Chapter 17

- **Software Testing Strategies**

Slide Set to accompany

Software Engineering: A Practitioner's Approach, 7/e

by Roger S. Pressman

17.1 测试的重要性

ARIANE火箭, 耗资**70**亿美元，**1996**年发射**37**秒后爆炸

```
sensor_get(vertical_veloc_sensor);
sensor_get(horizontal_veloc_sensor);
vertical_veloc_bias := integer(vertical_veloc_sensor);
horizontal_veloc_bias := integer(horizontal_veloc_sensor);
...
exception
  when numeric_error => calculate_vertical_veloc();
  when others => use_irs1();
end;
```

17.1 测试的重要性

- 发射失败的原因

程序中试图将64位浮点数转换成16位整数时的溢出错误。

如果看其浮点转换程序，并没有任何问题。问题在于他们复用了Ariane 4的部分软件需求文档，而软件工程师不知道其Ariane 5的水平加速度比Ariane 4快5倍，因此要求额外3位整数存储。

17.2 测试的目的

Grenford J. Myers 就软件测试提出以下观点：

- 测试是选择适当的测试用例并执行被测程序的过程。
- 目的在于发现错误，而不是为了验证程序不存在错误。
- 好的测试用例可以发现今未发现的错误。

17.3 测试策略

- To perform effective testing, you should conduct effective technical reviews.
- Testing begins at the component level and works "outward" toward the integration of the entire computer-based system.
- Different testing techniques are appropriate for different software engineering approaches and at different points in time.
- Testing is conducted by the developer of the software and an independent test group.
- Testing and debugging are different activities, but debugging must be accommodated in any testing strategy.

策略1：测试不能取代评审

据说，检验项目质量的标准之一：

质量 = 项目review时骂的次数 / 项目review时间

被骂的次数越多，评审耗时越短，质量越可靠。

代码缺陷还是设计缺陷？

1）抱怨举例 1

- 为什么操作这么费劲（输入太繁琐）、为什么要走这么多步才能完成一次输入?
- 不用系统工作就够忙的了，结果用了系统更忙了。
- 输入的数据对我的工作没有帮助，都是给领导输入的...等等。

2）抱怨举例 2

- 数据计算有错误、页面切换后 xx 标题就找不到了...
- 经常死机、系统不稳定、输入 xx 数据后就进入了死循环...
- 页面数据链接错了，找不到相关的数据...等等。

策略2：Who Test Software

developer

Understands the system

but, will test "gently"

and, is driven by "delivery"

