

第7章 实现

目录

- 7.1 编码
- 7.2 软件测试基础
- 7.3 单元测试
- 7.4 集成测试
- 7.5 确认测试
- 7.8 调试
- 7.9 软件可靠性

编码与测试

- 编码和测试统称为实现。
- 编码：把软件设计结果翻译成程序。
- 测试：检测程序并改正错误的过程。

7.1 编码

选择程序设计语言

选择一种编程语言的理论标准：

- 1) 有理想的模块化机制；
- 2) 可读性好的控制结构和数据结构；
- 3) 便于调试和提高软件可靠性；
- 4) 编译程序发现程序错误的能力强；
- 5) 有良好的独立编译机制。

选择程序设计语言

选择语言时除了考虑理论上的标准，还必须同时考虑主要的实用标准：

- (1) 系统用户要求
- (2) 可以使用的编译程序
- (3) 可以得到的软件工具
- (4) 工程规模
- (5) 程序员知识
- (6) 软件可移植性要求
- (7) 软件的应用领域

编码风格

1. 程序内部的文档
2. 数据说明
3. 语句构造
4. 输入/输出
5. 效率

程序内部的文档

- 选取含义鲜明的名字，如果使用缩写，缩写规则要一致，并给每个名字加注释；
- 通常在每个模块开始处要有一段注释，描述模块功能、算法、接口特点等；
- 程序清单布局应利用适当的阶梯形式，使程序的层次结构清晰明显。

数据说明

- 数据说明的次序应该标准化，如按数据类型确定说明的次序；
- 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；
- 如果设计时使用了复杂的数据结构，应该用注释说明实现该数据结构的方法和特点。

7.2 软件测试基础

测试目标

- (1) 测试是为了发现程序中的错误而执行程序的过程；
- (2) 好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案；
- (3) 成功的测试是发现了迄今为止尚未发现的错误的测试。

测试准则

- 1) 所有测试都应该能追溯到用户需求；
- 2) 应该远在测试前就制定出测试计划；
- 3) 把Pareto原理应用到软件测试中；
- 4) 应该从“小规模”测试开始，并逐步进行“大规模”测试；
- 5) 穷举测试是不可能的；
- 6) 为了达到最佳测试效果，应该由独立的第三方从事测试工作。

穷举测试：包含所有可能情况的测试。为什么说穷举测试是不可能的？

测试方法

- 黑盒测试（功能测试）
 - 如果已经知道软件应该具有的功能，可以通过测试来检验是否每个功能都能正常使用，这种测试称黑盒测试。
- 白盒测试（结构测试）
 - 如果知道软件内部工作过程，可以通过测试来检验软件内部动作是否按照设计说明书的规定正常进行，这种测试称为白盒测试。

