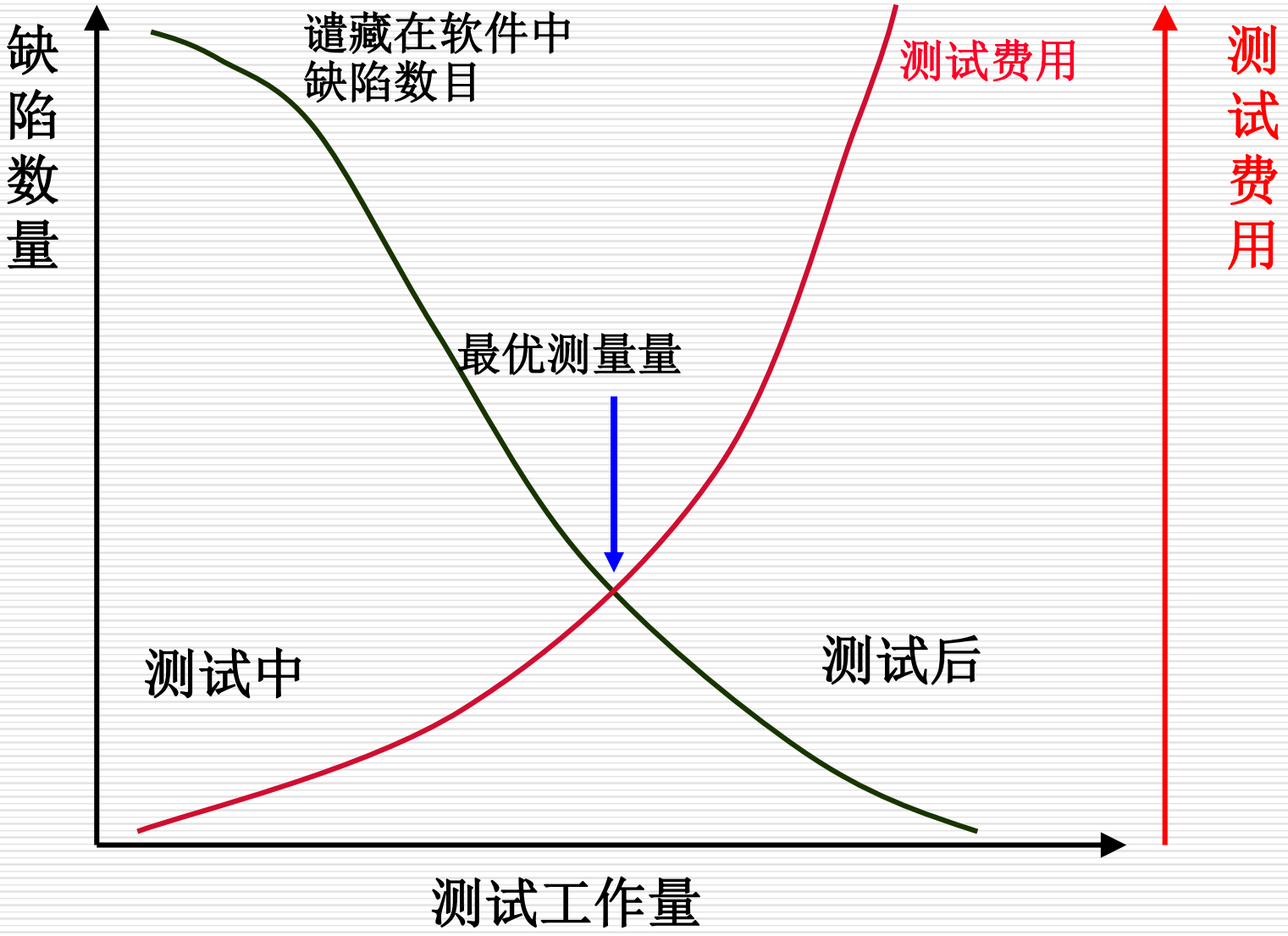
CHAPTER 7

Software Testing

Outline

- When to stop testing ?
- Software testing Vs. Reliability
- Software debugging

软件测试是有风险的行为



每一个软件项目都有一个最优的测量

停止测试考虑的因素

- ✓ **Testing is a trade-off between budget, time and quality.**
- ✓ **It is driven by profit models.**
- ✓ **pessimistic→ time, budget, or test cases -- are exhausted.**
- ✓ **optimistic→ reliability meets the requirement, or the benefit from continuing testing cannot justify the testing cost.**

When to stop testing ?