independent tester

Must learn about the system,

but, will attempt to break it

and, is driven by quality

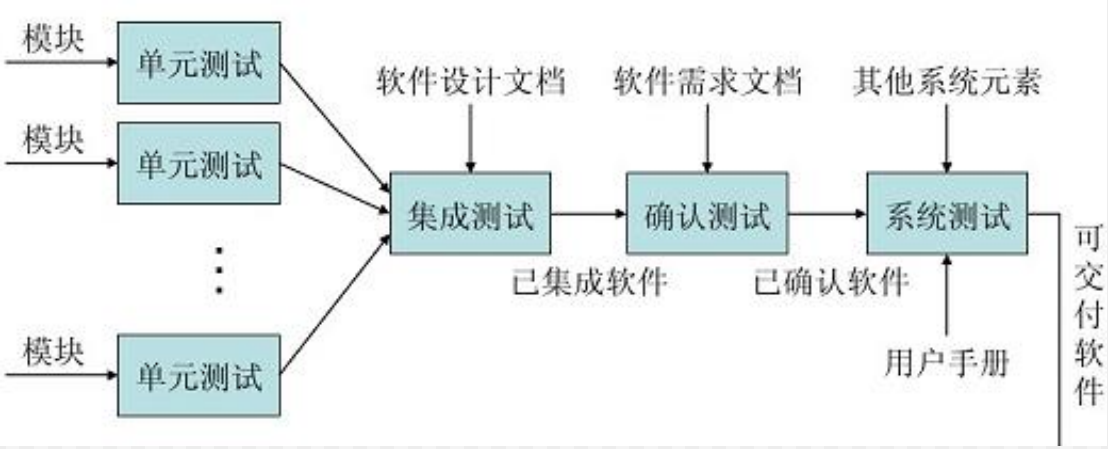
策略3：测试什么时候开始

- 编码阶段开始？？？
- 软件测试从项目确立时就开始了，主要经过以下环节：

  需求分析→测试计划→测试设计→测试环境搭建→测试执行→测试记录。

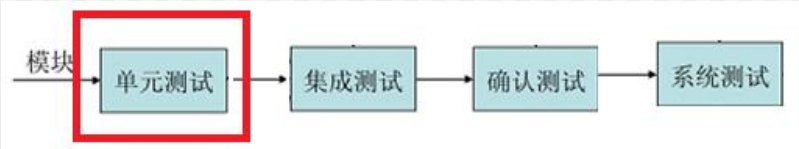
策略4：Testing Process

- We begin by ‘testing-in-the-small’ and move toward ‘testing-in-the-large’

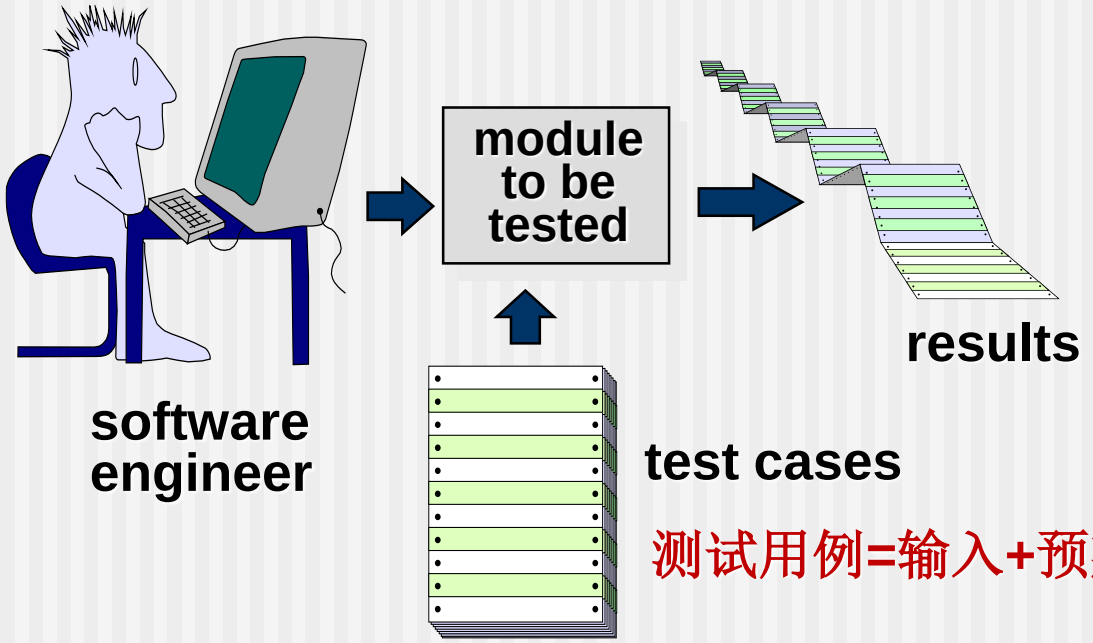


- For conventional software
  - The module (component) is our initial focus
  - Integration of modules follows
- For OO software
  - Our focus when “testing in the small” is OO class