测试步骤

1. 模块测试

模块测试又称单元测试，它把每个模块作为单独的实体来测试。

2. 子系统测试（集成测试）

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。

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。

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（利用用户的实际数据测试）。

测试阶段的信息流

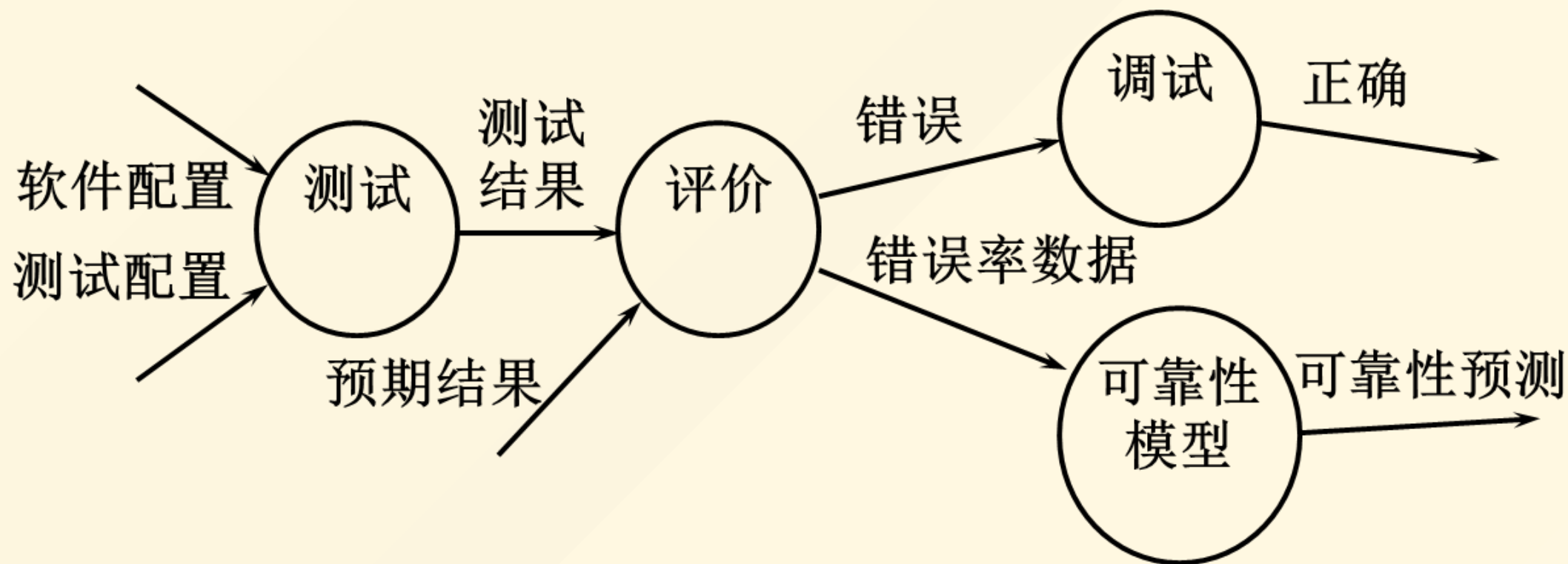


图7.1 测试阶段的信息流

7.3 单元测试

- 单元测试的一般过程是：
 - 首先通过编译系统检查并改正程序中所有的语法错误；
 - 然后用详细设计模块说明为指南，对重要的控制路径进行测试，以便发现模块内部的错误。
- 通常，单元测试使用白盒测试方法。

测试重点

1. 模块接口

应该对穿过模块接口的数据流进行检测，以保证正确的输入和输出。

2. 局部数据结构

这是错误的主要来源，应该设计相应的测试用例，以便发现数据结构方面的错误。

3. 重要的执行路径

由于不可能进行穷尽测试，因此选择测试路径是非常关键的。

4. 出错处理通路

5. 边界条件

代码审查（静态测试）

审查小组：

- 1) 组长；
- 2) 程序的设计者；
- 3) 程序的编写者；
- 4) 程序的测试者。

计算机测试（动态测试）

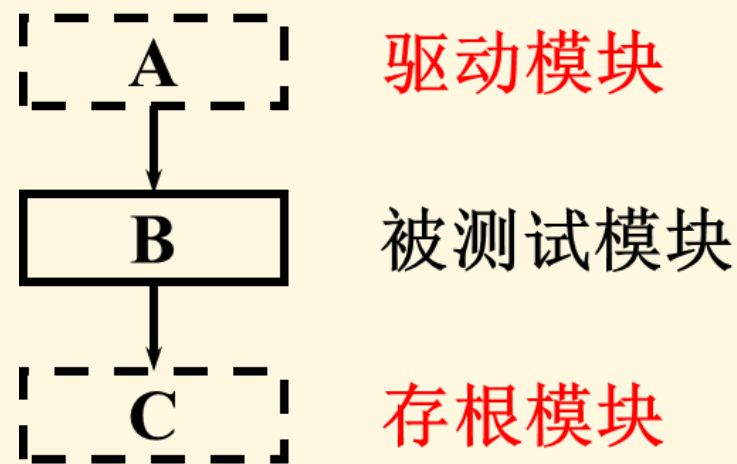
由于软件模块不是一个独立的系统，不能独立运行，要依靠其他模块调用，或需要调用其他模块。因此，必须要为测试的单元开发**驱动程序**（ driver ）或**存根程序**（ stub ）。

- 驱动程序
 - 相当于一个“主程序”，用来把测试数据传送给被测试的模块，并打印有关结果。
- 存根程序
 - 用来代替被测试模块所调用的模块，是对后者的简化。

驱动程序和存根程序

如，为了测试B模块，设计了A模块和C模块。

- 由A负责传送测试数据，由C负责模拟被B调用的模块。C模块可能没有，这取决于B有没有调用其他程序。
- A、C都是一次性程序，只在测试时临时使用，应尽量设计得简单一些，以节省费用和时间。