- ✓ deadline is reached
- ✓ budget has been consumed
- ✓ test plan has been completed
- ✓ if x expected defects has been detected
- ✓ average cost per defect has reached a certain limit
- ✓ if in the last *n* days
  - no defect, no bug
  - less than *x* defects, bugs
  - no severe defect has been found

国内某公司软件测试停止标准

1. 软件系统经过单元、集成、系统测试，分别达到单元、集成、系统测试停止标准。
2. 软件系统通过验收测试，并已得出验收测试结论。
3. 软件项目需暂停以进行调整时，测试应随之暂停，并备份暂停点数据。
4. 软件项目在其开发生命周期内出现重大估算，进度偏差，需暂停或终止时，测试应随之暂停或终止，并备份暂停或终止点数据。

国内某公司软件测试停止标准

- 单元测试停止标准：

✓单元测试用例设计已经通过评审

✓按照单元测试计划完成了所有规定单元的测试

✓达到了测试计划中关于单元测试所规定的覆盖率的要求

✓被测试的单元每千行代码必须发现至少 3 个错误

✓软件单元功能与设计一致

✓在单元测试中发现的错误已经得到修改，各级缺陷修复率达到标准

国内某公司软件测试停止标准

➢ 集成测试停止标准：

- ✓ 集成测试用例设计已经通过评审
- ✓ 按照集成构件计划及增量集成策略完成了整个系统的集成测试
- ✓ 达到了测试计划中关于集成测试所规定的覆盖率的要求
- ✓ 被测试的集成工作版本每千行代码必须发现 2 个错误
- ✓ 集成工作版本满足设计定义的各项功能、性能要求
- ✓ 在集成测试中发现的错误已经得到修改，各级缺陷修复率达到标准

国内某公司软件测试停止标准

- 系统测试停止标准：
  - ✓ 系统测试用例设计已经通过评审
  - ✓ 按照系统测试计划完成了系统测试
  - ✓ 达到了测试计划中，测试所规定的覆盖率的要求
  - ✓ 被测试的系统每千行代码必须发现 1 个错误
  - ✓ 系统满足需求规格说明书的要求
  - ✓ 在系统测试中发现的错误已经得到修改，各级缺陷修复率达到标准

国内某公司软件测试停止标准

- 缺陷修复率标准：
  - ✓ 一、二级错误修复率应达到100%（是否应该对一、二、三级错误进行定义？）
  - ✓ 三、四级错误修复率应达到80%以上
  - ✓ 五级错误修复率应达到60%以上
- 覆盖率标准：
  - ✓ 语句覆盖率最低不能小于80%
  - ✓ 测试用例执行覆盖率应达到100%
  - ✓ 测试需求覆盖率应达到100%

测试中的可靠性分析

- 定义：

利用测试的统计数据来估算软件的可靠性，以控制软件的质量。

- ✓ 推测错误的产生频度
- ✓ 推测残留在程序中的错误数
- ✓ 评价测试的精确度和覆盖率

What is Software Quality?

- **Basic definition**
  - **meeting the users' needs**
  - **needs, not wants**
  - **true functional needs are often unknowable**
- **There is a hierarchy of needs**
  - **do the required tasks**
  - **meet performance requirements**
  - **be usable and convenient**
  - **be economical and timely**
  - **be dependable and reliable**

- **Functionality** ⇒functional testing
  - ✓ suitability accuracy, security, compliance, interoperability
- **Reliability** ⇒reliability testing
  - ✓ maturity, fault tolerance, recoverability
- **Usability** ⇒usability testing
  - ✓ understandability, learnability, operability
- **Efficiency** ⇒performance testing
  - ✓ time behaviour, resource utilization
- **Maintainability** ⇒maintainability testing
  - ✓ Analysability, changeability, stability, testability
- **Portability** ⇒portability testing ?
  - ✓ Adaptability, intallability, conformance, replaceability