17.4 传统软件的测试策略:单元测试

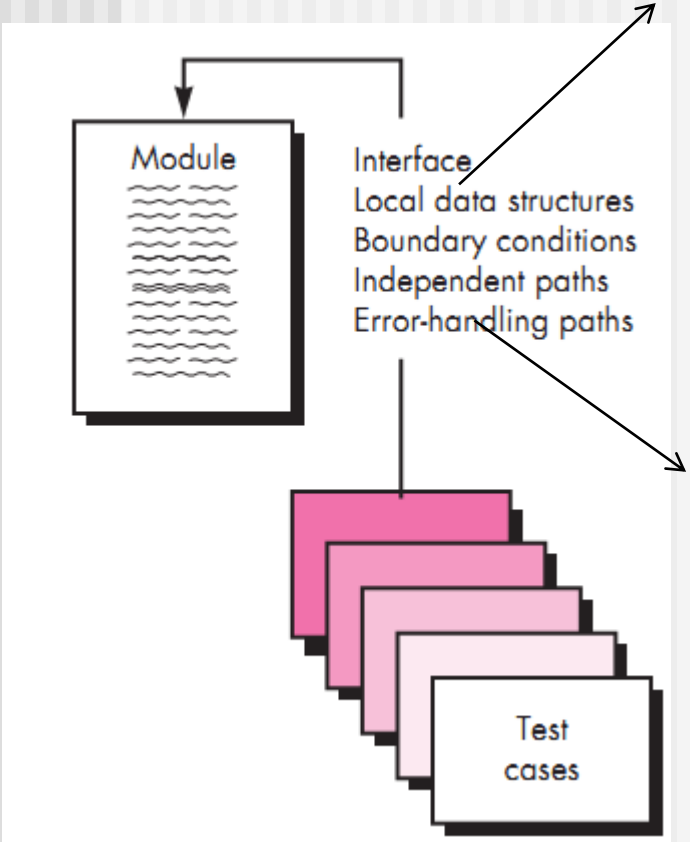


目的：验证开发人员编写的模块或单元是否符合预期。



测试用例=输入+预期的输出

17.4.1 单元测试重点



说明不正确或不一致；
初始化或缺省值错误；
数据类型不相容；
上溢、下溢或地址错误等。
全局数据与模块的相互影响。

```
public static double area(double a,double b,double c) throws triangularExceptio
{
    double s;
    s=(a+b+c)/2;
    if(a+b>c&&a+c>b&&c+b>a)
    {
        return Math.sqrt(s*(s-a)*(s-b)*(s-c));
    }
    else
    {
        throw new triangularException();
    }
```

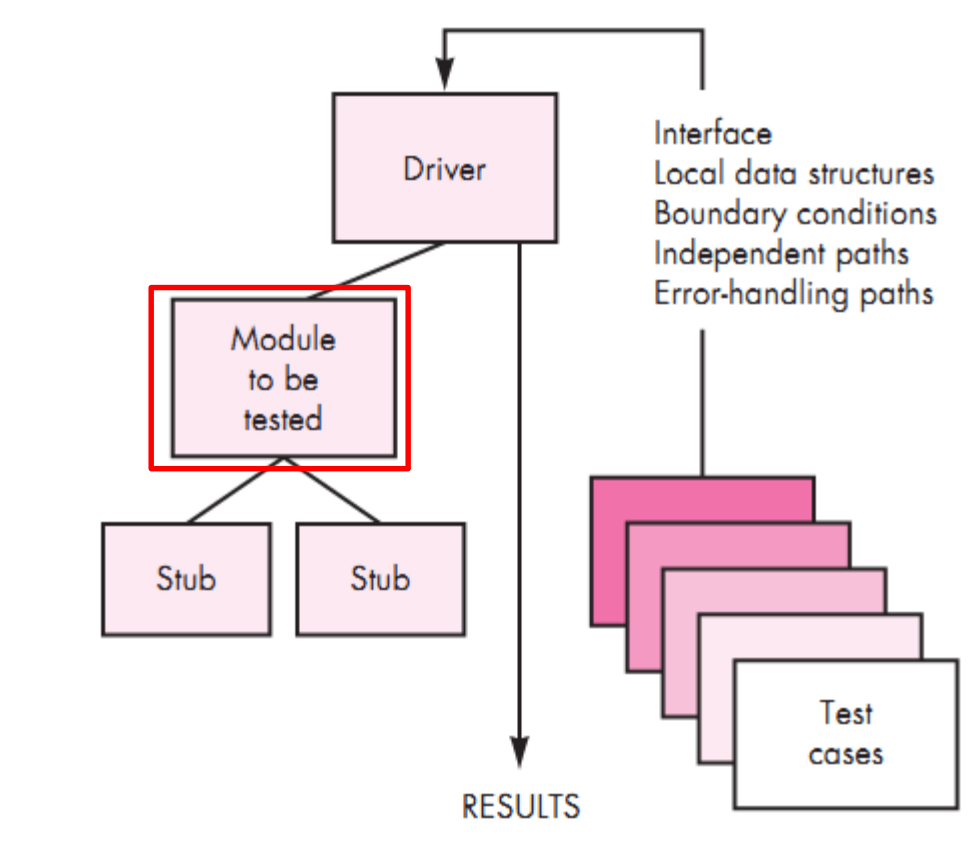
17.4.1 单元测试重点

```
public class Library {
    ….
   private static final int _S_CHARS_PER_LINE = 80;
   private static final int _S_LINES_PER_PAGE = 50;
   private static Set _s_books = new HashSet ();
   public static void addBook (String title, String author, int num_pages)
    {
      int num_chars = num_pages * _S_LINES_PER_PAGE *
       _S_CHARS_PER_LINE;
      _s_books.add (new Book (title, author, new char [num_chars]));
}
```