7.4 集成测试

集成测试是组装软件的系统化技术，它将经过单元测试的模块联系在一起进行测试。

由模块组装成程序时有两种方法：

- 非渐增式测试方法
先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序。
- 渐增式测试方法
每次增加一个待测试模块，把它同已经测试好的那些模块结合起来进行测试，反复进行直到完成所有模块测试的方法。

渐增式测试方法

- 自顶向下集成

自顶向下集成是一种递增的装配软件结构的方法，这种方法应用非常广泛。它需要存根程序，但是不需要驱动程序。

- 自底向上集成

自底向上集成方法是从软件结构最底层模块开始进行组装和测试，它与自顶向下结合方法相反，需要驱动程序，不需要存根程序。

自顶向下集成：深度 优先策略

先组装软件结构的一条主控制通路上的所有模块，选择哪条主控制通路，具有较大的任意性。

如图，如果选取左通路，就先把模块M1、M2、M5结合起来测试，然后结合模块M8、M6，再构造中央和右侧的控制通路。

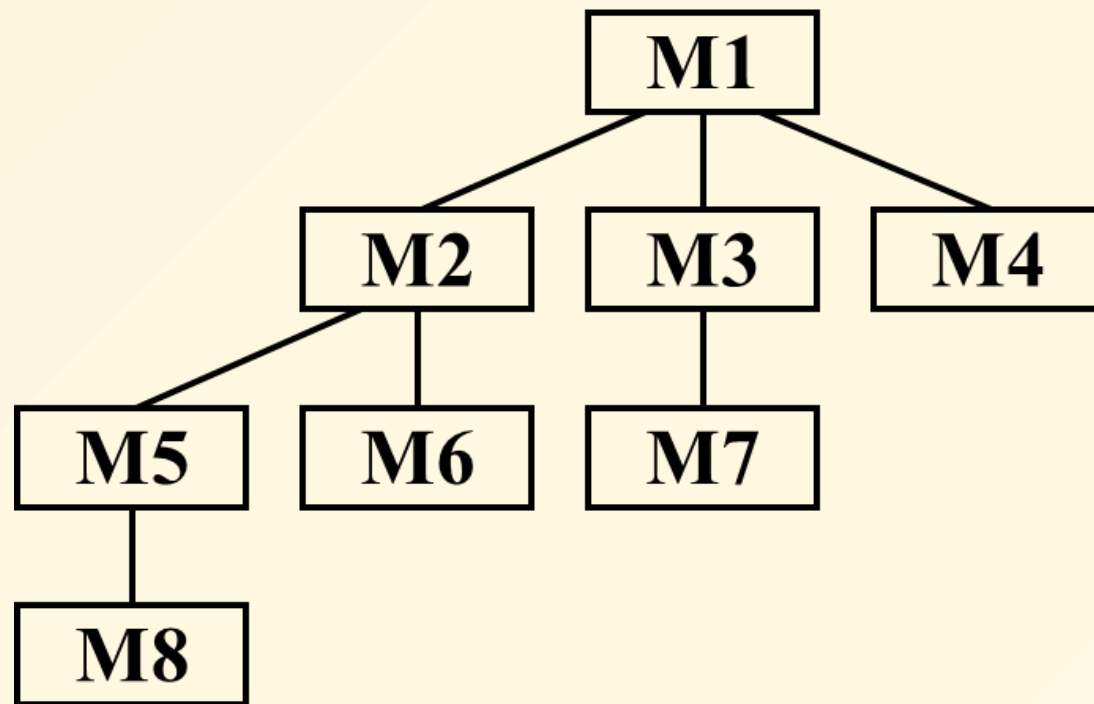


