6.5 调试

- 软件调试是在进行了成功的测试之后才开始的工作。它与软件测试不同，调试的任务是进一步诊断和改正程序中潜在的错误。
- 调试活动由两部分组成：
  1.确定程序中可疑错误的确切性质和位置。
  2.对程序（设计，编码）进行修改，排除这个错误。
- 调试工作是一个具有很强技巧性的工作。

- 软件运行失效或出现问题，往往只是潜在错误的外部表现，而外部表现与内在原因之间常常没有明显的联系。如果要找出真正的原因，排除潜在的错误，不是一件易事。
- 调试是通过现象，找出原因的一个思维分析的过程。

6.5.1 调试的步骤

① 从错误的外部表现形式入手，确定程序中出错位置；
② 研究有关部分的程序，找出错误的内在原因；
③ 修改设计和代码，以排除这个错误；
④ 重复进行暴露了这个错误的原始测试或某些有关测试。

- 从技术角度来看，查找错误的难度在于：
  - ✓现象与原因所处的位置可能相距甚远。
  - ✓当其他错误得到纠正时，这一错误所表现出的现象可能会暂时消失，但并未实际排除。
  - ✓现象实际上是由一些非错误原因（例如，舍入不精确）引起的。

- ✓ 现象可能是由于一些不容易发现的人为错误引起的。
- ✓ 错误是由于时序问题引起的，与处理过程无关。
- ✓ 现象是由于难于精确再现的输入状态（例如，实时应用中输入顺序不确定）引起。
- ✓ 现象可能是周期出现的。在软、硬件结合的嵌入式系统中常常遇到。

6.5.2 几种主要的调试方法

- 调试的关键在于推断程序内部的错误位置及原因。可以采用以下方法：

1)强行排错

这种调试方法目前使用较多，效率较低。它不需要过多的思考，比较省脑筋。例如：

a.通过内存全部打印来调试，在这大量的数据中寻找出错的位置。

b.在程序特定部位设置打印语句，把打印语句插在出错源程序的各个关键变量改变部位、

重要分支部位、子程序调用部位，跟踪程序的执行，监视重要变量的变化。

- ◆ 自动调试工具。利用某些程序语言的调试功能或专门的交互式调试工具，分析程序的动态过程，而不必修改程序。
- ✓ 应用以上任一种方法之前，都应当对错误的征兆进行全面彻底的分析，得出对出错位置及错误性质的推测，再使用一种适当的调试方法来检验推测的正确性。

2) 回溯法调试

- ✓ 这是用于小程序一种有效的调试方法。
- ✓ 一旦发现了错误，人们先分析错误征兆，确定最先发现“症状”的位置。然后，人工沿程序的控制流程，向回追踪源程序代码，直到找到错误根源或确定错误产生的范围。
- ✓ 例如，程序中发现错误处是某个打印语句。通过输出值可推断程序在这一点上变量的值。再从这一点出发，回溯程序的执行过程，…，直到找到错误的位置。

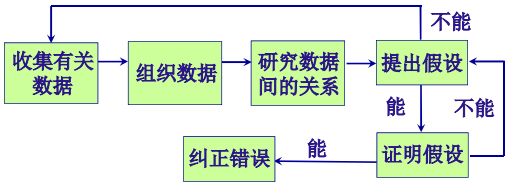
3) 对分查找法

基本思路：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。如果输出结果是正确的，则错误原因在程序的前半部分；反之，错误原因在程序的后半部分。

对错误原因所在的那部分再重复使用这个方法，直到把出错范围缩小到容易诊断的程度为止。

4) 归纳法调试

- ✓ 归纳法是一种从特殊推断一般的系统化思考方法。归纳法调试的基本思想：从一些线索（错误征兆）着手，通过分析它们之间的关系来找出错误。



a) 收集有关的数据　列出所有已知的测试用例和程序执行结果。看哪些输入数据的运行结果正确，哪些输入数据的运行结果有错误。

b) 组织数据　归纳法是从特殊到一般的推断过程，所以需要组织整理数据以发现规律。

常以3W1H形式组织可用的数据：

- ➢ “what” 列出一般现象；
- ➢ “where” 说明发现现象的地点；
- ➢ “when” 列出现象发生时所有已知情况；
- ➢ “how” 说明现象的范围和量级；

