第六章

软件测试及软件质量控制

软件系统的开发过程中，软件测试占据着重要地位。尽管人们采取了多种保证软件质量的措施，由于软件系统的客观复杂性，人们的主观认识不可能完全符合客观实际，完美无缺，每个阶段的技术审查也不可能毫无遗漏地查出和纠正所有的设计和分析上的错误，在软件生命周期的各个阶段，都不可避免地会产生差错，这些差错迟早会在软件的生产和使用过程中暴露出来。

软件工程实践的经验表明，发现软件的时刻越晚，改正这些错误所花费的代价也越高，如果在软件投入使用之前没有发现和纠正软件的大部分错误，人们付出的代价会更高，往往会造成恶劣的后果。

从广义上来说，软件测试工作散布在软件生命周期的各个开发阶段，人们认识到，软件测试是保证软件质量的主要手段，各阶段的评审工作和验证工作，均是广义概念上的测试工作。而主要的测试是在编码和测试这两个阶段进行的。因此，狭义的软件测试就是程序测试。

6.1 软件测试基本概念

G.J.Myers给出了关于测试的一些规则，被软件工程领域认可:

（1）测试是为了发现程序中的错误而执行程序的过程;

（2）好的测试方案极有可能发现迄今为止尚未发现的错误;

（3）成功的测试是发现了至今为止尚未发现的错误。

6.1 软件测试基本概念

这些规则，实际上是软件测试的狭义概念——程序测试。

狭义的软件测试：测试是为了发现错误而执行程序的过程。是根据软件开发的各个阶段的说明和程序的内部结构而精心设计的一批测试用例（有输入数据及预期的结果），并利用这些测试用例执行程序及发现错误的过程。

6.1 软件测试基本概念

广义的软件测试是对软件计划、软件系统分析、软件设计、软件编码进行的查错活动，包括代码执行和人工审查活动，测试的目的是找出软件生命周期的各个阶段的错误，有利于以后进行修改和纠正。但测试本身不修正错误，调试才会修正错误。即找错的活动是测试；分析错误的性质与位置，进行纠错的活动是调试，保证算法的正确实现。软件测试与程序测试都是查找错误的活动，差别在于查找错误的范围不同。

6.1 软件测试基本概念

由于测试的目标是暴露程序的错误，从心理学角度看，由设计者自己进行测试是不恰当的，设计小组和测试小组应该分别设立，有利于进行客观和公正的软件测试。测试是有限的，由于通常的测试过程不可能穷尽一切情况，即使经过了严格的测试之后，仍然可能存在没有被发现的错误隐藏在程序中，不能证明程序中没有错误。

6.1 软件测试基本概念

因此，测试仅仅有可能找出程序的错误，测试不能证明程序是正确的。软件工程中所有其它阶段都是“建设性”的，软件工程师力图从抽象概念出发，逐步设计出具体的软件系统，而测试人员的工作表面上看却是“破坏性”的，竭力证明软件中含有错误，不能按预定要求正确工作。凡是进行对比的方式均可理解为测试验证。

6.1.2 软件测试的对象

软件测试应该贯穿于软件生命期的各个阶段，各阶段的工作是相互衔接、相互影响的，前一阶段发生的问题自然要影响到下一阶段的工作。为了把握各个环节的正确性，人们需要进行各种确认和验证工作。

软件确认是广义上的软件测试，它是企图证明软件在一个给定的外部环境中软件的逻辑正确性的一系列活动和过程，如需求说明书的确认、程序的确认等。

6.1.2 软件测试的对象

- 程序的确认又分为静态确认与动态确认。

静态确认一般不在计算机上执行程序，而是通过程序正确性证明、静态分析或人工分析来确认程序的正确性;

动态确认主要通过动态分析和动态测试，用执行程序的过程来检查执行的状态，确认程序是否有问题;

6.1.2 软件测试的对象

正确性证明主要是企图绕过复杂的测试，直接证明程序的正确性。

- 如程序的输入输出断言法。

设程序段为S，其前断言为P，后断言为R。如果执行S以前P为真，则执行S后R也为真，则证明S是正确的，记为{P}S{R}。

6.1.2 软件测试的对象

任何程序总可以分成S1、S2、... Sn个结点，对应的断言为R1、R2、...、Rn，起初R1为输入断言，R2为输出断言，也是下一个输入断言，... Rn为最后的输出断言，我们总可以，将S1、S2、... Sn逐个证明，自顶向下或自底向上都可证明程序的正确性，该分支已发展为计算机代数学;

6.1.2 软件测试的对象

软件验证也属于广义上的软件测试，它试图证明在软件生命期的各个阶段、各阶段的逻辑协调性、完备性和正确性。

包括系统分析员理解用户要求的正确性、表达的正确性、设计人员对需求规格说明理解的正确性、设计与设计表达的正确性、程序编码的正确性和运行软件程序时输入的正确性、运行结果的正确性等，运行结果与用户预期的结果是否一致等，这说明任何一个环节上发生了问题都可能在软件测试中表现出来。