图7.3 自顶向下结合实例

自顶向下集成：宽度 优先策略

沿着软件结构水平地移动，把处于同一个层次的所有模块组装起来。

如图，首先结合M2、M3、M4和主控模块M1，然后结合下一个控制层次中的模块M5、M6和M7，最后结合模块M8。

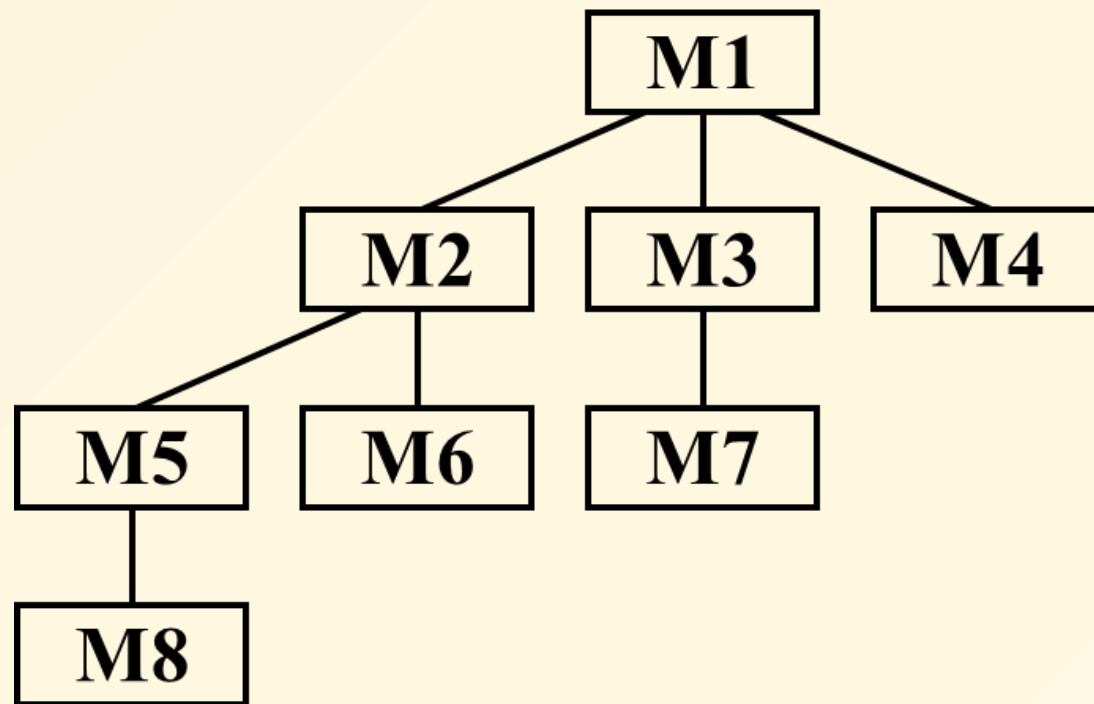


图7.3 自顶向下结合实例

自顶向下集成的基本过程

1. 对主控模块进行测试，测试时用存根程序代替所有直接被主控模块调用的模块；
2. 根据选定的结合策略（深度优先或宽度优先），每次用一个实际模块代替一个存根程序（新结合的模块往往又需要新的存根程序）；
3. 每结合一个模块，就测试一个；
4. 为保证不引入新的错误，需要进行**回归测试**，即重复以前进行过的部分或全部测试；
5. 重复回到第二步，直到构成整个软件结构。

自底向上集成的基本过程

1. 把底层模块组合成实现一个特定软件子功能的族，如图族1、2、3。
2. 为每个模块设计一个驱动程序，作为测试的控制程序，以协调测试用例的输入和输出。图中D1、D2、D3分别是族1、2、3的驱动程序。