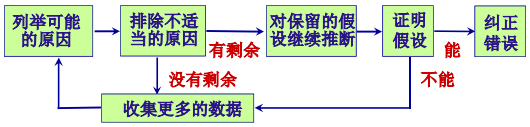
c) 提出假设　分析线索之间的关系，利用在线索结构中观察到的矛盾现象，设计一个或多个关于出错原因的假设。如果一个假设也提不出来，归纳过程就需要收集更多的数据。此时，应当再设计与执行一些测试用例，以获得更多的数据。

d) 证明假设　把假设与原始线索或数据进行比较，若它能完全解释一切现象，则假设得到证明；否则，就认为假设不合理，或不完全，或是存在多个错误，以致只能消除部分错误。

5) 演绎法调试

✓演绎法是一种从一般原理或前提出发，经过排除和精化的过程来推导出结论的思考方法。

✓首先根据已有的测试用例，设想出所有可能出错的原因；然后再用原始测试数据或新的测试，从中逐个排除不可能正确的假设；最后，再用测试数据验证余下的假设确是出错的原因。



a) 列举所有可能出错原因的假设 把所有可能的错误原因列成表。通过它们，可以组织、分析现有数据。

b) 利用已有的测试数据，排除不正确的假设 仔细分析已有的数据，寻找矛盾，力求排除前一步列出所有原因。如果所有原因都被排除了，则需要补充一些数据(测试用例)，以建立新的假设。

c) 改进余下的假设 利用已知的线索，进一步改进余下的假设，使之更具体化，以便可以精确地确定出错位置。

d) 证明余下的假设

6.5.3 调试原则

- 在调试方面，许多原则本质上是心理学方面的问题。
- 确定错误的性质和位置的原则
  - ✓ 分析思考与错误征兆有关的信息。
  - ✓ 避开死胡同。只把调试工具当做辅助手段来使用。利用调试工具，可以帮助思考，但不能代替思考。
  - ✓ 避免用试探法，最多把它当做最后手段。

- 修改错误的原则
  - ✓ 在出现错误的地方很可能还有别的错误。
  - ✓ 修改错误的一个常见失误是只修改了这个错误的征兆或这个错误的表现，而没有修改错误的本身。
  - ✓ 当心修正一个错误的同时有可能会引入新的错误。
  - ✓ 修改错误的过程将迫使人们暂时回到程序设计阶段。
  - ✓ 修改源代码程序，不要改变目标代码。

6.6 软件可靠性

1. 基本概念

- 测试阶段的根本目标是消除错误，保证软件可靠性。用户通常关注软件可以使用的程度。
- 软件可靠性的一般定义(MTTF)：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。
- 软件可用性定义：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。

- 随着运行时间的增加，运行时出现程序故障的概率也将增加，即可靠性随着给定的时间间隔的加大而减少。
- 区分“故障”与“错误”：
  - ❑ “错误”的含义是由开发人员造成的软件差错
  - ❑ “故障”的含义是由错误引起的软件的不正确行为。
- 可靠性和可用性之间的主要差别：可靠性意味着在0到t这段时间间隔内系统没有失效；可用性只意味着在时刻t，系统是正常运行的。

2. 估算软件可靠性(MTTF)

- 软件的平均无故障时间MTTF(Mean Time To Failure)是一个重要的质量指标。
- 程序中潜藏的错误的数目是一个十分重要的量，它既直接标志软件的可靠程度，又是计算软件平均无故障时间的重要参数。
- 程序中的错误总数E_T与程序规模I_T、类型、开发环境、开发方法论、开发人员的技术水平和管理水平等都有密切关系。