A memory leak will be reported for the above line because it allocates memory that will never be freed each time this method is called with the same arguments.

内存溢出错误

17.4.2 单元测试环境


Driver
Interface
Local data structures
Boundary conditions
Independent paths
Error-handling paths
Module
to be
tested
Stub
Stub
Test
cases
RESULTS

17.4.2 单元测试环境

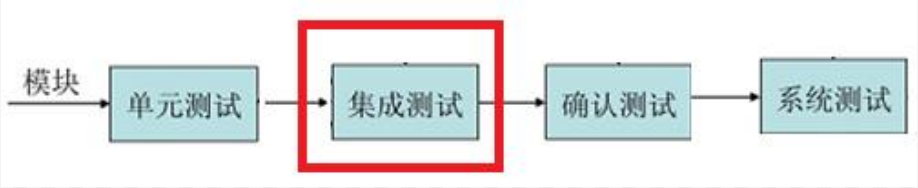
驱动模块：用来模拟被测模块的上级调用模块。

桩模块：模拟被测试模块所调用的模块，返回被测模块所需的信息。

```
public void Test(String args[]){
    String s=opendoc(“d:\\MyBook.doc”);    //假如opendoc的功能是打开文件，并输出该文件的内容
    ……..
}

String stub_opendoc (String s) {                    //编写的桩模块
    //输入参数检查
     ……
    if 参数检查合格 then
        return    预先定义的字符串S1;
    else
       return   “open error” ;
}
```

17.5 传统软件的测试策略：集成测试

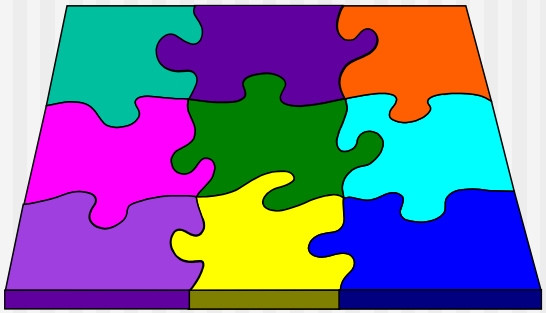


通过单元测试的模块集成在一起，仍然可能无法正常工作，因为模块相互调用时仍然可能出错。

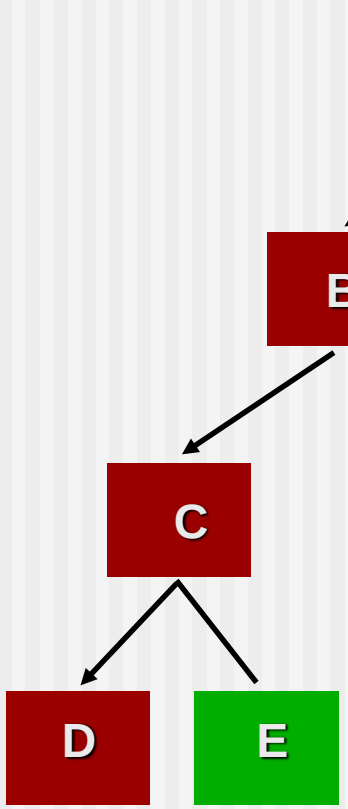
集成测试：旨在发现与调用接口相关的错误。

Options:

- the “big bang” approach
- an incremental construction strategy



17.5.1 自顶向下集成

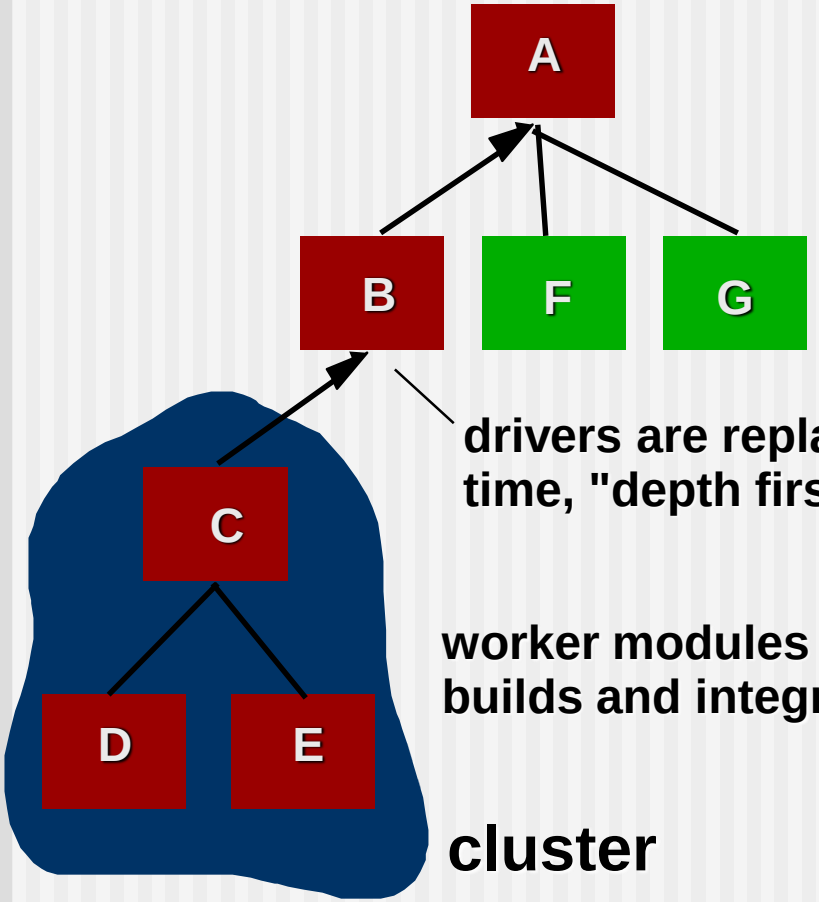


top module is tested with stubs

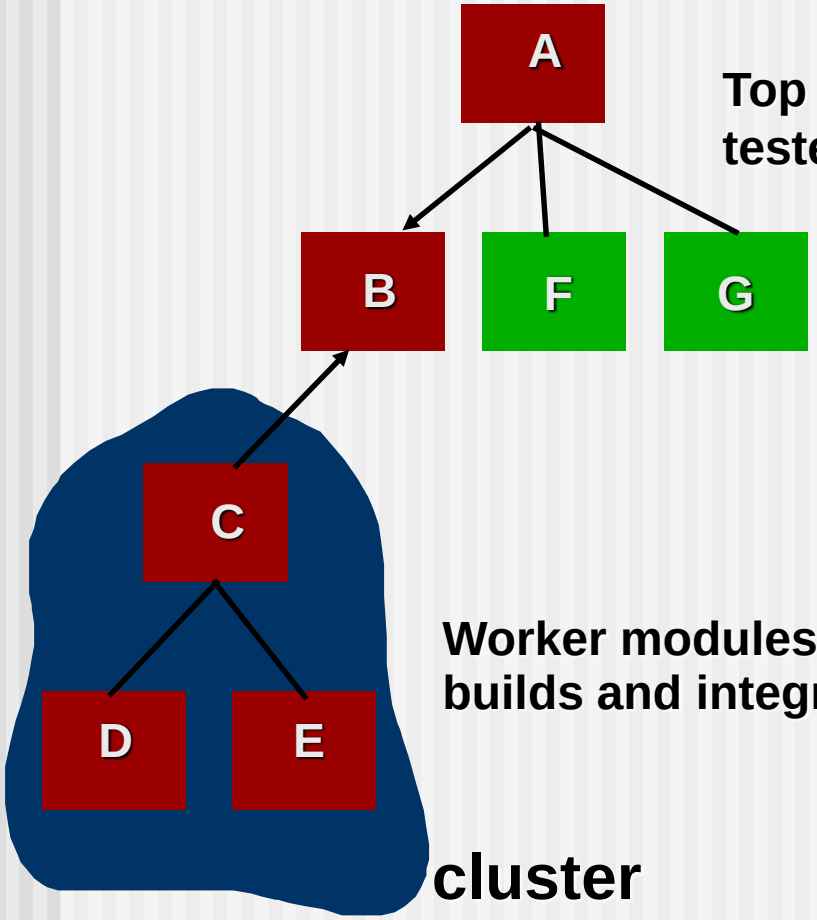
stubs are replaced one at a time, "depth first"