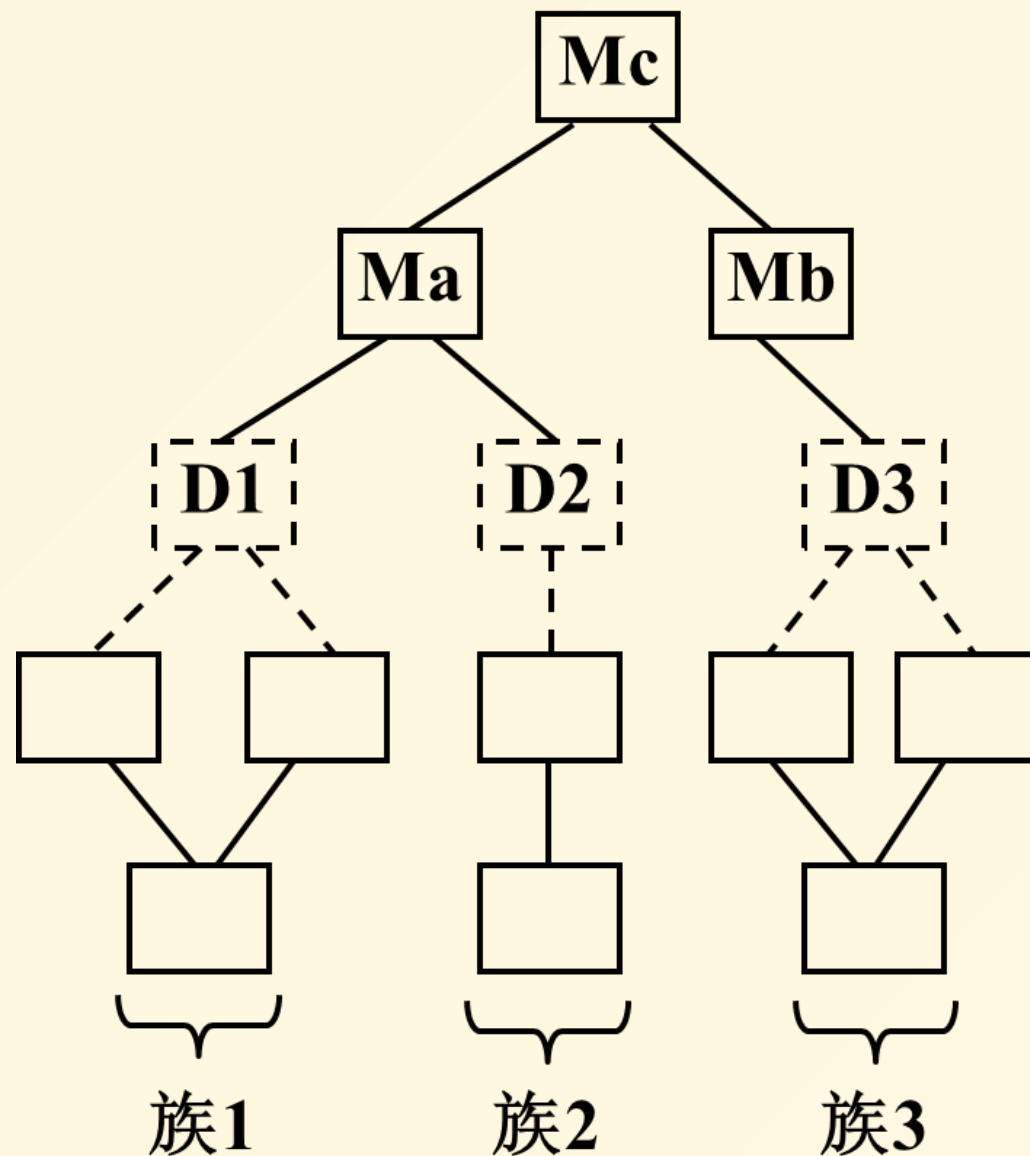


图7.4 自底向上实例

自底向上集成的基本过程（续）

3. 对模块进行测试；
4. 用实际模块代替驱动程序组装成新的模块族，在新加入的实际模块上面加上新的驱动程序进行测试；
5. 重复第二到第四步，逐渐向上加入实际模块，直至构造出整个软件结构。