6.1.3 测试信息流

将测试的过程用数据流图表示，可得测试信息流如图6-1所示。

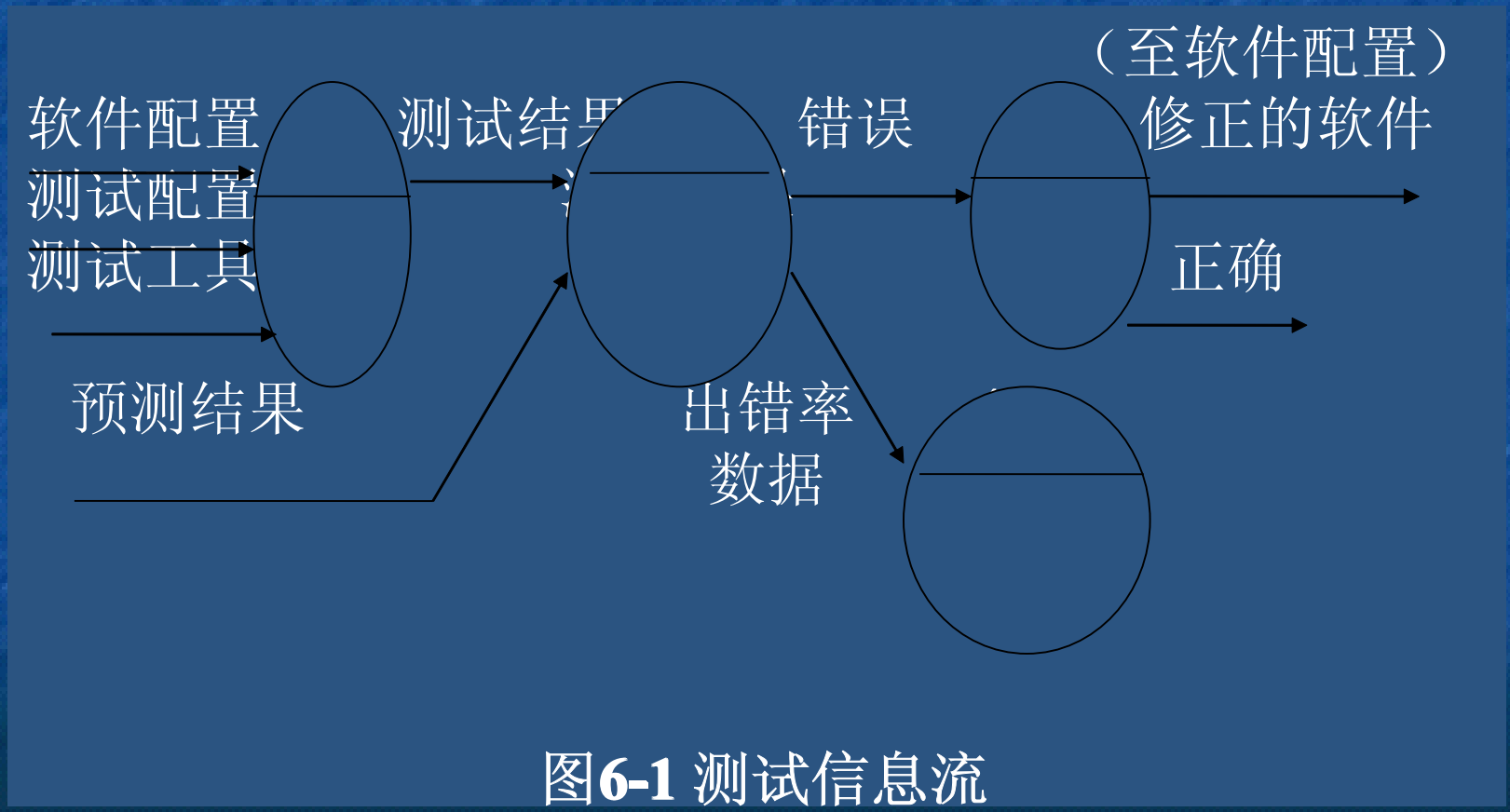


图6-1 测试信息流

6.1.3 测试信息流

1. 测试过程需要三类输入:

（1）软件配置：包括软件开发文档（用户文档、需求规格说明、软件设计说明、源程序代码）、目标执行程序、数据结构;

（2）测试配置：包括测试计划、测试用例、测试驱动程序等；实际上在整个软件开发过程中，测试配置只是软件配置的一个子集;

6.1.3 测试信息流

（3）测试工具：为提高软件测试效率，使用测试工具为测试工作服务；如：测试数据自动生成程序，静态分析程序、动态分析程序、测试结果分析程序及标准例程测试数据库等。

6.1.3 测试信息流

测试之后，对所有测试结果进行分析，将实际测试的结果与预期的结果进行比较。如果发现出错的数据，则意味着软件有错误，需要纠错，应进行调试，确定错误的位置和出错的性质，改正这些错误，同时修正相关文档。修正过的文档一般需经过再次测试，直到通过测试为止。

6.1.3 测试信息流

通过收集和分析测试结果的有关数据，可以建立软件评估的可靠性模型。

如果经常出现需要修改设计的严重错误，那么软件的质量和可靠性就值得怀疑，同时也表明需要进一步测试。

相反，如果软件功能能够正确完成，出现的错误易于修改，那么就可能有两种评价：

6.1.3 测试信息流

- 一种是软件的质量和可靠性达到可以接受的程度。
- 另一种是所做的测试还不足以发现软件的严重错误。

如果得到的评价是没有发现错误，很有可能测试的配置考虑得不够充分和细致，软件仍有潜伏的错误，以后改正错误需要付出高昂的代价。

6.1.3 测试信息流

2. 软件错误可以从不同角度进行分类:

（1）从错误对程序的影响程度来分:

<1>严重性错误：严重影响程序的运行，甚至不能运行;

<2>一般性错误：经常影响程序的运行，特殊情况下表现正常;

6.1.3 测试信息流

<3> 微小错误：一般情况下程序能运行，特殊情况下表现异常；

<4> 无影响性错误：不影响程序的运行。

6.1.3 测试信息流

（2）从开发过程的转换环节上分:

<1> 构造错误：编码实现与设计不一致;

<2> 设计错误：设计逻辑与说明不一致;

<3> 说明书错误：说明书与用户要求不一致;

<4> 需求错误：不满足用户的实际要求;

<5>配置错误：软件配置不满足实际环境。

6.1.3 测试信息流

（3）从测试结果的表现上分类：

1）功能错误：由系统需求分析不完整引起的；

2）结构错误：由总体设计的错误引起的；

3）过程错误：由详细设计的错误引起的；

4）数据错误：由软件编码或详细设计的错误引起的；

5）编码错误：由软件编码引起的错误；

6）其它错误：由文档和其它系统元素引起的错误；

6.1.4 软件测试步骤与软件开发各阶段的关系

软件测试一般分为四个步骤:

（1）单元测试（也称模块测试）：针对软件设计的基本单元——程序模块，进行正确性检验的测试工作。目的在于发现各个模块内部可能存在的各种差错。单元测试需要从程序内部结构出发设计测试用例，多个模块可以平行、独立地进行测试;

6.1.4 软件测试步骤与软件开发各阶段的关系

（2）集成测试（也称组装测试，联合测试）：在单元测试的基础上，将所有模块按设计要求集成在一起进行测试，以检验总体设计中各模块间的接口设计问题、模块之间的相互影响、上层模块存在的各种差错及全局数据结构对系统的影响等方面。

6.1.4 软件测试步骤与软件开发各阶段的关系

（3）确认测试（也称验收测试，有效性测试）：主要检验软件的功能和性能是否与需求说明书中的规定一致。

（4）系统测试：将软件系统作为一个元素，放入整个实际的计算机系统中，与计算机硬件、其他软件、使用人员等系统元素结合在一起，在实际使用环境下进行综合全面的测试。

6.1.4 软件测试步骤与软件开发各阶段的关系

前面多次强调，使用软件生命期（瀑布模型）模型，软件开发过程是一个自顶向下，逐步细化的过程，而软件测试过程则是与开发过程相反的次序进行的，是一个自底向上，逐步集成的过程，低一层测试为上一层测试准备测试条件和数据驱动环境，也包含两者平行进行测试。

6.1.4 软件测试步骤与软件开发各阶段的关系

因此，发现引起错误的原因顺序也与开发过程的相次序反，首先对每一个模块进行单元测试，消除程序模块内部逻辑上和功能上的错误和缺陷，再对照软件设计进行集成测试（有时也叫整体测试），检测和排除子系统或系统结构上的错误，再对照需求进行确认测试（也称为有效性测试），最后进行系统测试，运行系统，看软件系统是否满足功能和性能及其它要求。

