软件测试

- 测试程序以发现程序中的缺陷

目标

- 多个用来发现程序缺陷的技术
- 组件接口测试的方法指南
- 面向对象系统的组件测试和集成测试
- 用于支持测试的CASE工具工作原理

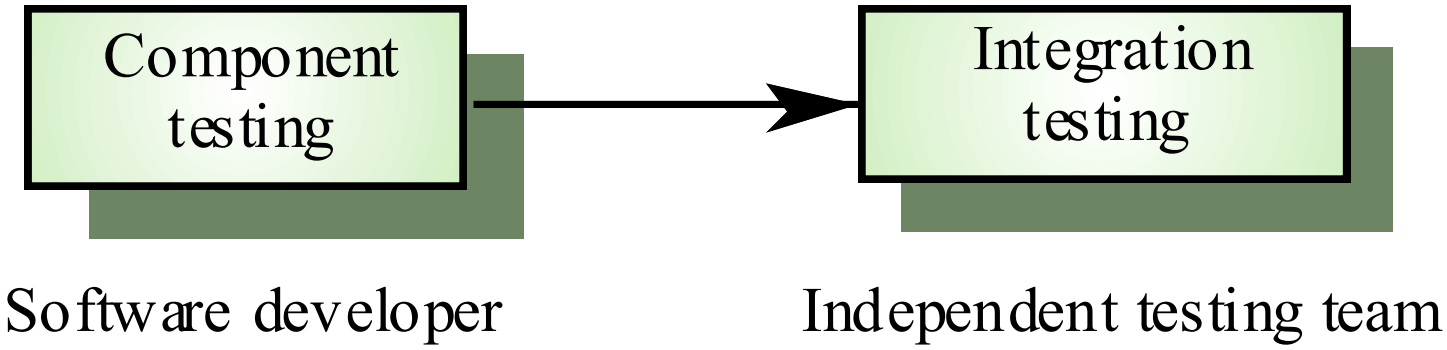
内容

- 缺陷测试
- 集成测试
- 面向对象的测试
- 测试工作平台

测试过程

- 组件测试
  - 个别的程序组件的测试
  - 通常是组件开发者负责(除了某些苛求系统)
  - 测试基于开发者的经验
- 集成测试
  - 对一个系统或子系统进行测试
  - 独立的测试团队负责
  - 测试基于系统描述

测试的阶段


Component testing
Integration testing
Software developer
Independent testing team