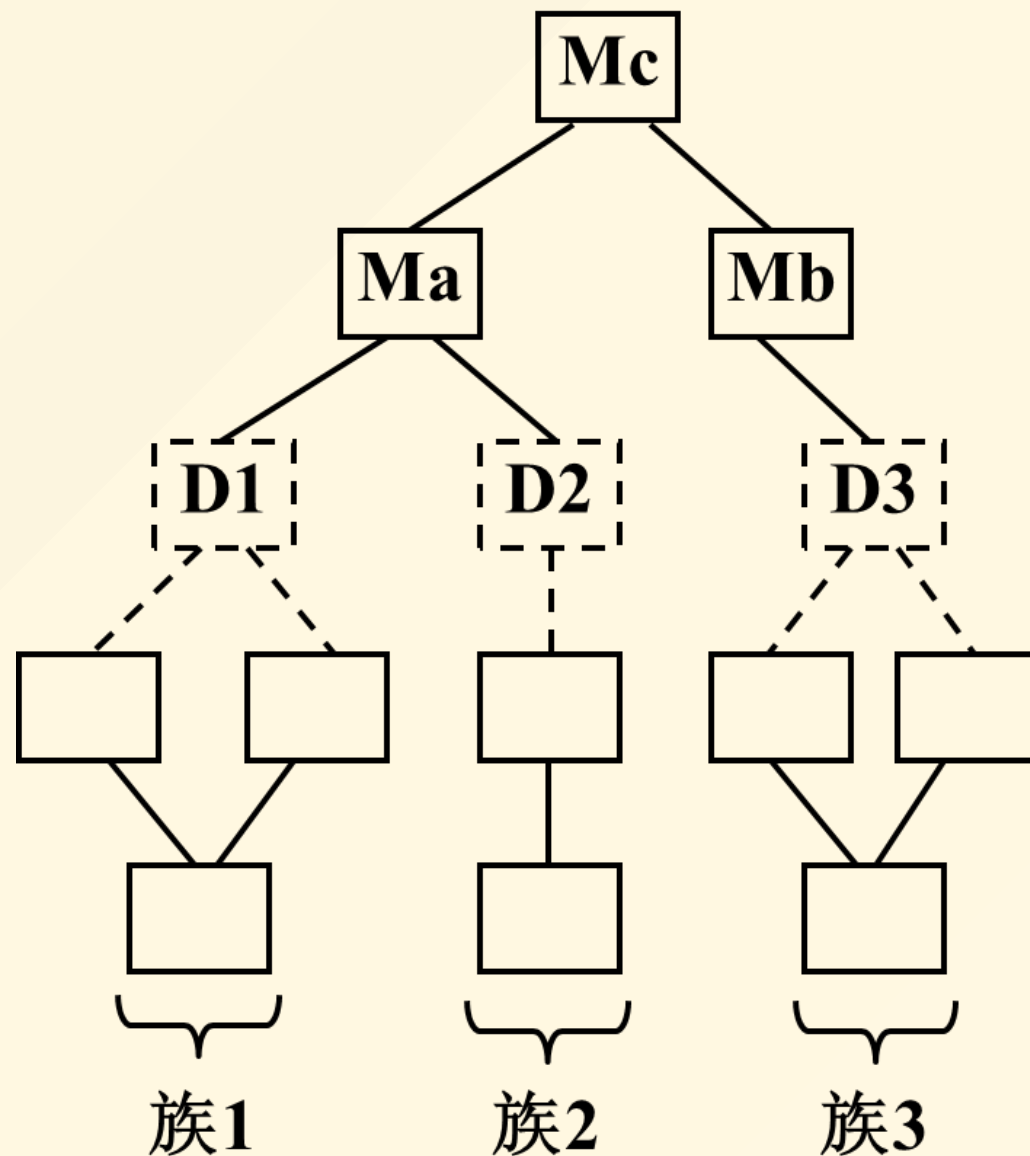


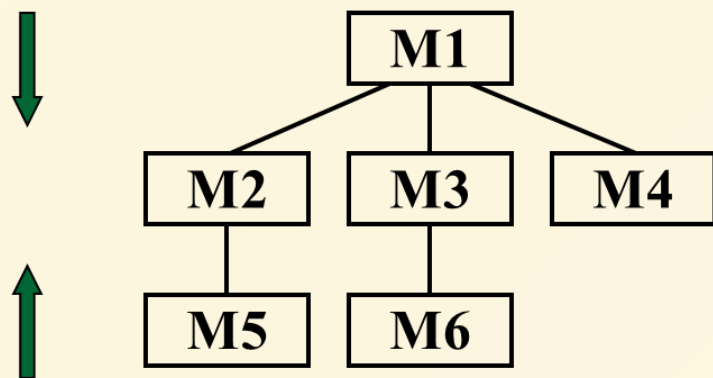
图7.4 自底向上实例

不同集成测试策略的比较

- 自顶向下
 - 优点：不需要驱动程序，能够在测试阶段的早期实现并验证系统的主要功能，能在早期发现上层模块的接口错误。
 - 缺点：需要存根程序，低层关键模块中的错误发现较晚，早期不能充分展开人力。
- 自底向上
 - 优缺点与自顶向下的优缺点正好相反。

混合策略

- 改进的自顶向下测试方法
 - 基本使用自顶向下方法，但在早期使用自底向上的方法测试少数关键模块。
- 混合法



回归测试 (Regression Testing)