6.1.4 软件测试步骤与软件开发各阶段的关系

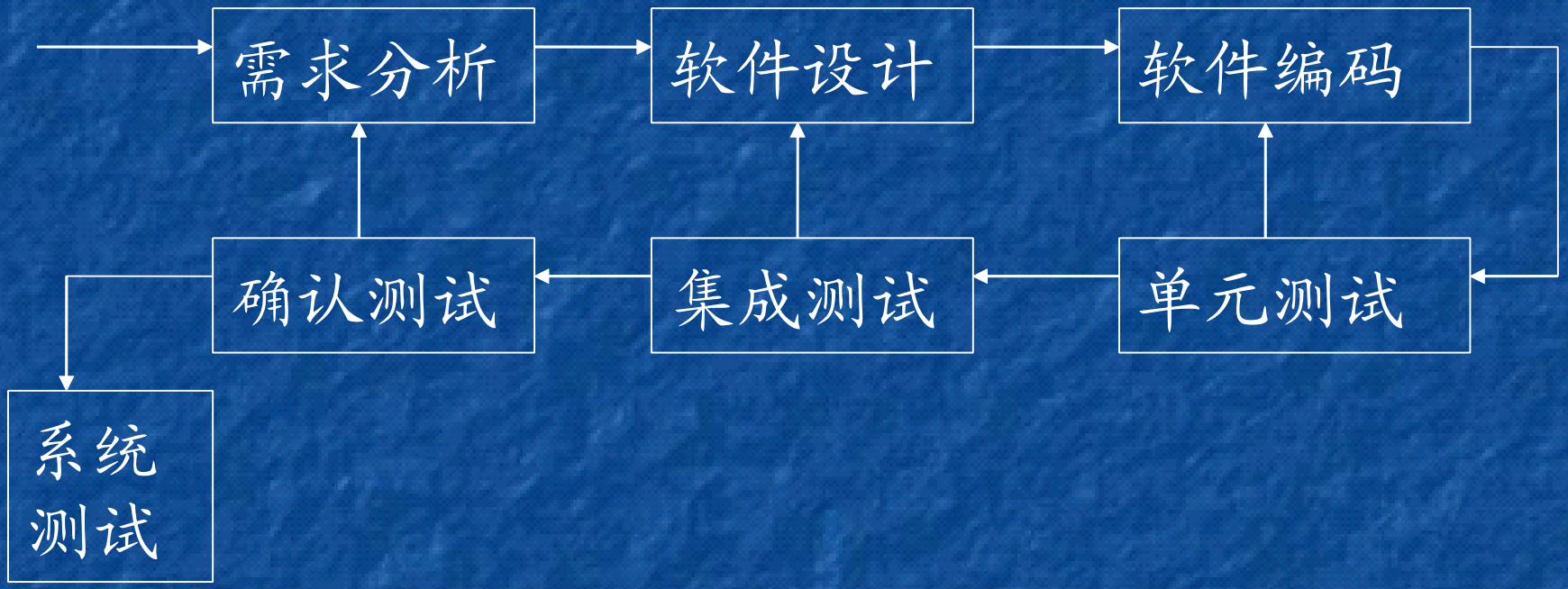


图6-2 软件测试与软件开发过程间的关系

6.1.4 软件测试步骤与软件开发各阶段的关系

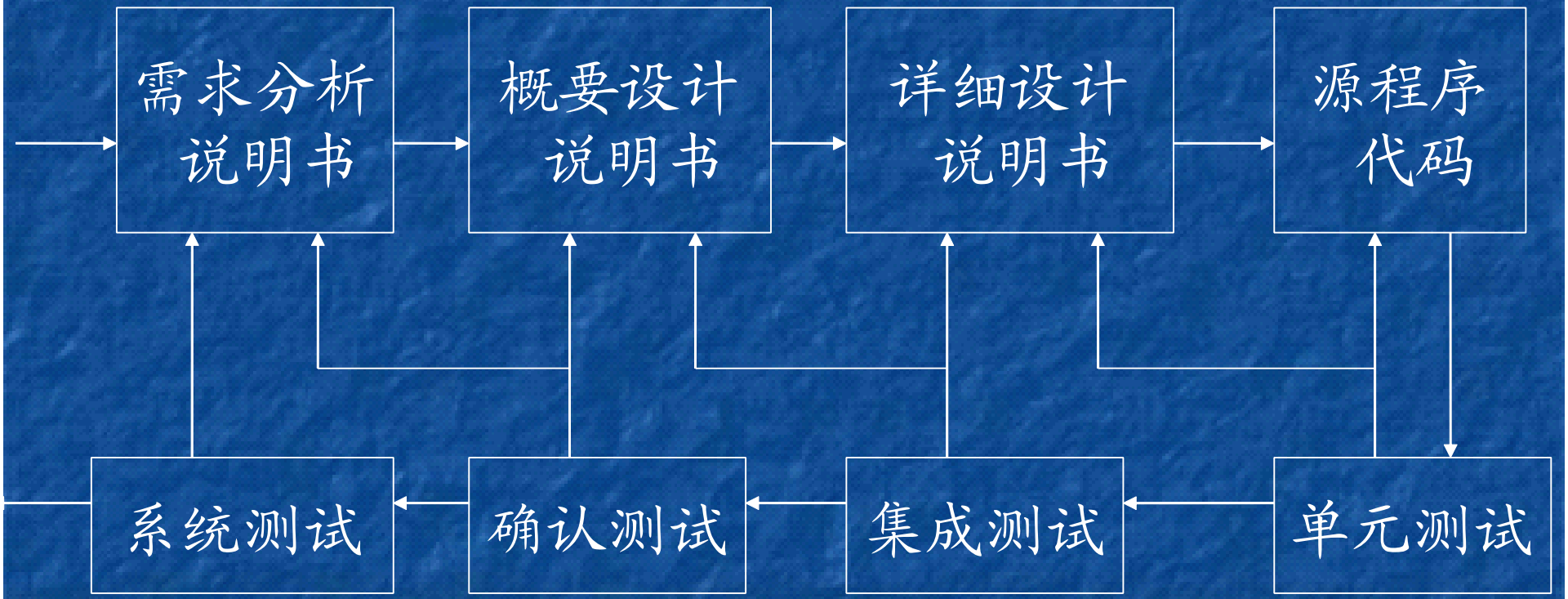


图6-3 软件测试与开发文档之间的关系

6.1.5 软件测试原则

- （1）将软件测试贯穿于软件开发的各个阶段中，在开发过程中尽早地发现和预防错误，杜绝隐患，提高软件质量；
- （2）测试用例必须包含输入数据和与之对应的预期输出结果，精心设计测试用例；
- （3）测试时应避免设计者检查自己设计的程序；
- （4）设计测试用例时，应包括合理的与不合理的输入条件；

6.1.5 软件测试原则

- （5）充分注意测试中出现的错误群集现象，若发现错误数目较多，则可能残存的错误数目也较多，这种错误出现的群集现象，已为许多程序测试实践所证实；
- （6）严格执行测试计划，以软件需求说明书为基准设计测试用例，排除测试的随意性；

6.1.5 软件测试原则

- （7）对每一个测试结果做全面检查，不能遗漏错误出现的征兆，软件修改后要进行回归测试，即用修改前测试过的测试用例进行测试，再用新的测试用例测试;
- （8）妥善保存测试计划、测试用例、出错统计数据和最终分析报告，为维护提供方便。

