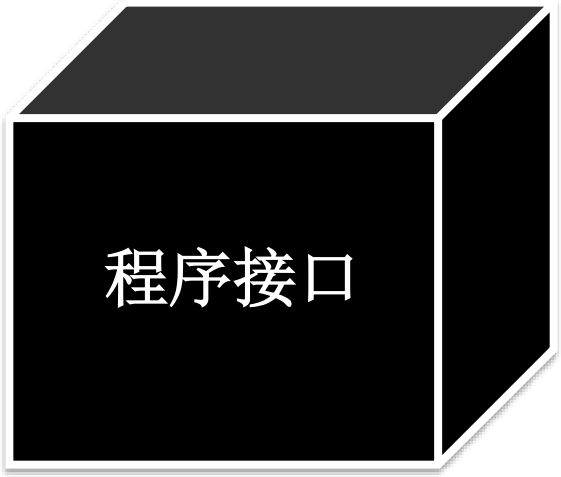
主要的软件测试准则如下：

- 所有测试都应该能追溯到用户需求；
- 应该远在测试开始之前就制定出测试计划；
- 应该从“小规模”测试开始，并逐步进行“大规模”测试；
- 穷举测试是不可能的；
- 为了达到最佳的测试效果，应该由独立的第三方从事测试工作。

6.2 软件测试基础

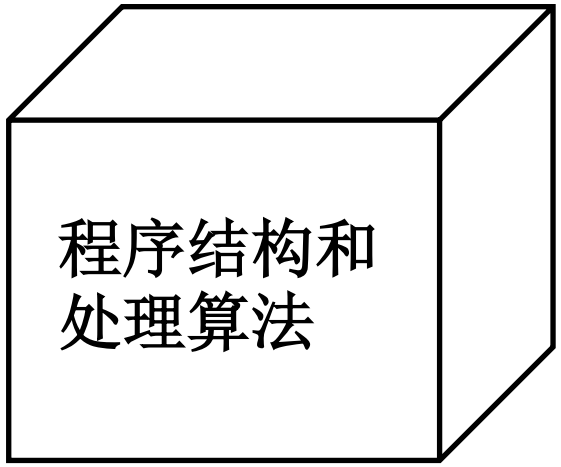
6.2.3. 测试方法

黑盒测试（又称功能测试）把程序看作一个黑盒子，完全不考虑程序的内部结构和处理过程。黑盒测试是在程序接口进行的测试，只检查程序功能是否能按照规格说明书的规定正常使用，程序是否能适当地接收输入数据并产生正确的输出信息，程序运行过程中能否保持外部信息（例如数据库或文件）的完整性。



6.2 软件测试基础

白盒测试（又称结构测试）是把程序看成装在一个透明的白盒子里，测试者完全知道程序的结构和处理算法。这种方法按照程序内部的逻辑测试程序，检测程序中的主要执行通路是否都能按预定要求正确工作。



6.2 软件测试基础

6.2.4. 测试步骤

根据第4条测试准则，测试过程也必须分步骤进行，后一个步骤在逻辑上是前一个步骤的继续。

大型软件系统通常由若干个子系统组成，每个子系统又由许多模块组成，因此，大型软件系统的测试过程基本上由**模块测试**、**子系统测试**、**系统测试**、**验收测试**和**平行运行**等五个步骤组成。

6.2 软件测试基础

1. 模块测试

在设计得好的软件系统中，每个模块完成一个清晰定义的子功能，而且这个子功能和同级其他模块的功能之间没有相互依赖关系。因此，有可能把每个模块作为一个单独的实体来测试，而且通常比较容易设计检验模块正确性的测试方案。

模块测试的目的是保证每个模块作为一个单元能正确运行，所以模块测试通常又称为**单元测试**。在这个测试步骤中所发现的往往是编码和详细设计的错误。

6.2 软件测试基础

2. 子系统测试

子系统测试是把经过单元测试的模块放在一起形成一个子系统来测试。模块相互间的协调和通信是这个测试过程中的主要问题，因此，**这个步骤着重测试模块的接口**。

3. 系统测试

系统测试是把经过测试的子系统装配成一个完整的系统来测试。在这个过程中不仅应该发现设计和编码的错误，还应该验证系统确实能提供需求说明书中指定的功能，而且系统的动态特性也符合预定要求。**在这个测试步骤中发现的往往是软件设计中的错误，也可能发现需求说明中的错误**。