测试要素

- 一致性：确保最终设计和用户需求完全一致
- 可靠性：在规定的时间内都可以正常运转。
- 易于使用：多数人均感觉易于使用。
- 可维护性：可以很容易的定位问题，并且进行修改。
- 可移植性：数据或者程序易于移至到其它系统上。
- 耦合性：系统中的组件可以很容易的联接。
- 性能：系统资源的占用率，响应时间，并发处理
- 操作性：易于操作（Operator）

测试要素，继续……

测试要素	压力	执行	恢复	操作	完整性	安全性	需求	回归	错误处理	手工支持	系统兼容	管理	平行	单元
服务水平	√	√		√										
权限控制						√								
一致性						√	√				√			√
正确性							√	√	√	√	√	√	√	√
易用性					√		√			√				√
可维护性					√									√
兼容性				√	√									
耦合性				√						√	√			
性能	√	√			√									√
可操作性				√	√									

可靠性和可用性的量化计算

- 定义：**reliability of software**

程序在给定的时间间隔内，按照规格说明书的规定，成功运行的概率。

- 定义：**usability of software**

程序在给定的时间点，按照规格说明书的规定，成功运行的概率。

- 公式

$A_{use} = T_{up} / (T_{up} + T_{down})$

$A_{use} = MTTF / (MTTF + MTTR)$

MTTF: 平均无故障时间， **MTTR**：平均修复时间

MTTF 的计算

Some notions:

E_T : total error before testing

I_T : size of program to be tested

τ : time used by testing

$E_d(\tau)$: found errors in $[0, \tau]$

$E_c(\tau)$ corrected errors in $[0, \tau]$

通过测试数据计算MTTF

- 两个基本假定：

(1) $0.005 <= E_T / I_T <= 0.02$

(2) MTTF ∝ 1/ hidden bugs

- 计算公式:

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

其中： K=200

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

Testing stop rule

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

举例：假设

$E_T = 3000$

$I_T = 10000$

MTTF=0.5 hour

k=200

Ec=2000

E_T ?

Predicting total errors (ET)

- 植入法

N_p : planting into errors on purpose before testing

n_p : found errors within N_p

n : found new errors

N: predicting total errors

$$N/n = N_P/n_p \ , \quad N = N_p \times n/n_p$$

An example

环境研究学者想调查一山区金丝猴种群的数量。第一在该山区捕捉到50只金丝猴， 将它们系上铁环标志后，全部放归山中。过了一段时间，进行第二次捕捉，共捕捉到100只金丝猴，其中系有铁环标志为10只。问该山区可能有多少只金丝猴？

Predicting total errors (ET)