指重新执行已经做过的部分测试。

回归测试用于保证由于调试或其他原因引起的程序变化，不会导致额外错误的测试活动。

7.5 确认测试（验收测试）

- 确认测试的目标：验证软件的有效性。
 - 如果软件的功能和性能符合用户的期待，软件就是有效的。
 - 软件规格说明书是进行确认测试的基础。
- 确认测试的主要特点
 - 某些已经测试过的纯粹技术性的测试项可能不需要再次测试，而对用户特别感兴趣的功能或性能，可能需要增加一些测试；
 - 通常确认测试主要使用实际生产中的数据来进行测试；
 - 确认测试必须有用户的积极参与，甚至以用户为主，可能需要进行一些与用户使用步骤有关的测试。

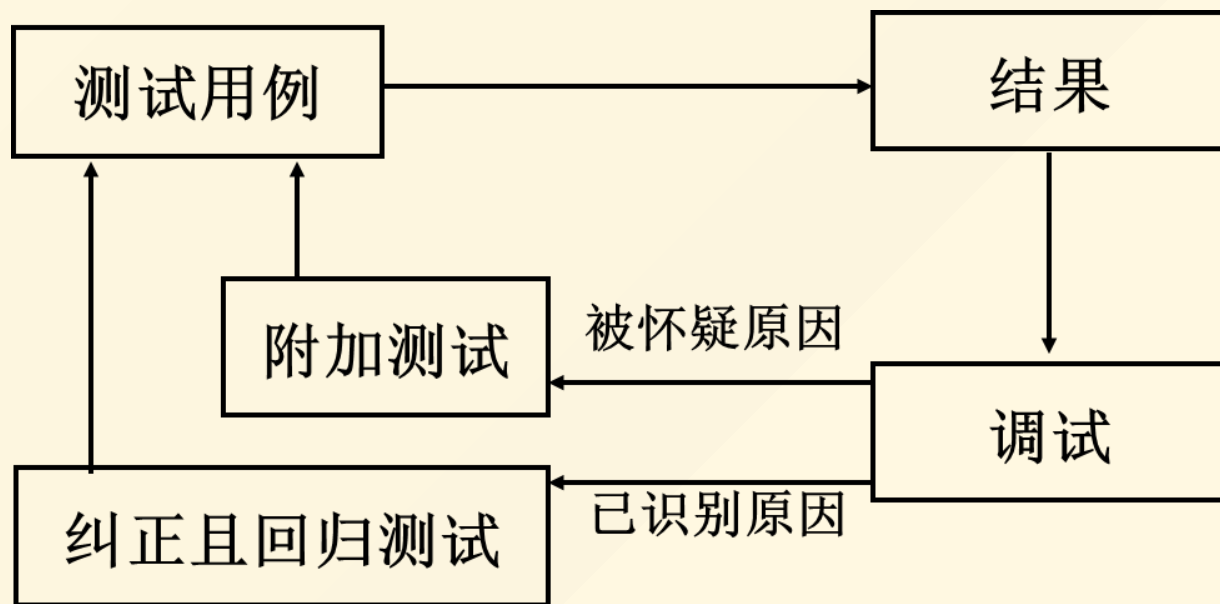
Alpha和Beta测试

- Alpha测试：用户在开发者的场所进行测试，并且在开发者的指导下进行，测试在受控环境中进行，开发者记录发现的错误和问题。
- Beta测试：用户在一个或多个客户场所进行测试，不受开发者控制，测试者记录发现的问题和错误，定期将问题报告发送给开发者。

7.8 调试

调试是在测试发现错误之后排除错误的过程。

调试过程



调试途径

1. 蛮干法

打印内存的内容，从中寻找错误的线索，是效率最低的程序调试方法。

2. 回溯法

从发现问题的程序段开始人工地往回追踪分析程序代码，直到找到错误。

3. 原因排除法

包括：对分查找法、归纳法、演绎法。

7.9 软件可靠性

- 软件可靠性：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。
- 软件可用性：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。
- 可靠性和可用性的区别：可靠性是在0到 t 时间间隔内，系统没有失效的概率。而可用性是在 t 时刻，系统是正常运行的概率。
如果在 t 时刻，系统是可用的，则有两种可能：
 - 1) 在0到 t 时刻这段时间内，系统一直没有失效（可靠）；
 - 2) 在0到 t 时刻这段时间内失效过，但是系统修复后运行到 t 时刻时情况良好。

稳态可用性

如果在一段时间内，软件系统故障停机时间分别为： t_{d1}, t_{d2}, \dots ，正常运行时间分别为： t_{u1}, t_{u2}, \dots ，则系统的稳态可用性定义为：

$$A_{SS} = \frac{T_{up}}{T_{up} + T_{down}}$$

其中 $T_{up} = \sum t_{ui}$, $T_{down} = \sum t_{di}$.