子系统测试和系统测试，都兼有检测和组装两重含义，通常称为集成测试。

6.2 软件测试基础

4. 验收测试

验收测试把软件系统作为单一的实体进行测试，测试内容与系统测试基本类似，但是它是在用户积极参与下进行的，而且可能主要使用实际数据(系统将来要处理的信息)进行测试。

验收测试的目的是验证系统确实能够满足用户的需要，**在这个测试步骤中发现的往往是系统需求说明书中的错误**。验收测试也称为**确认测试**。

6.2 软件测试基础

5. 平行运行

所谓平行运行就是同时运行新开发出来的系统和将被它取代的旧系统，以便比较新旧两个系统的处理结果。这样做的具体目的有如下几点。

(1) 可以在准生产环境中运行新系统而又不冒风险。

(2) 用户能有一段熟悉新系统的时间。

(3) 可以验证用户指南和使用手册之类的文档。

(4) 能够以准生产模式对新系统进行全负荷测试，可以用测试结果验证性能指标。

主要内容

6.1 编码

6.2 软件测试基础

6.3 单元测试

6.4 集成测试

6.5 确认测试

6.6 白盒测试技术

6.7 黑盒测试技术

6.8 调试

6.9 软件可靠性

主要内容

6.1 编码

6.2 软件测试基础

6.3 单元测试

6.4 集成测试

6.5 确认测试

6.6 白盒测试技术

6.7 黑盒测试技术

6.8 调试

6.9 软件可靠性

6.3 单元测试

2. 局部数据结构

对于模块来说，局部数据结构是常见的错误来源。应该仔细设计测试方案，以便发现局部数据说明、初始化、默认值等方面的错误。

3. 重要的执行通路

由于通常不可能进行穷尽测试，因此，在单元测试期间选择最有代表性、最可能发现错误的执行通路进行测试是十分关键的。应该设计测试方案用来发现由于错误的计算、不正确的比较或不适当的控制流而造成的错误。

6.3 单元测试

4. 出错处理通路

好的设计应该能预见出现错误的条件，并且设置适当的处理错误的通路。不仅应该在程序中包含出错处理通路，而且应该认真测试这种通路。评价出错处理通路应该着重测试下述一些可能发生的错误。

(1)对错误的描述是难以理解的；

(2)记下的错误与实际遇到的错误不同；

(3)在对错误进行处理之前，错误条件已经引起系统干预；

(4)对错误的处理不正确；

(5)描述错误的信息不足以帮助确定造成错误的位置。

6.3 单元测试

5.边界条件

- 边界测试是单元测试中最后的也可能是最重要的任务。
- 软件常常在它的边界上失效，例如，处理n元数组的第n个元素时，或做到i次循环中的第i次重复时，往往会发生错误。
- 使用刚好小于、刚好等于和刚好大于最大值或最小值的数据结构、控制量和数据值的测试方案，非常可能发现软件中的错误。

6.3 单元测试

6. 代码审查

代码检查是指由审查小组正式对源程序进行人工测试。它是一种非常有效的程序验证技术，对于典型的程序来说，可以查出30%～70%的逻辑设计错误和编码错误。审查小组最好由下述4人组成。

(1) 组长，应该是一个很有能力的程序员，而且没有直接参与这项工程；

(2) 程序的设计者；

(3) 程序的编写者；

(4) 程序的测试者。

主要内容

6.1 编码

6.2 软件测试基础

6.3 单元测试

6.4 集成测试

6.5 确认测试

6.6 白盒测试技术

6.7 黑盒测试技术

6.8 调试

6.9 软件可靠性

6.4 集成测试

6.4.1. 自顶向下集成

- **自顶向下集成方法**是从主控制模块开始，沿着程序的控制层次向下移动，逐渐把各个模块结合起来。在把附属于（及最终附属于）主控制模块的那些模块组装到程序结构中去时，或者使用深度优先的策略，或者使用宽度优先的策略。
- **深度优先的结合方法**先组装在软件结构的一条主控制通路上的所有模块。选择一条主控制通路取决于应用的特点，并且有很大任意性。
- **宽度优先的结合方法**是沿软件结构水平地移动，把处于同一个控制层次上的所有模块组装起来。