在一个程序段中，还存在着尚未发现的错误概率与已发现的错误数正相关。

6.1.5 软件测试原则

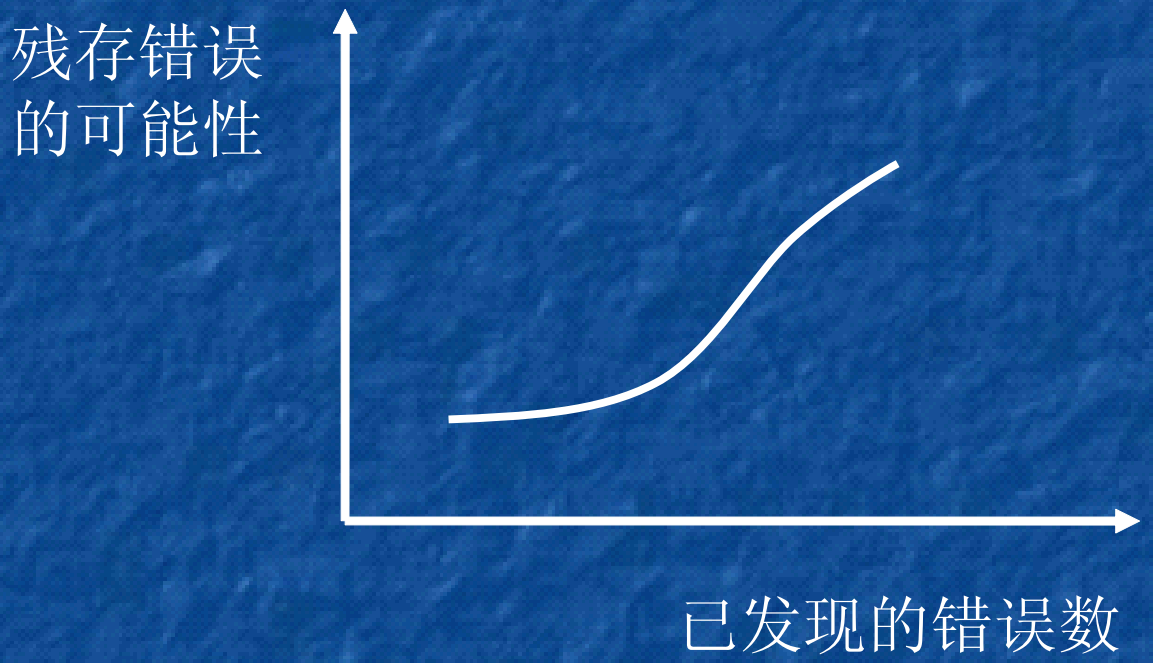


图6-4 软件错误的群集现象示意图

6.2 软件测试的方法

软件的测试方法很多，不同的出发点得到不同的测试方法。有：

- 从测试过程来分：静态分析法、动态测试法；
- 从观察结构的透明性方式来分：白盒法、黑盒法、灰盒法；
- 从获得测试数据形式上分：穷尽法；等价类划分法；边界值分析法；

6.2 软件测试的方法

- 从逻辑分析上分：因果图法；错误推测法；
- 从测试步骤上分：单元测试、集成测试、确认测试、系统测试等；
- 从考察形式上分：功能测试，逻辑测试；

6.2 软件测试的方法

如何测试得更完全、怎样进行测试用例的设计，是软件测试中的关键技术。无论用哪种方法进行测试，都是设法用较少的测试用例集合测试出程序中较多的潜在错误。

静态分析时，不执行程序，可对需求分析说明书、软件设计说明书、源程序做结构检查、流图分析、符号执行来分析软件可能导致的异常情况，找出软件错误。从测试过程来分：静态分析法、动态测试法；

6.2 软件测试的方法

结构检查是手工分析技术，对需求说明、程序设计、编码、测试工作进行评议，虚拟地（模拟）执行程序，在评议中发现和检查错误;

流图分析是通过分析流程图、代码结构来检查程序错误，便于进行编码分析和测试结果分析;

6.2 软件测试的方法

符号执行是定义符号化数据，为程序的每条路径给出符号表达式，对特定路径输入符号，经处理输出符号，判断程序的行为是否错误，这种方法复杂，易出错，较少使用。

灰盒法是白盒法和黑盒法相结合使用的方法，仅对重点路径和程序段用白盒法测试，大部分用黑盒法进行测试。

6.2 软件测试的方法

动态测试是直接执行程序进行测试，包括功能测试、接口测试和结构测试，观察程序的行为，记录执行的结果，从执行结果来分析程序可能出现的错误；

有些人设想，不管使用那种测试方法，只要对每一种可能发生的情况都进行测试，能正确通过，就可以得到完全正确的程序。

6.2 软件测试的方法

包含所有可能情况的测试称为穷尽测试，实际上，通常不可能做到穷尽测试。因为各种输入数据的排列组合情况往往多到无法实际测试完成的程度。如用黑盒法测试三个整数型的输入数据，如果每个整数是16位二进制数，则输入数据有

$2^{16} \times 2^{16} \times 2^{16}=2^{48} \approx 2.8 \times 10^{14}$种排列组合。

6.2 软件测试的方法

如果每测试一次需要1毫秒，测试完毕这些排列组合的各种情况需要一万年，另外还需测试不合法的输入情况，实际上不可能穷尽所有组合情况。因此，一般的软件测试是有限测试。

Alpha（α）测试：通用软件产品为了征集用户的意见，在开发者的场所，由用户进行的测试，记录用户发现的错误和问题。

6.2 软件测试的方法

Beta(β)测试：在一个或多个用户自己的场所，由最终用户进行，并记录在测试中遇到的所有问题和想法。

重要的通用软件产品，大多经过α和β测试。

6.3 测试方案与测试用例

设计测试方案是软件测试中的关键问题。测试方案包括预定要测试的功能、结构，应该要输入的测试数据和输入这些数据后预期的结果——测试用例。测试用例的设计是其中较困难的问题，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测试，选用少量高效的测试数据，进行尽可能完备的测试就显得更重要了。

6.3 测试方案与测试用例

设计测试方案的基本目标是，确定一组最有可能发现某个或某类错误的测试用例。有多种测试技术，同一种测试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测试技术来设计测试用例。

通常的做法是用黑盒法设计基本测试方案，再用白盒法补充一些方案。

6.4 白盒法（逻辑覆盖）

白盒法也称逻辑驱动法（逻辑覆盖法），从软件的具体逻辑结构和执行路径出发，设计测试用例。具有语句覆盖、判定覆盖（分支覆盖）、条件覆盖、判定/条件覆盖、路径覆盖、条件组合覆盖、点覆盖、边覆盖，下面以一个经典例子分别介绍:

设有某个算法片段的程序流程图如下:

6.4 白盒法（逻辑覆盖）

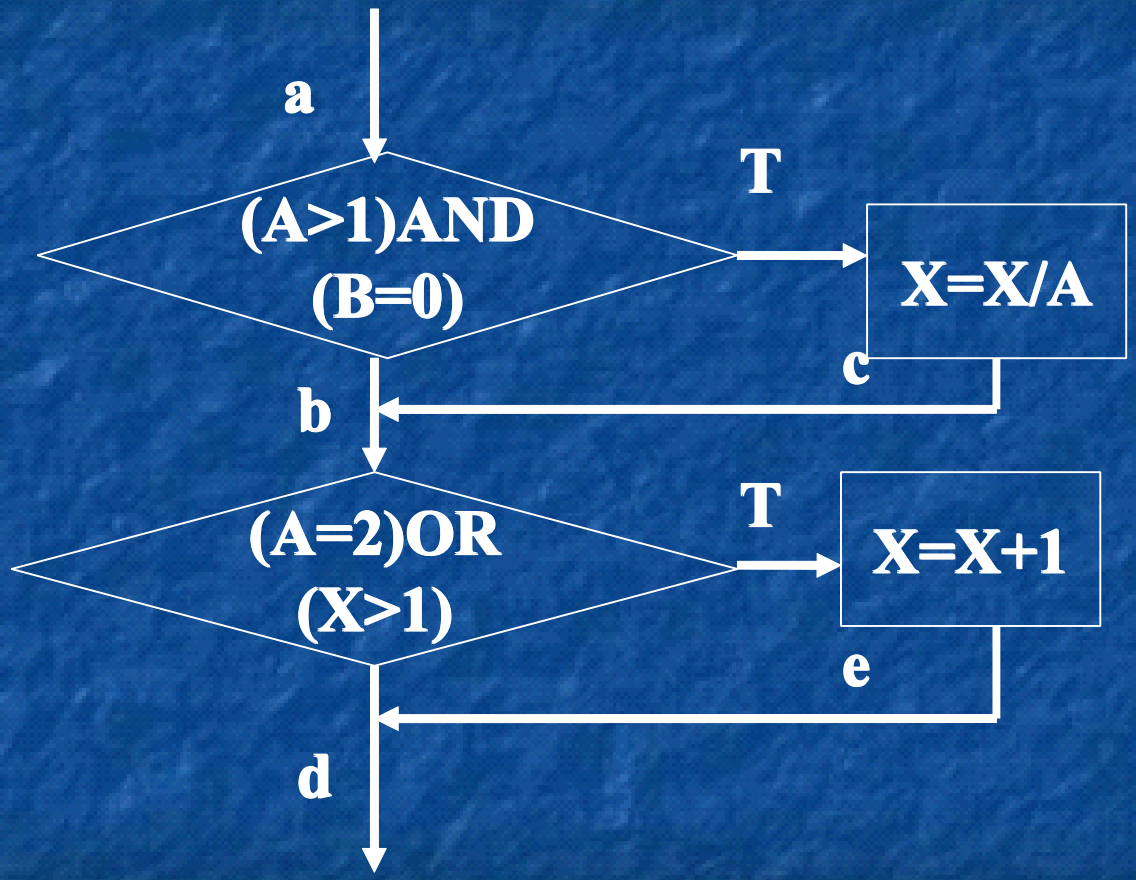


图6-5程序段程序框图

6.4 白盒法（逻辑覆盖）

该程序片段有四条路径：abd, acd, ace, aed。

- (1) 语句覆盖：选择足够的测试用例使程序中每条语句至少执行一次。

为了使每个语句都执行一次，程序的执行路径只需经过a、b、c、d、e各点即可。如果选择路径ace，则能保证程序中的语句都执行一次。

6.4 白盒法（逻辑覆盖）

例如，选择测试用例：A=2, B=0, X=3,

预期的结果为：

A=2, B=0, X=2.5;

但是，许多路径得不到测试，这种测试很不充分。

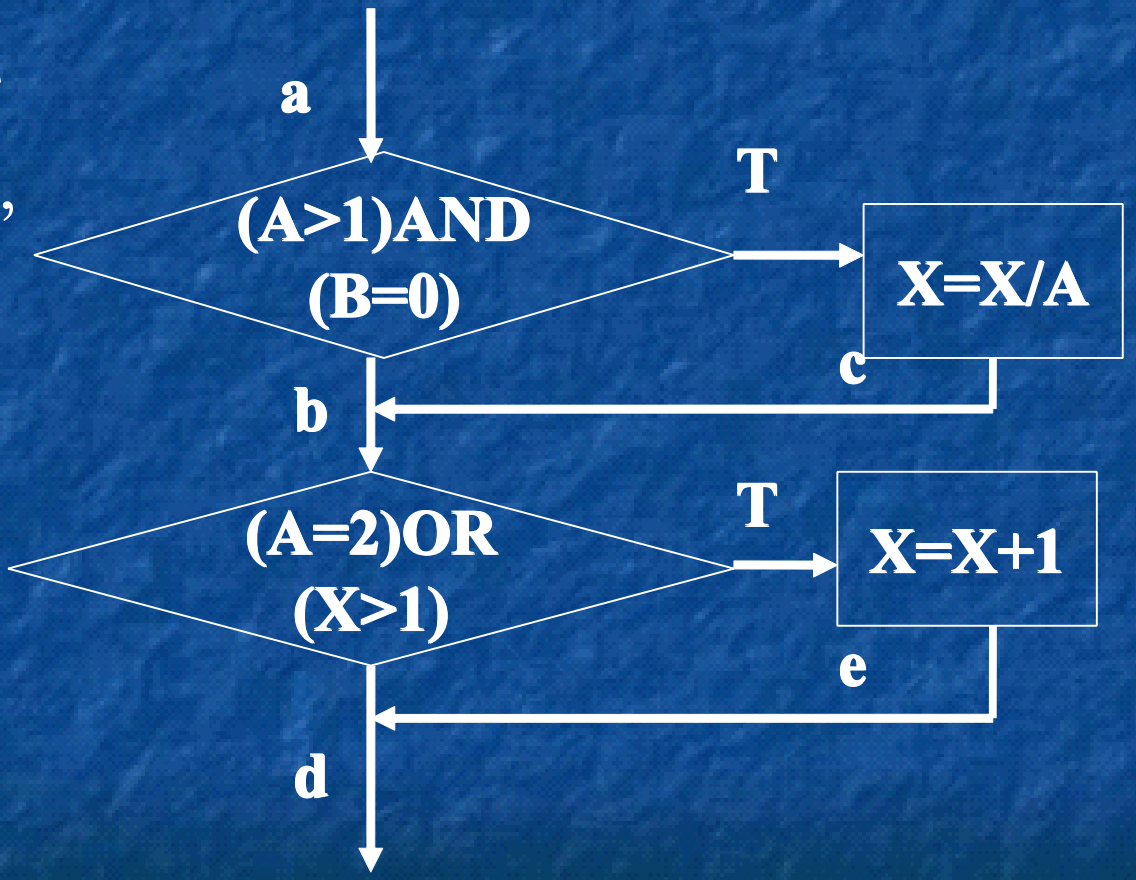


图6-5程序段程序框图

6.4 白盒法（逻辑覆盖）

- (2) 判定覆盖（也称分支覆盖）：判定是一个逻辑表达式的结果。选择足够的测试用例，使程序中每个判定至少都能获得一次“真”值和一次“假”值，从而使程序的每个判定的每个分支至少都执行一次。

例如，选择测试用例：A=3, B=0, X=3, 预期结果为：A=3, B=0, X=1;

选择测试用例：A=2, B=1, X=0, 预期结果为：A=2, B=1, X=1;

6.4 白盒法（逻辑覆盖）

这组测试用例覆盖了路径acd和aed，满足了判定覆盖要求。判定覆盖比语句覆盖强，但是判定覆盖只关心整个判定表达式的值，对程序逻辑的覆盖程度仍然不高，如上面的测试，只覆盖了全部路径的一半路径。

6.4 白盒法（逻辑覆盖）

- (3) 条件覆盖：条件为逻辑表达式中的各个逻辑分量。选择足够的测试用例，使得程序判定中的每个条件都能获得各种可能的结果。

如图6-5中，有四个条件：A>1, B=0, A=2, X>1, 每个条件可能出现的各种结果为：a点出现：A>1, A≤1； B=0, B≠0;

b点出现：A=2, A≠2； X>1, X≤1;

6.4 白盒法（逻辑覆盖）

例如，选择测试用例：A=2, B=0, X=4, 预期结果为：A=2, B=0, X=3;

选择测试用例：A=1, B=1, X=1, 预期结果为：A=1, B=1, X=1;