MTTF 和 MTTR

- MTTR（平均维修时间）：修复一个故障平均需要用的时间，取决于维护人员的技术水平和对系统熟悉程度。
- MTTF（平均无故障时间）：系统按照规格说明书规定成功运行的平均时间（即两次故障的平均间隔时间），取决于系统中潜伏的错误数量。
- 引入MTTF和MTTR之后，系统的稳态可用性 A_{SS} 可表示为：

$$A_{SS} = \frac{MTTF}{MTTF + MTTR}$$

估算平均无故障时间MTTF的方法

符号定义

E_T	测试之前程序中故障总数
I_T	程序长度（ 机器指令总数 ）
τ	测试（ 包括调试 ）时间
$E_d(\tau)$	在0至 τ 期间发现的错误数
$E_c(\tau)$	在0至 τ 期间改正的错误数

基本假定

1. 单位长度里的故障数 E_T / I_T 近似为常数。一些统计数字表明，通常有： $0.5 \times 10^{-2} \leq E_T / I_T \leq 2 \times 10^{-2}$ 。
2. 失效率正比于软件中剩余的（潜藏的）故障数，而平均无故障时间 MTTF 与剩余的故障数成反比。
3. 调试过程没有引入新的故障，即 $E_c(\tau) = E_d(\tau)$ 。

估算平均无故障时间MTTF

由于系统剩余的故障数为： $E_r(\tau) = E_T - E_c(\tau)$

所以单位长度程序中剩余的故障数为： $\epsilon_r(\tau) = E_T / I_T - E_c(\tau) / I_T$

因为平均无故障时间与单位长度程序中剩余的故障数 $\epsilon_r(\tau)$ 成反比，所以：

$$MTTF = \frac{1}{K \left[\frac{E_T}{I_T} - \frac{E_c(\tau)}{I_T} \right]}$$

其中： K 为常数，它的值根据经验选取，经典值是200。

需要改正多少错误？

由上式变换后得到程序中改正的错误数：

$$E_c = E_T - \frac{I_T}{K \times MTTF}$$

根据对软件平均无故障时间的要求，可以估计需要改正多少个错误后，测试工作就可以结束。

估计故障总数ET的方法

1. 植入故障法
2. 分别测试法

1. 植入故障法

假设人为地植入的故障数为 N_s ，经过一段时间的测试之后发现 n_s 个植入的故障，同时还发现了 n 个原有的故障，则可以估计出程序中原有的故障总数：

$$N' = \frac{n}{n_s} N_s$$

其中 N' 是故障总数 E_T 的估计值。

植入错误法人为植入的错误与原有程序错误可能性质很不相同，发现它们的难度也不同，用此法估计的错误数有时可能不太准确。

2. 分别测试法

分别测试法随机把程序中一部分原有错误加上标记，根据测试发现的有标记和无标记错误的比例，估计程序错误总数。

分别测试法使用两个测试员，独立地测试同一个程序的两个副本，由另一名分析员分析他们的测试结果，把其中一个测试员发现的故障作为有标记的故障。

用 τ 表示测试时间，假设

$\tau = 0$ 时故障总数为 B_0 （即 E_T ）；

$\tau = \tau_1$ 时测试员甲发现的故障数为 B_1 ；

$\tau = \tau_1$ 时测试员乙发现的故障数为 B_2 ；

$\tau = \tau_1$ 时两个测试员发现的相同故障数为 b_c 。

2. 分别测试法（续）

如果认为测试员甲发现的故障是有标记的，即程序中有标记的故障总数为 B_1 ，那么测试员乙发现的 B_2 个故障中有 b_c 个是有标记的。所以可以估计出测试前程序中的故障总数为：

$$B'_0 = \frac{B_2}{b_c} B_1$$

其中， B'_0 是故障总数 E_T 的估计值。

每隔一定时间，分析员分析两名测试员的测试结果，来估计错误总数。几次估计结果差不多时，用其平均值作为错误总数的估计值。

本章重点

- 测试的目标和准则
- 白盒测试和黑盒测试的含义
- 测试的步骤和各种集成策略
- 软件可靠性