6.4 集成测试

6.4.2. 自底向上集成

自底向上测试从“原子”模块(即在软件结构最低层的模块)开始组装和测试。因为是从底部向上结合模块，总能得到所需的下层模块处理功能，所以不需要存根程序。

用下述步骤可以实现自底向上的结合策略。

① 把低层模块组合成实现某个特定的软件子功能的族；

② 写一个驱动程序(用于测试的控制程序)，协调测试数据的输入和输出；

③ 对由模块组成的子功能族进行测试；

④ 去掉驱动程序，沿软件结构自下向上移动，把子功能族组合起来形成更大的子功能族。

上述第②～④步实质上构成了一个循环。

6.4 集成测试

6.4.3. 不同集成测试策略的比较

- **自顶向下测试方法**的**主要优点**是不需要测试驱动程序，能够在测试阶段的早期实现并验证系统的主要功能，而且能在早期发现上层模块的接口错误。
- **自顶向下测试方法**的**主要缺点**是需要存根程序，可能遇到与此相联系的测试困难，低层关键模块中的错误发现较晚，而且用这种方法在早期不能充分展开人力。
- **自底向上测试方法的优缺点与上述自顶向下测试方法的优缺点刚好相反。**

主要内容

6.1 编码

6.2 软件测试基础

6.3 单元测试

6.4 集成测试

6.5 确认测试

6.6 白盒测试技术

6.7 黑盒测试技术

6.8 调试

6.9 软件可靠性

6.5 确认测试

- 确认测试也称为验收测试，它的目标是验证软件的有效性。
- 通常，验证指的是保证软件正确地实现了某个特定要求的一系列活动；确认指的是为了保证软件确实满足了用户需求而进行的一系列活动。
- 软件有效性的一个简单定义是：如果软件的功能和性能如同用户所合理期待的那样，软件就是有效的。
- 需求分析阶段产生的软件需求规格说明书，准确地描述了用户对软件的合理期望，因此是软件有效性的标准，也是进行确认测试的基础。

6.5 确认测试

Alpha和Beta测试

- 如果一个软件是为许多客户开发的（例如，向大众公开出售的盒装软件产品），那么绝大多数软件开发商都使用被称为Alpha测试和Beta测试的过程，来发现那些看起来只有最终用户才能发现的错误。
- Alpha测试由用户在开发者的场所进行，并且在开发者对用户的“指导”下进行测试。开发者负责记录发现的错误和使用中遇到的问题。
- Alpha测试是在受控的环境中进行的。
- Beta测试由软件的最终用户们在一个或多个客户场所进行。与Alpha测试不同，开发者通常不在Beta测试的现场。
- Beta测试是软件在开发者不能控制的环境中的“真实”应用。

主要内容

6.1 编码

6.2 软件测试基础

6.3 单元测试

6.4 集成测试

6.5 确认测试

6.6 白盒测试技术

6.7 黑盒测试技术

6.8 调试

6.9 软件可靠性

6.6 白盒测试技术

- 通常把测试数据和预期的输出结果称为测试用例。
- 不同的测试数据发现程序错误的能力差别很大，为了提高测试效率降低测试成本，应该选用高效的测试数据。因为不可能进行穷尽的测试，所以选用少量“最有效的”测试数据，做到尽可能完备的测试就更重要了。
- 设计测试方案的基本目标是，确定一组最可能发现某个错误或某类错误的测试数据。已经研究出许多设计测试数据的技术，这些技术各有优缺点；同一种技术在不同的应用场合效果可能相差很大，因此，通常需要联合使用多种设计测试数据的技术。