缺陷测试

- 缺陷测试的目标是发现程序中的缺陷
- 成功的缺陷测试是引起程序异常动作的测试
- 测试可以证明缺陷的存在但不能证明缺陷不存在。

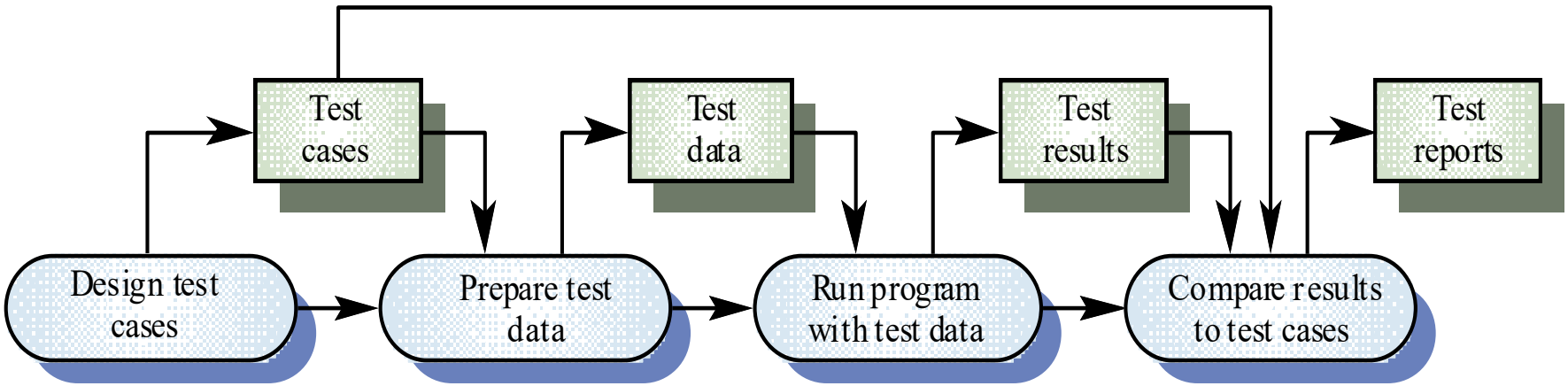
测试的优先级

- 只有穷尽的测试才能证明一个程序没有缺陷。但是穷尽的测试是不可能的。
- 相比组件，测试更应关注系统的功能和性能
- 相比新的功能，测试旧的功能更加重要
- 相比边界值案例，测试典型值情形更加重要