as new modules are integrated, some subset of tests is re-run

17.5.2 自底向上集成


A
B
F
G
C
D
E
drivers are replaced one at a time, "depth first"
worker modules are grouped into builds and integrated
cluster

17.5.3 三明治集成



Top modules are tested with stubs

Worker modules are grouped into builds and integrated

cluster

17.5.4 回归测试

- 集成测试中每加入新的模块，都会导致软件发生变更，这些变更可能使得原本正常工作的模块产生问题。
- 回归测试就是重新执行已经测试过的某些子集，以确保变更不会引入新的或额外的错误。

- 回归测试可以复用之前的测试用例和测试脚本。
- 可以使用自动测试工具提高回归测试效率。

捕获-回放工具（如WinRunner）能够捕获在测试过程中传递给软件的输入数据，生成执行的脚本，并且在回归测试中，重复执行该过程。

17.5.5 冒烟测试

- 冒烟测试也是一种特殊的集成测试，指完成一个新版本的集成、构建后，对该版本最基本的功能进行测试，目的是保证基本的功能和流程能走通。
- 如果通不过，则打回开发部门重新开发；如果通过测试，才会进行下一步的测试(功能测试，集成测试，系统测试等)。
- 例如，冒烟测试内容：
  - 检查被测系统或模块能否正常启动和退出；
  - 数据库能否正常连接，控件能否正常加载；
  - 是否存在严重错误或者数据严重丢失等。

17.6 面向对象的单元与集成测试

- Class testing is the equivalent of unit testing
  - operations within the class are tested
- Integration applied three different strategies
  - 传统的自顶向下或自底向上的集成测试策略在面向对象软件的集成测试中无意义。
  - 基于类之间的协作关系进行集成。
  - thread-based testing— integrates the set of classes required to respond to one input or event
  - use-based testing— begins with testing independent classes， then dependent classes that use the independent classes are tested

17.7 确认测试

- Validation testing 确认测试
  - Focus on software requirements
  - 确认测试：检验所开发的软件是否满足需求规格说明中的各种功能和性能需求，以及软件配置是否完全和正确。
- Alpha/Beta testing
  - 可归为确认测试，产品正式发布前的测试，由用户主导。

α测试是由用户在开发环境下进行的测试；

β测试是由软件的多个用户在实际使用环境下进行的测试

17.8 系统测试

- System testing
  - 将软件与其它系统部分（如外部系统，硬件设计等）结合进行的整体测试。
  - 关注非功能性需求。
- 系统测试主要包括以下几种类型

17.8.1 恢复测试

- 有些系统失效后必须在规定时间段内被恢复，否则将会导致严重的经济损失。
- 如：系统供电故障后的恢复
  日志处理恢复能力
  程序运行崩溃后恢复能力
- 关注导致软件运行失败的各种条件，并验证其恢复过程能否正确执行。

17.8.2 安全测试

- 检测系统对非法侵入的防范能力。如口令截取或破

17.8.3 压力测试

- 压力测试目的是为了发现系统能支持的最大负载等，前提是系统性能处在可以接受的范围内，如规定的3秒钟内响应。
- 压力测试通过不断对系统增加压力，确定一个系统的瓶颈或者不能接受的性能点，从而获得系统最大负载能力。