- 分别测试法：区分标记故障和非标记故障

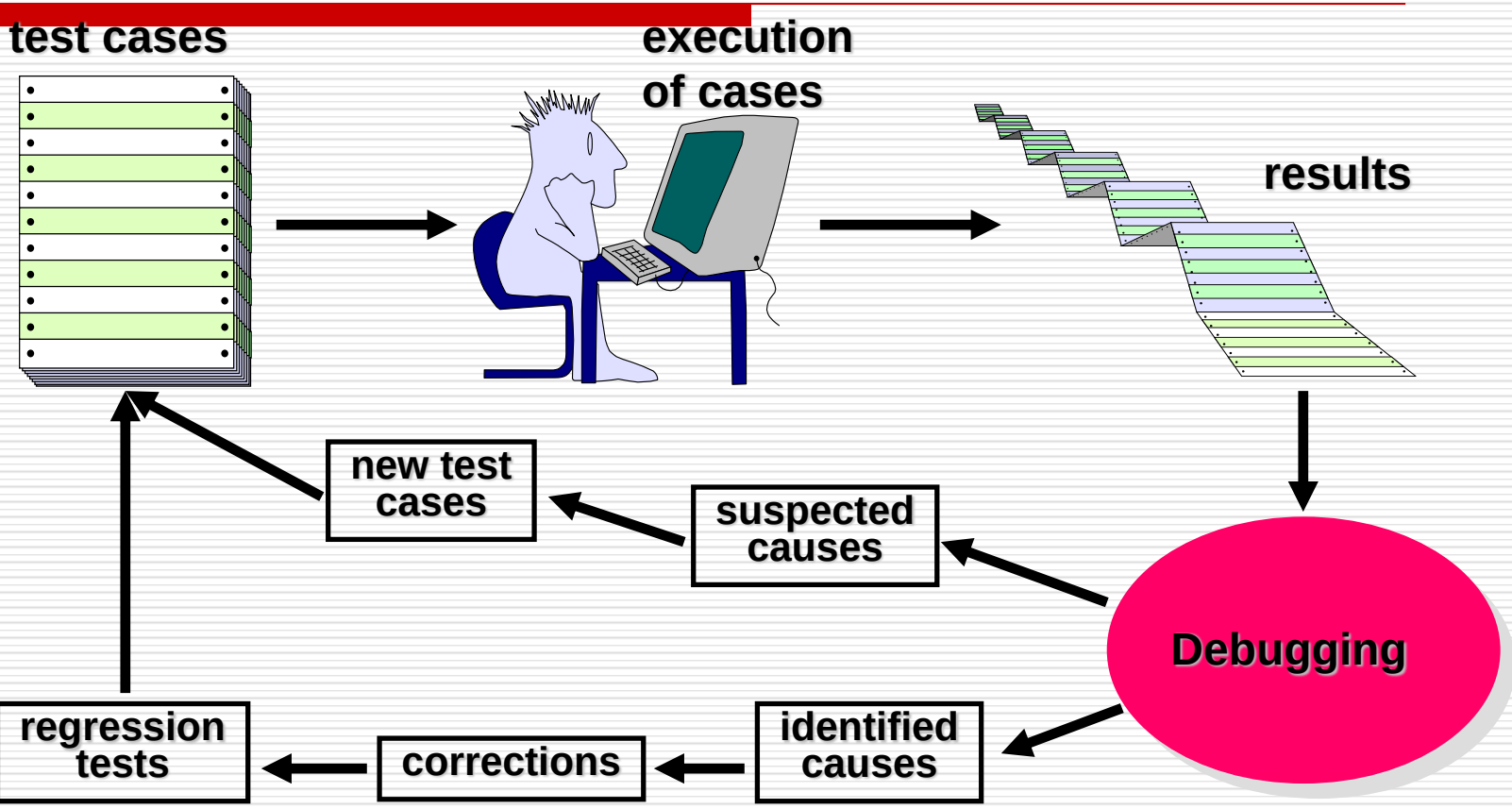
N_1 : found errors by person 1 (标记故障）

N_2 : found errors by person 2 （非标记故障，潜在故障）

n_b : found errors by both person1 and person 2

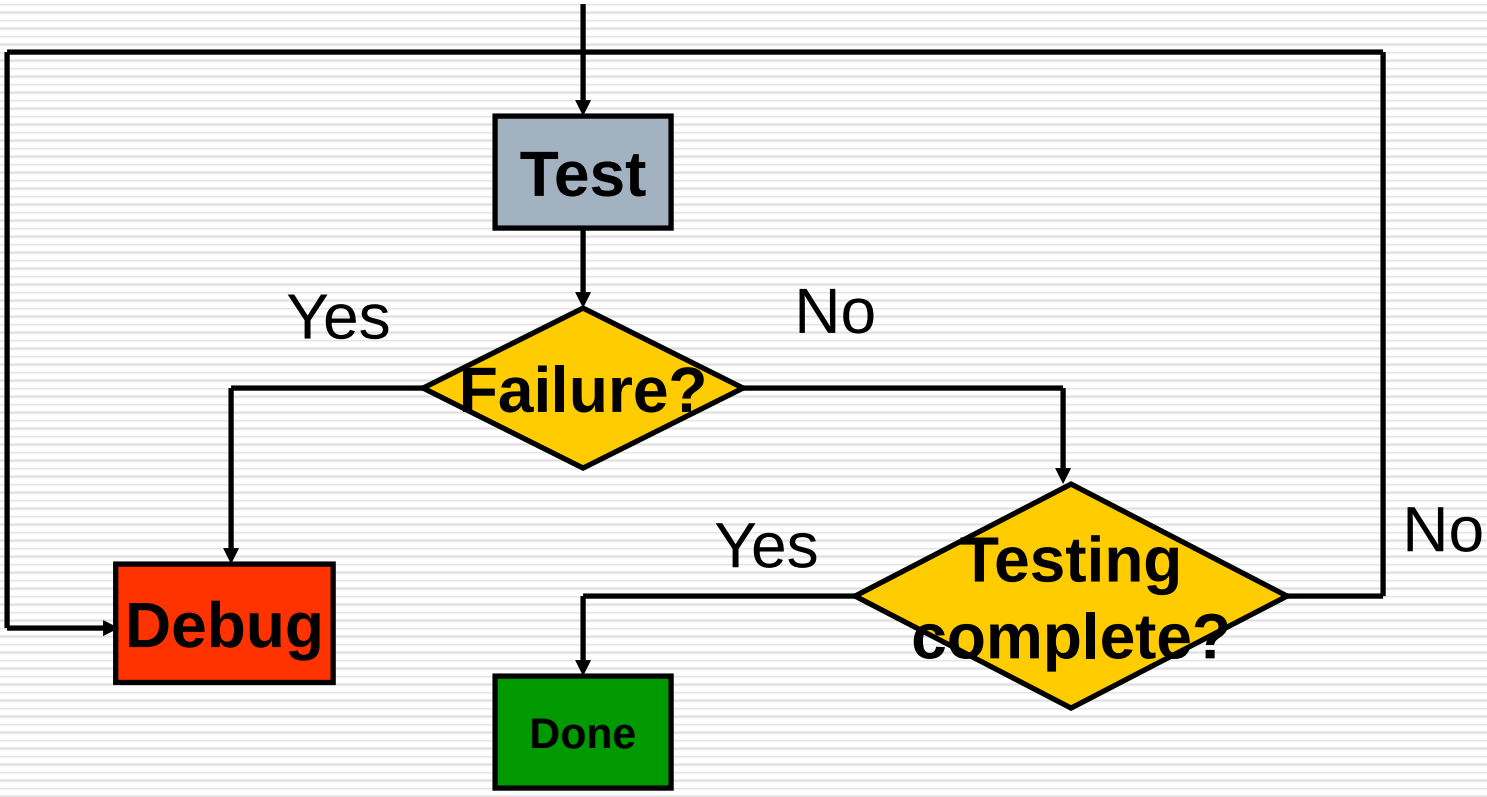
$$N = N_1 \times N_2 / n_b$$

Software debugging



- ✓ Testing is such a process that identifies an error's "symptoms"
- ✓ Debugging is a diagnostic process that identifies an error's "cuase"

Test-debug cycle



Debugging

- 软件调试是在进行了成功的测试之后才开始的工作.
- 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：性质原因和位置,修改排除这个错误
- 调试工作是一个具有很强技巧性和经验性的工作
- 调试是通过现象，找出原因的一个思维分析的过程
- 通过debuger工具来进行
- Most integrated development environments, such as JBuilder, include a debugger.

几种主要的调试方法

- 强行排错法

- ✓ 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。
- ✓ 在程序特定部位设置打印语句，把打印语句插在出错的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。