这组测试用例覆盖了路径acd，aed 和abd ，满足了条件覆盖要求。条件覆盖比判定覆盖强，它使判定表达式中的每个条件都取得了两个不同的结果。

6.4 白盒法（逻辑覆盖）

但也有相反的情况，每个条件虽然取得两个不同的结果，判定表达式却始终只取一个值，例如：取数据：

- A=2, B=0, X=1； 满足A>1, B=0, A=2, X ⩽ 1的条件，执行路径ace;
- A=1, B=1, X=2； 满足A ⩽ 1, B ≠ 0, A ≠ 2, X>1的条件，执行路径abd;

满足了条件覆盖，却不满足判定覆盖，第二个判定表达式的值总为真。

6.4 白盒法（逻辑覆盖）

- (4) 判定/条件覆盖：选择足够的测试用例，使得程序判定中的每个条件都能获得各种可能的结果，并且使得每个判定都取得各种可能的结果。

  例如，选择测试用例：A=2, B=0, X=4, 预期结果为：A=2, B=0, X=3;

  选择测试用例：A=1, B=1, X=1, 预期结果为：A=1, B=1, X=1;

6.4 白盒法（逻辑覆盖）

这组测试用例覆盖了路径acd，aed 和abd ，满足了判定/条件覆盖要求。但它也并不比条件覆盖更强。

6.4 白盒法（逻辑覆盖）

- （5）条件组合覆盖：选择足够的测试用例，使得程序判定中的条件的各种可能组合都至少出现一次。

如图6-5中，需要测试覆盖条件组合的下述八种情况：

1） A>1, B=0 ； 2） A>1, B ≠ 0； 3） A ⩽ 1, B=0；

4 ） A ⩽ 1, B ≠ 0； 5） A=2, X>1 6） A=2, X<=1

7） A ≠ 2, X>1 8） A ≠ 2, X ⩽ 1

6.4 白盒法（逻辑覆盖）

- 用A=2, B=0, X=4,　预期结果：A=2, B=0, X=3,　覆盖情况1）、5）；
- 用A=2, B=1, X=1,　预期结果：A=2, B=1, X=2,　覆盖情况2）、6）
- 用A=1, B=0, X=2,　预期结果：A=1, B=0, X=3,　覆盖情况 3)、7)
- 用A=1, B=1, C=1,　预期结果：A=1, B=1, X=1,　覆盖情况 4)、8)

6.4 白盒法（逻辑覆盖）

条件组合覆盖是最强的覆盖，虽然这四个测试实现了条件组合覆盖，但并没有覆盖每一条路径，如：路径acd遗漏了。

以上各种技术基本上是依次增强的顺序，但测试用例的数量也急剧增加。开销大，应注意权衡。

6.4 白盒法（逻辑覆盖）

- (6)点覆盖：图论中的点覆盖定义为：如果连通图G的子图G'是连通的，而且包含G的所有节点，则称G'是G的点覆盖。如果把程序流程图的每个处理框（含一个或多个语句）作为一个节点，就画出了程序图。满足点覆盖的要求是选取足够多的测试用例，测试执行程序时的路径，至少经过程序图的每个节点一次。显然，点覆盖的要求和语句覆盖的要求是相同的。

6.4 白盒法（逻辑覆盖）

(7) 边覆盖：图论中的边覆盖定义为：如果连通图G的子图G'是连通的，而且包含G的所有边，则称G'是G的边覆盖。为了满足边覆盖的测试要求，使得程序的执行路径经过程序图中的每一条边。通常边覆盖和判定覆盖是一致的。

6.4 白盒法（逻辑覆盖）

(8) 路径覆盖：选择足够的测试用例，使得程序中的每条可能组合路径都至少执行一次。（如果程序图中有环，则每个环至少经过一次。）它是相当强的逻辑覆盖标准，选择的测试用例更具有代表性，暴露错误的能力也更强。

6.4 黑盒法（逻辑覆盖）

黑盒测试法把程序看成是一个黑盒子，不考虑程序内部的执行过程，着眼于外部特性，在接口上进行测试，仅考虑输入与输出能否与需求规格说明书对应起来，输入能否正确的接收，输出能否得到正确的结果。也称为数据驱动或输入/输出驱动测试，或功能测试。

黑盒法包括等价类划分、边界值分析、因果图法。

6.5.1 等价类划分

一个理想的测试用例能够独自发现某一类错误。一般的测试是以输入数据为基础进行的，我们着眼于划分输入数据值的情况，以便找出有代表性的测试数据，减少测试工作量。

6.5.1 等价类划分

假设我们可以把输入的数据域划分成有限的等价类，用每个等价类的代表值作为测试用例的输入数据进行测试，等价于该类的任何其它值作为设计用例的输入数据进行的测试。即：如果等价类中的一个测试用例检测出程序的一个错误，那么这一等价类的其余测试用例也能发现同样的错误。相反，若测不出错误，则该等价类的其他测试用例，也测不出错误。

6.5.1 等价类划分

等价类划分的原则：

（1）如果规定了输入值的取值范围，则可划分出一个有效的等价类（输入值在此范围内），两个无效的等价类（输入值小于最小值、或大于最大值）；

（2）如果规定了输入数据的个数，则类似地也可以划分出一个有效等价类和两个无效等价类。

6.5.1 等价类划分

（3）如果规定了输入数据的一组值，而且程序对不同输入值做不同地处理，则每个允许的值是一个有效的等价类，还有一个无效的等价类（任何一个不允许的输入值）；

（4）如果规定了输入数据必须遵循的规则，则可以划分出一个有效的等价类（符合规则的输入数据）和若干个无效的等价类（从各种不同角度违反规则）；

6.5.1 等价类划分

（5）如果输入数据为整型，则可以划分出正整数、零和负整数三个有效类；

（6）如果程序处理的对象是表格，则应该使用空表、以及含有一项或多项的表进行测试。

以上列举了可能情况的一部分，还可以根据经验进行划分。上面是针对输入数据而言，对输出数据也可类似划分。

6.5.1等价类划分

根据等价类划分来设计测试方案时主要使用下面的两个步骤（先划分好等价类）：

- 1）设计一个新的测试方案，尽可能多的覆盖尚未被覆盖的有效等价类；重复这一步骤直到所有有效等价类都被覆盖为止；
- 2）设计一个新的测试方案，使它覆盖一个，而且只覆盖一个尚未被覆盖的无效等价类，重复这一步骤直到所有无效等价类都被覆盖为止；

6.5.2 边界值分析

经验表明，程序在处理边界情况时最容易发生错误——忽略边界数据域问题。所以选取稍微高于或低于边界值的数据进行测试。启发规则如下：

- （1）输入条件规定取值范围或输入个数时，取边界值的上下值或个数的上下界设计测试用例；
- （2）如果输出条件规定取值范围，取边界上下浮动值作为测试数据；

6.5.2 边界值分析

- （3）规格说明中提出输入输出有序集，取有序集的第一个和最后一个元素作为测试数据;
- （4）分析规格说明，找出其他可能存在的边界条件，取其上下浮动值作为测试数据。

6.5.3 因果图法

因果图可以提供逻辑条件和相应动作之间的简洁逻辑关系表示。

因果图的使用可以分为如下步骤:

（1）列出模块的原因——和效果（动作），给每个原因和效果一个标示符;

（2）把原因、效果用逻辑符号连接起来，画出原因效果图，标出约束条件;