估算错误总数的方法

(1) 植入错误法

- 基本思路：在测试之前由专人在程序中随机地植入一些错误，测试之后，根据测试小组发现的错误中原有的和植入的两种错误的比例，来估计程序中原有错误的总数E_T。

- 人为地植入的错误数为N_s，经过一段时间的测试之后发现n_s个植入的错误，此外还发现了n个原有的错误。如果可以认为测试方案发现植入错误和发现原有错误的能力相同，则能够估计出程序中原有错误的总数：

$$n/N\hat{}= n_s/N_s \Rightarrow N\hat{}=n/n_s \times N_s$$

$N\hat{}$为错误总数E_T的估计值。

(2) 分别错误法

- 基本思路：为了消除植入错误法的基本假定带来的误差，对程序中的原有错误加标记，然后根据测试过程中发现的有标记错误和无标记错误的比例，来估计程序中原有错误的总数E_T。
- 具体说来，为了随机地给一部分错误加标记，两个测试员彼此独立地测试同一个程序的两个副本，把其中一个测试员发现的错误作为有标记的错误。

- 用 τ 表示测试时间，假设
  - τ =0时错误总数为B_0；
  - $\tau = \tau_1$时测试员甲发现的错误数为B_1；
  - $\tau = \tau_1$时测试员乙发现的错误数为B_2；
  - $\tau = \tau_1$时两个测试员发现的相同错误数为b_c。
- 如果将甲发现的错误认为是有标记的，并且乙发现有标记与无标记错误的概率相同，则能估计原有错误的总数为：

$$b_c/B_1 = B_2 / B_0\hat{} \Rightarrow B_0\hat{} = B_2/b_cB_1$$

MTTF的计算公式

- 经验表明，平均无故障时间MTTF与单位长度程序中剩余的错误数成反比，即

$$MTTF=1/[K(E_T/I_T-E_c(\tau)/I_T)]$$

其中：

- ✓ K为常数，典型值是200
- ✓ E_T：测试之前程序中错误总数
- ✓ 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
  - 在类似的程序中，单位长度里的错误数E_T/I_T近似为常数。
  - 失效率正比于软件中剩余的错误数，而平均无故障时间MTTF与剩余的错误数成反比。
  - 假设发现的每一个错误都立即正确地改正了(即调试过程没有引入新的错误)，$E_c(\tau)=E_d(\tau)$ 。
- 根据对软件平均无故障时间的要求，估计需要改正多少个错误之后，测试工作才能结束

$$E_c=E_T-I_T/(K\times MTTF)$$

3. 计算软件可用性

- 如果在一段时间内，软件系统故障停机时间分别为td1，…， tdn，正常运行时间分别为tu1，…，tun，则系统的稳态可用性为：

  Ass=Tup/(Tup+Tdown);Tup=∑tui，Tdown=∑tdi
- 如果引入系统平均无故障时间MTTF和平均维修时间MTTR的概念，则

  Ass=MTTF/(MTTF+MTTR)

  平均维修时间MTTR是修复一个故障平均需要用的时间，它取决于维护人员的技术水平和对系统的熟悉程度，和系统的可维护性有重要关系。

小结

- 实现包括编码和测试两个阶段。编码是把软件设计的结果翻译成用某种程序设计语言书写的程序。编码使用的语言，特别是写程序的风格，对程序质量有相当大的影响。
- 软件测试是保证软件可靠性的主要手段。测试阶段的根本任务是发现并改正软件中的错误。
- 大型软件的测试应该分阶段地进行，分为单元测试、集成测试、确认测试、系统测试和验收测试。软件测试包括利用计算机进行的测试和人工测试。

- 测试方案是测试阶段的关键技术问题，基本目标是选用最少量的高效测试数据，做到尽可能完善的测试，从而尽可能多地发现软件中的问题。
- 白盒测试和黑盒测试是软件测试的两类基本方法。白盒测试技术主要有逻辑覆盖、基本路径测试和控制结构测试；黑盒测试技术主要有等价划分、边界值分析和错误推测。
- 通常，在测试过程的早期阶段主要使用白盒方法，在测试过程的后期阶段主要使用黑盒方法。

- 调试的主要任务是改正测试过程中发现的软件错误。调试包括确定错误的位置与性质和排除错误两个步骤。
- 软件可靠性与可用性是评价软件系统的重要指标。程序中潜藏的错误的数目，直接决定了软件的可靠性。通过测试可以估算出程序中剩余的错误数。根据测试和调试过程中已经发现和改正的错误数，可以估算软件的平均无故障时间。