- 压力测试工具

WebRunner：RadView公司推出的一个性能测试和分析工具，它让web应用开发者自动执行压力测试；webload通过模拟真实用户的操作，生成压力负载来测试web的性能。

17.8.3 性能测试

- 通过测试确定系统(常态下)运行时的性能表现，如运行速度、响应时间、占有系统资源等方面的系统数据。

- 性能测试工具

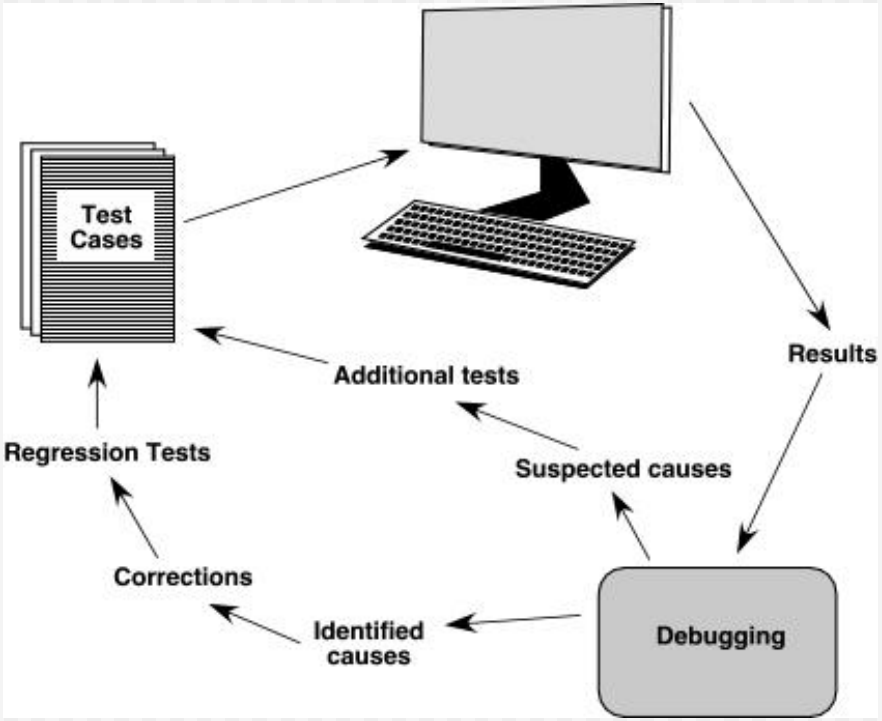
LoadRunner：它能预测系统行为，优化性能。它通过模拟实际用户的操作行为和实行实时性能监测。

17.9 软件调试

- 调试目标：定位错误发生的原因和位置

  如： 客户说我们系统运行界面很卡，死活找不到原因……

工程师赶到现场，给客户换了个鼠标垫，故障排除。



17.9.1 调试技巧

- Brute force
- Backtracking
- Induction(归纳法)
- Deduction(演绎法)

17.9.1 调试技巧

（1）Brute force

一种比较简单，但效率较低的方法。包括：

① 主存信息输出法。

将计算机存储器和寄存器的全部内容打印出来。

② 关键部分设置打印语句。

③ 使用自动调试工具。

17.9.1 调试技巧

（2）Backtracking

反向跟踪是从可能的错误征兆处逐步向回追溯，直到找准错误，纠正为止。

17.9.1 调试技巧

（3）Induction(归纳法)

从线索（错误征兆出发），通过分析这些线索之间的关系找出故障。主要有4步：

① 收集数据。列出已知的测试用例和执行结果。

② 整理分析数据，以便发现规律，即“什么条件下出现错误，什么条件下不出现错误”。

如：字符计数程序对“输入型”文本正确，对“拷贝型”文本错误。

③ 导出假设。分析线索之间的关系，提出关于错误的一个或多个假设。

④ 证明假设。证实提出的假设是导致错误发生的真正原因

17.9.1 调试技巧

（4） Deduction(演绎法）

① 设想出所有可能的出错原因；

② 试图用测试来排除每一个假设的原因，如果测试表明某个假设的原因可能是真的原因，则对数据进行细化以确定位错误。