6.5.3 因果图法

（3）原因相对于判定表中的条件，效果相对于判定表中的动作，把原因效果图转换为判定表；

（4）把判定表中右边部分的每一列表示的情况转换为测试用例。

6.5.3 错误推测法

错误推测法在很大程度上依靠人的直觉和经验进行。有时利用其他测试方法测试后的程序表现，推测应该如何进行下一步的测试。也可利用程序错误清单作为推测测试的依据。

6.5.3 综合测试策略

对软件系统的实际测试，往往利用多种测试方法进行，形成综合测试策略。通常是用黑盒法设计一些基本的测试用例，再用白盒法补充设计一些必要的测试用例，黑盒、白盒法相结合——灰盒法进行测试。具体策略为：

- (1) 在任何情况下 都应该使用边界值分析法进行测试。经验表明这种方法设计出的测试用例，暴露程序错误的能力最强，应该包括输入和输出数据的边界情况；

6.5.3 综合测试策略

- （2）必要时用等价类划分法补充测试用例;
- （3）必要时用错误推测法补充测试用例;
- （4）对照程序逻辑，检查设计测试用例，可根据对程序的可靠性要求采用不同的逻辑覆盖标准，补充测试用例，达到逻辑覆盖标准;
- （5）如果有输入条件的组合，就应从输入条件及其组合开始测试。

6.5.3 综合测试策略

对于集成测试可以使用模块的自顶向下的结合方式，也可以使用自底向上的结合方式进行测试，还可用辅助测试工具协助测试。

软件系统测试完后，应对软件配置进行复查，确保软件的有关文档资料的完整齐全，分类编目，便于软件的维护和修改。审查的资料包括：用户所需的文档（用户手册、操作手册）；设计文档；源程序；测试文档（测试说明书，测试报告）及其它说明等。

6.6 软件调试

程序测试只是发现错误的迹象，并不清楚具体错误的位置和产生的原因，应该立即进行调试，即纠正错误的工作，它包含两方面工作：

（1）确定程序中错误的具体位置和性质；

（2）修改错误。

调试必须由程序员自己来进行。

6.6 软件调试

调试技术有以下类别:

输出存储器内容：发现问题时，设法保留现场信息，把所有寄存器和主存中相关部分的内容打印出来进行分析研究。

打印关键变量的动态内容：为取得关键变量的动态值，在程序中插入标准的打印语句，检验在某个事件发生后变量是否按预期的要求进行变化。

6.6 软件调试

利用调试工具跟踪程序的动态变化，单步跟踪，检查主存和寄存器内容，检查重要变量内容，检查是否进入预定的程序分支，设置断点，当程序运行到检查的重点位置，暂停程序运行，观察主要的变化信息，分析程序状态，决定继续跟踪还是停止执行，为程序的调试提供了有力的手段。

调试的策略有：试探法，回溯法，对分查找法、归纳法、演绎法。可根据个人经验和具体情况灵活应用。

6.7 软件质量控制

高质量是产品得以存在和生长的前提，软件工程的主要目标之一就是要获得高质量的软件。在软件工程诞生之前，由于计算机设备条件的限制，计算机发展的早期，内存容量有限，执行速度不高，当时的软件设计特别强调效率。随着技术的发展，软件规模的扩大，软件复杂性的增加，人们对软件质量的观点早已发生了很大变化，更强调软件的全面质量评价。

6.7.1软件质量评价

不同的人员对软件质量关心的着重点不同，反映了对软件质量的不同要求。

用户关心软件产品是否满足规定的功能和性能要求，软件运行是否可靠，是否易于学习掌握，易于使用，是否有较高的运行效率，是否可以从一个环境移植到另一个环境等问题。

软件开发人员既要开发满足质量要求的最终产品，又要注意软件开发过程中的每个阶段的质量。

6.7.1软件质量评价

开发人员常把产品外部特性用软件内部质量结构来对应。

维护人员要求软件系统、软件文档清晰、软件文档与源代码一致，软件易于修改、易于维护。

管理人员关心的是软件的总体质量特性，在软件质量与开发工期之间进行折中选择;

6.7.1软件质量评价

但影响软件质量的各因素之间是相互联系、甚至是相互矛盾的，如追求可靠性要牺牲一定的时间和空间效率为代价，要求软件不但能在合法的输入情况下正确地运行，而且还应该能够安全地排除非法的入侵和处理意外的事件。

一般要求软件具有良好的结构，齐全的文档资料，易于阅读和理解，便于修改和维护，内部层次结构清晰、人机界面友好，用户乐于使用。

6.7.1软件质量评价

国际标准化机构建议，软件质量模型由三层组成:

高层：软件质量需求评价准则（SQRC）；

中层：软件质量设计评价准则（SQDC）；

低层：软件质量度量评价准则（SQMC）；

6.7.1软件质量评价

多数软件同行公认的一般质量按如下特性进行评价:

- 正确性: （功能度）在预定的环境下，软件实现的功能达到设计规范和满足用户要求的程度;
- 可靠性: 软件在给定的条件下和规定的时间内完成预定职能的概率; 即保持其性能的能力相关的属性;
- 易用性: 用户学习软件、运行操作软件、准备输入、理解输出所做的努力程度，根据用户评估使用软件

6.7.1软件质量评价

- 效率：在规定的条件下，软件表现的性能级别与所使用资源总量（包括人员、时间、财力）关系的属性；
- 可维护性：软件修改难易程度的一组属性；
- 可移植性；（可转换性）指一个软件从一个环境转换到另一个环境运行的能力的相关属性。

以上六条是常见的评价方面。

6.7.1软件质量评价

还有其它方面的特性来评价软件质量，衡量总体质量的优劣程度。如:

- 健壮性：在硬件发生故障、输入数据无效或操作失误等意外情况下，系统不至于崩溃，能作出适当响应的程度;
- 完整性：（安全性）对未经授权的人使用软件或数据的企图，系统能够控制（禁止）的程度;

6.7.1软件质量评价

- 可测试性：系统容易测试的程度；
- 可再用性：在其他应用中该软件可以被再次使用的程度（或范围）；
- 可互连性：把软件系统与其他系统连接起来的能力。

除了定性评价外，人们逐渐重视软件的质量度量，它是在系统运行过程中进行动态检测，不断收集软件性能方面的数据，利用软件质量模型（如软件可靠性模型、软件复杂度模型等）进行分析和评价。

6.7.2软件质量控制

做好软件质量的控制，就要加强软件生命期各个阶段的软件质量保证，需要做好如下几方面的软件管理工作:

（1）采用技术手段保证软件质量：在软件开发过程中，注意采用科学的软件工程方法和工具来保证所开发软件的质量;

6.7.2软件质量控制

（2）组织技术评审：在软件开发的每个阶段结束后，都要组织评审，对质量进行评价，可以及早地发现软件开发过程中的可能引起软件质量问题的潜在错误；

（3）加强软件测试：软件测试是软件质量保证的重要手段。测试可以发现软件中大多数隐藏的错误，测试愈充分，软件中的隐患就有可能暴露得愈彻底；

6.7.2软件质量控制

（4）推行软件工程标准：不同的软件开发机构都有自己的工程规范，根据ISO9000系列质量管理保证体系确定的规范，进行企业的认证工作。一旦确认，就应在软件开发中得到遵循，软件规范则成为软件技术评审的一项重要内容;