调试方法

- 回溯法调试

这是在小程序中常用的一种有效的调试方法。

一旦发现了错误，先分析错误征兆，确定最先发现“症状”的位置。

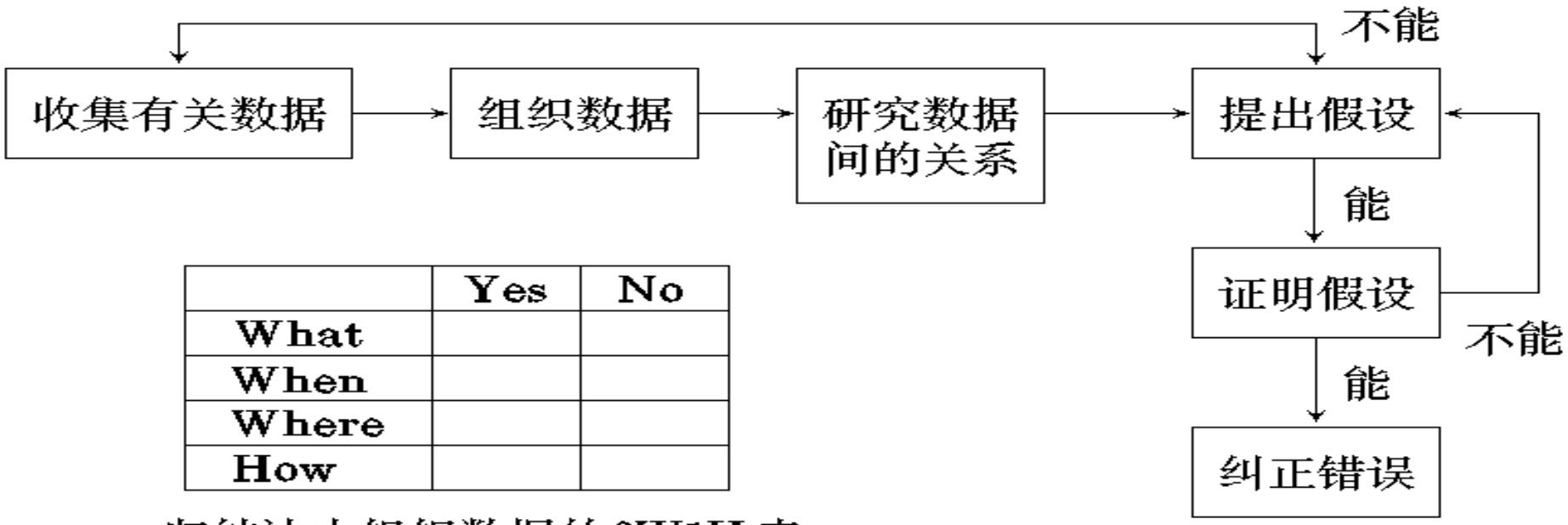
然后，人工沿程序的控制流程，向回追踪源程序代码，直到找到错误根源或确定错误产生的范围。

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，反复考虑：“如果程序在这一点上的状态（变量的值）是这样，那么程序在上一点的状态一定是这样...”， 直到找到错误的位置。

调试方法

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是：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。



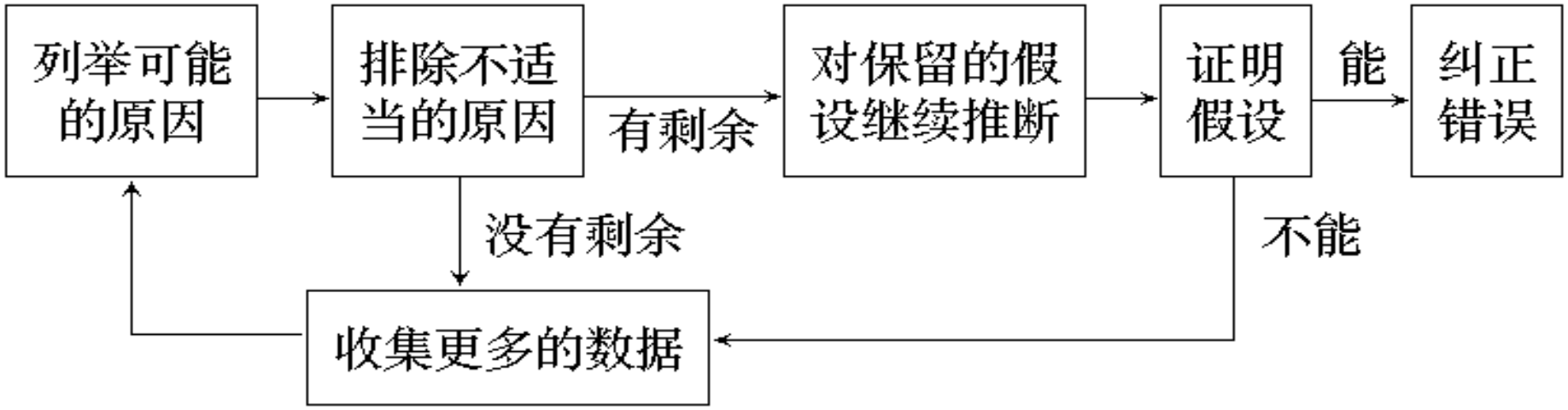
	Yes	No
What		
When		
Where		
How		

归纳法中组织数据的3W1H 表

调试方法

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。演绎法排错是测试人员首先根据已有的测试用例，设想及枚举出所有可能出错的原因做为假设；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。



Homework 2024-12-02

Page 184

T4

T5