6.6 白盒测试技术

- 逻辑覆盖：语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖、点覆盖、边覆盖和路径覆盖
- 控制结构测试：基本路径测试、条件测试、循环测试

主要内容

6.1 编码

6.2 软件测试基础

6.3 单元测试

6.4 集成测试

6.5 确认测试

6.6 白盒测试技术

6.7 黑盒测试技术

6.8 调试

6.9 软件可靠性

6.7 黑盒测试技术

黑盒测试着重测试软件功能。黑盒测试并不能取代白盒测试，它是与白盒测试互补的测试方法，它很可能发现白盒测试不易发现的其他类型的错误。

黑盒测试力图发现下述类型的错误：

(1) 功能不正确或遗漏了功能；

(2) 界面错误；

(3) 数据结构错误或外部数据库访问错误；

(4) 性能错误；

(5) 初始化和终止错误。

6.7 黑盒测试技术

白盒测试在测试过程的早期阶段进行，而黑盒测试主要用于测试过程的后期。设计黑盒测试方案时，应该考虑下述问题。

（1）怎样测试功能的有效性？

（2）哪些类型的输入可构成好测试用例？

（3）系统是否对特定的输入值特别敏感？

（4）怎样划定数据类的边界？

（5）系统能够承受什么样的数据率和数据量？

（6）数据的特定组合将对系统运行产生什么影响？

应用黑盒测试技术，能设计出满足下述标准的测试用例集。

（1）所设计出的测试用例能够减少为达到合理测试所需要设计的测试用例的总数。

（2）所设计出的测试用例能够告诉人们，是否存在某些类型的错误，而不是仅仅指出与特定测试相关的错误是否存在。

主要内容

6.1 编码

6.2 软件测试基础

6.3 单元测试

6.4 集成测试

6.5 确认测试

6.6 白盒测试技术

6.7 黑盒测试技术

6.8 调试

6.9 软件可靠性

6.8 调试

- **调试**（也称为纠错）作为成功测试的后果出现，即调试是在测试发现错误之后排除错误的过程。
- 软件错误的外部表现和它的内在原因之间可能并没有明显的联系。**调试**就是把症状和原因联系起来的尚未被人深入认识的智力过程。

6.8.1. 调试过程

- 调试不是测试。
- 调试过程从执行一个测试用例开始，评估测试结果，如果发现实际结果与预期结果不一致，则这种不一致就是一个症状，它表明在软件中存在着隐藏的问题。调试过程试图找出产生症状的原因，以便改正错误。

6.8 调试

调试过程总会有以下两种结果之一：①找到了问题的原因并把问题改正和排除掉了；②没找出问题的原因。在后一种情况下，调试人员可以猜想一个原因，并设计测试用例来验证这个假设，重复此过程直至找到原因并改正了错误。