6.7.2软件质量控制

（5）对软件的修改、变更进行严格控制：影响软件的一个不可忽视的危险因素来自软件的修改和变更。尽管修改和变更总是有理由的，但在修改过程中常常会引进一些潜伏的错误。因此严格控制软件的修改和变更是十分必要的；

（6）对软件质量进行度量：软件质量管理要求对软件质量进行跟踪，就必须进行软件质量度量，并对软件质量情况及时记录和报告。

6.7.2软件质量控制

软件的审查过程有以下步骤:

- 制定审查计划：组织审查小组，安排日程，分发软件项目材料;
- 项目概貌介绍：当项目复杂时，由作者介绍概貌;
- 评审准备：评审人员阅读项目材料，了解有关项目的情况;
- 项目评审：召开评审会，讨论项目情况，发现和记录错误，督促修改;

6.7.2软件质量控制

- 项目修改返工：由作者修正已经发现的问题，提交修改结果；
- 复查：判断修改是否真正解决了问题。
- 管理复审：向开发组织或使用部门的管理人员，提供有关项目的总体状况、成本和进度等方面的情况，以便他们从管理角度对开发工作进行审查。

6.7.2软件质量保证

ISO 9000-3标准不适用于面向多数用户的程序软件包，仅适用于依照合同进行单独定货的开发软件，对供需方的责任都作了明确的规定，最重要的是质量保证体系，也是企业建立质量保证体系的指南，要求证实企业具有持续提供符合要求产品的能力，主要强调将质量作入产品之中：

（1）软件质量保证体系是贯穿于整个生存期的集成化过程体系，而不仅仅体现在最后的产品交付验收时

6.7.2软件质量保证

- （2）强调防患于未然而不是事后纠正;
- （3）更强调质量体系的文件化，实施内部质量审核制度;
- （4）强调对每一项软件开发都按计划开展质量活动并且确保相关组织机构的了解和监督;
- （5）进行合同审查，需求规格说明可用于产品交付验证时的认证;

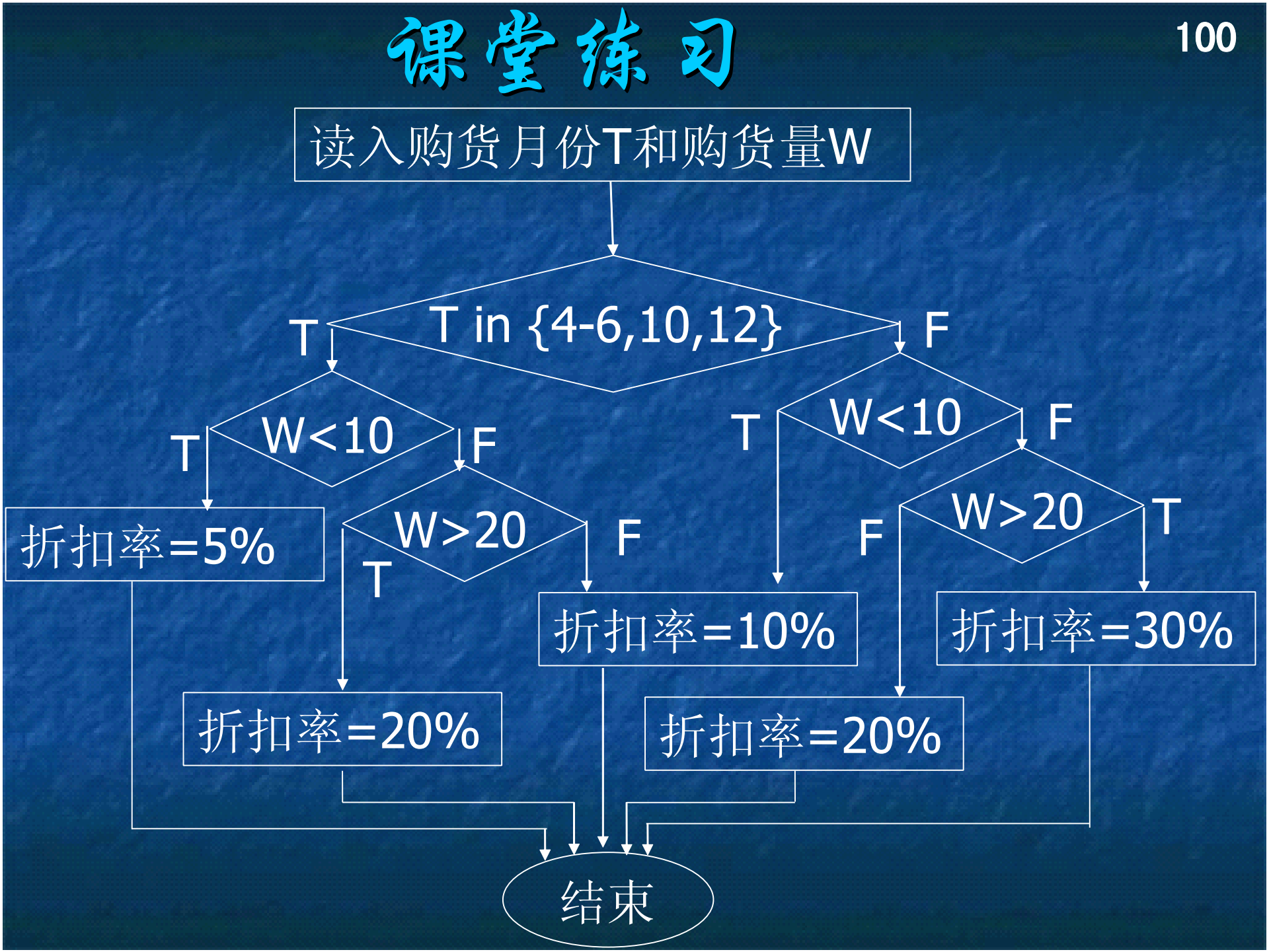
6.7.2软件质量保证

- （6）实施开发计划管理，质量计划管理，进行产品的设计和实现;
- （7）进行多层次的测试和验证，交付安装时提供义务期限;
- （8）提供维护支持、配置管理、文档控制、质量记录、培训人员和其它支持活动等;

6.7.2软件质量保证

总之，加强质量管理，针对所有可能影响软件质量的各个因素都要采取有力措施，作出加强质量管理和控制的决定。与质量有关的人员都要规定其职责和权限，使责任落实到人，保证产品质量真正得到控制。

课堂练习


读入购货月份T和购货量W
T in {4-6,10,12}
T
F
W<10
T
F
W>20
T
F
折扣率=5%
折扣率=20%
W<10
T
F
W>20
F
T
折扣率=10%
折扣率=20%
折扣率=30%
结束

习 题 六

1. 什么是软件测试？测试的根本目的是什么？

2. 测试用例包含哪两个部分？

3. 测试应遵循什么原则？

4. 测试有哪些方法？

5. 测试与软件开发的各个阶段有什么关系？

6. 分别用逻辑覆盖的几种方式设计某一程序片段，写出测试用例。

习 题 六

7. 解释下列术语：单元测试；集成测试；确认测试；系统测试；调试；

8. 测试与调试有什么区别？测试的策略有哪些？调试有哪几种技术？

9. 调试有哪几种策略？

10. 软件质量一般从哪几方面进行评价？

11. 如何加强软件质量控制？