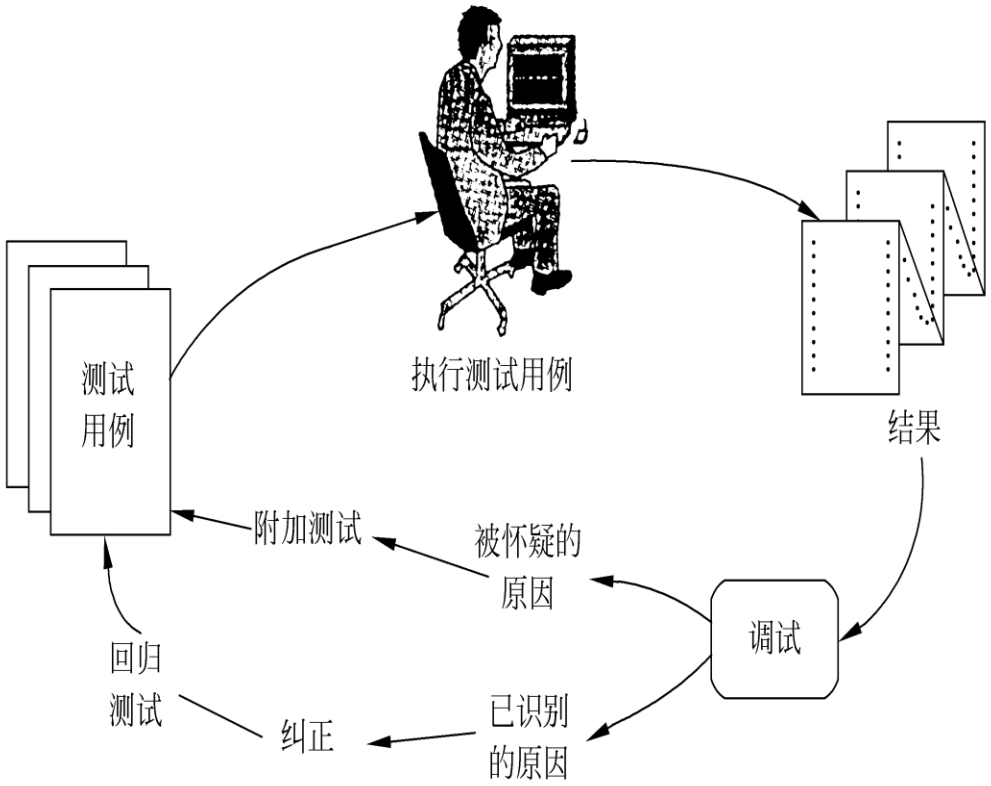


图 调试过程

6.8 调试

6.8.2. 调试途径

1. 蛮干法

- 蛮干法可能是寻找软件错误原因的最低效的方法。仅当所有其他方法都失败了的情况下，才应该使用这种方法。
- 蛮干法按照“让计算机自己寻找错误”的策略，这种方法印出内存的内容，激活对运行过程的跟踪，并在程序中到处都写上WRITE（输出）语句，希望在这样生成的信息海洋的某个地方发现错误原因的线索。
- 在更多情况下这样做只会浪费时间和精力。在使用任何一种调试方法之前，必须首先进行周密的思考，必须有明确的目的，应该尽量减少无关信息的数量。

6.8 调试

2. 回溯法

- 回溯是一种相当常用的调试方法，当调试小程序时这种方法是有效的。具体做法：从发现症状的地方开始，人工沿程序的控制流往回追踪分析源程序代码，直到找出错误原因为止。
- 随着程序规模的扩大，应该回溯的路径数目变得越来越大，回溯法不适用于这种规模的程序。

3. 原因排错法

对分查找法、归纳法和演绎法都属于原因排除法。

6.8 调试

对分查找法的基本思路是，如果已经知道每个变量在程序内若干个关键点的正确值，则可以用赋值语句或输入语句在程序中点附近“注入”这些变量的正确值，然后运行程序并检查所得到的输出。

归纳法是从个别现象推断出一般性结论的思维方法。使用这种方法调试程序时，首先把和错误有关的数据组织起来进行分析，以便发现可能的错误原因。然后导出对错误原因的一个或多个假设，并利用已有的数据来证明或排除这些假设。

演绎法从一般原理或前提出发，经过排除和精化的过程推导出结论。采用这种方法调试程序时，首先设想出所有可能的出错原因，然后试图用测试来排除每一个假设的原因。

主要内容

6.1 编码

6.2 软件测试基础

6.3 单元测试

6.4 集成测试

6.5 确认测试

6.6 白盒测试技术

6.7 黑盒测试技术

6.8 调试

6.9 软件可靠性

6.9 软件可靠性

基本概念

软件可靠性是程序在给定的时间间隔内，按照规格说明书的规定成功地运行的概率。软件可靠性随着给定的时间间隔的加大而减少。

一般说来，对于任何其故障是可以修复的系统，都应该同时使用可靠性和可用性衡量它的优劣程度。

软件可用性是程序在给定的时间点，按照规格说明书的规定，成功地运行的概率。

可靠性和可用性之间的主要差别是，可靠性意味着在0到t这段时间间隔内系统没有失效，而可用性只意味着在时刻t，系统是正常运行的。