测试数据和测试用例

- *测试数据* 设计用来测试系统的输入数据
- *测试用例* 测试系统的输入，以及根据系统描述预期的输出

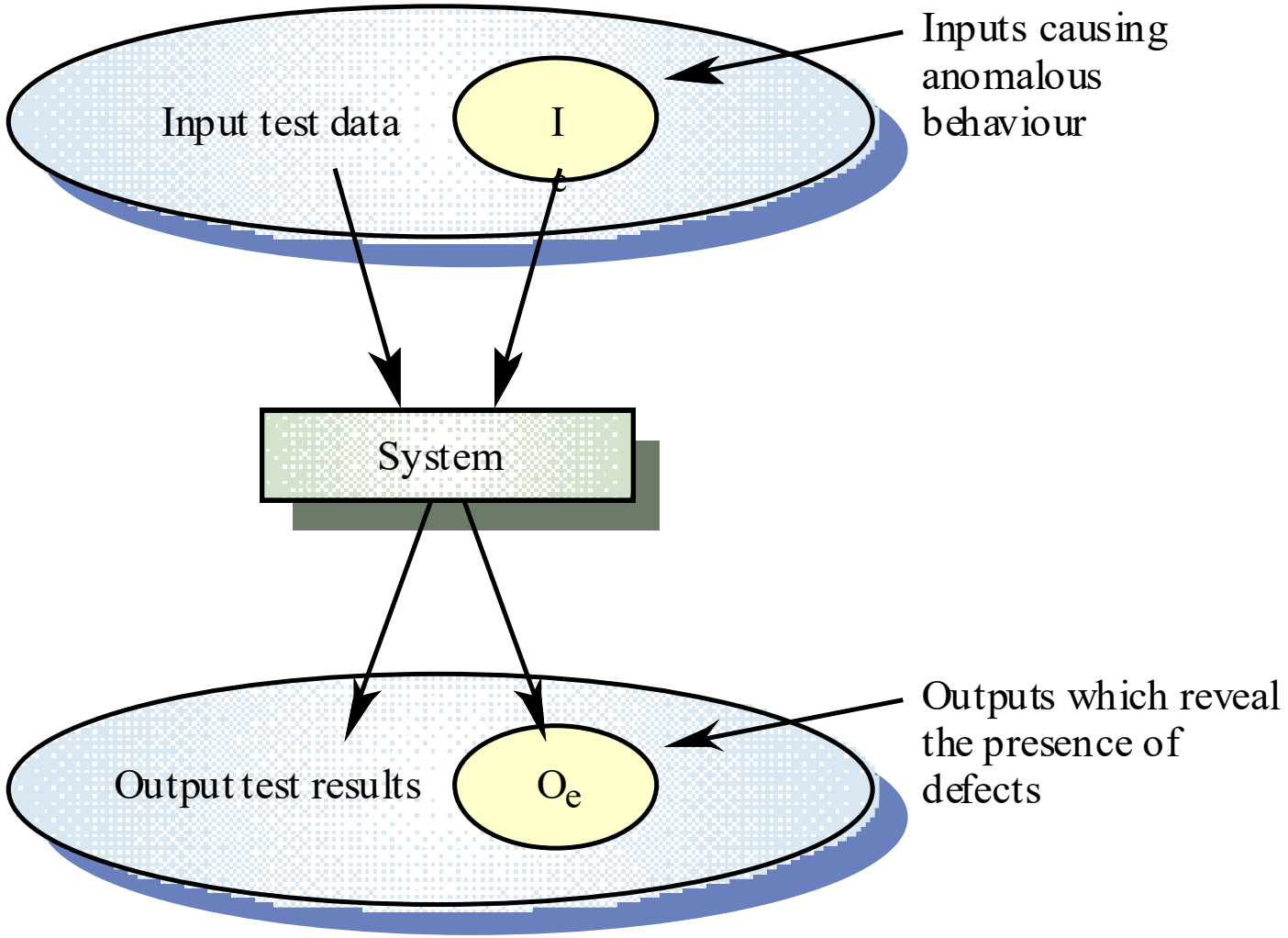
缺陷测试过程


Test cases
Test data
Test results
Test reports
Design test cases
Prepare test data
Run program with test data
Compare results to test cases

黑盒测试

- 把程序看成是一个“黑盒子”的测试方法
- 程序的测试用例是基于系统描述
- 测试规划可以在软件过程的早期开始

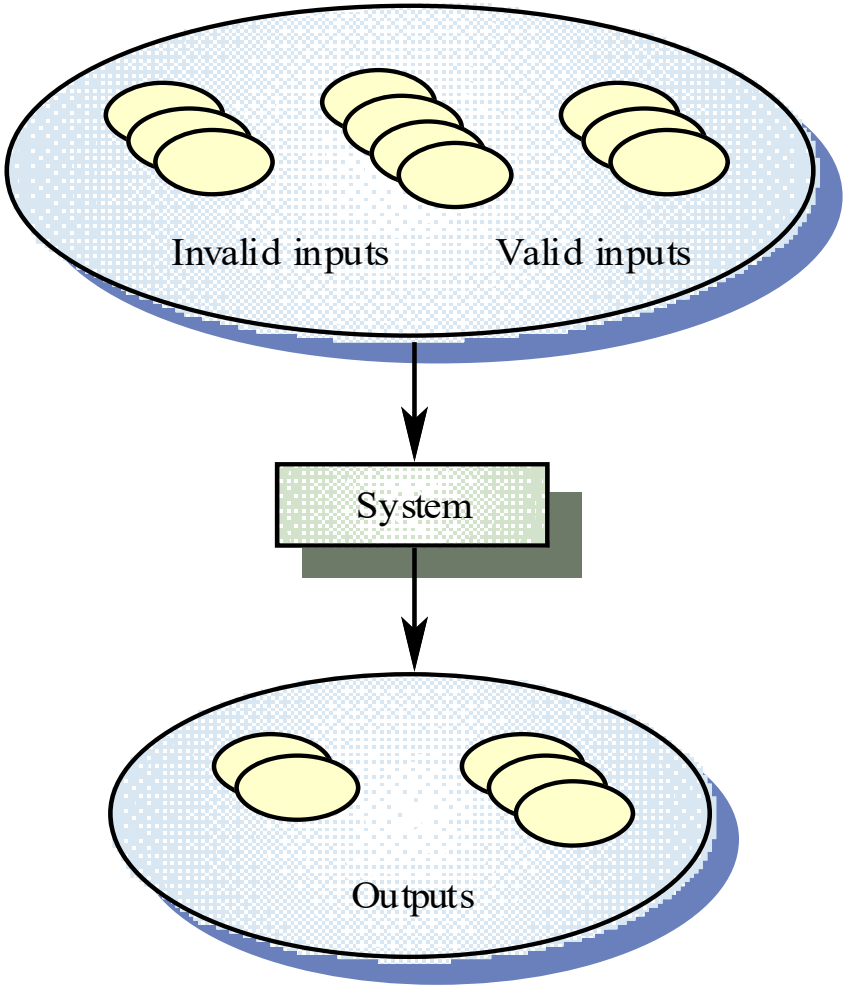
黑盒测试


Inputs causing anomalous behaviour
Input test data
I
System
Outputs which reveal the presence of defects
Output test results
O_e

等价划分

- 输入数据和输出结果通常可以分成几个不同的集合
- 每个集合都是一个等价划分，对集合中的每个成员程序的行为都是等价的
- 测试用例应该从每个等价划分中选取

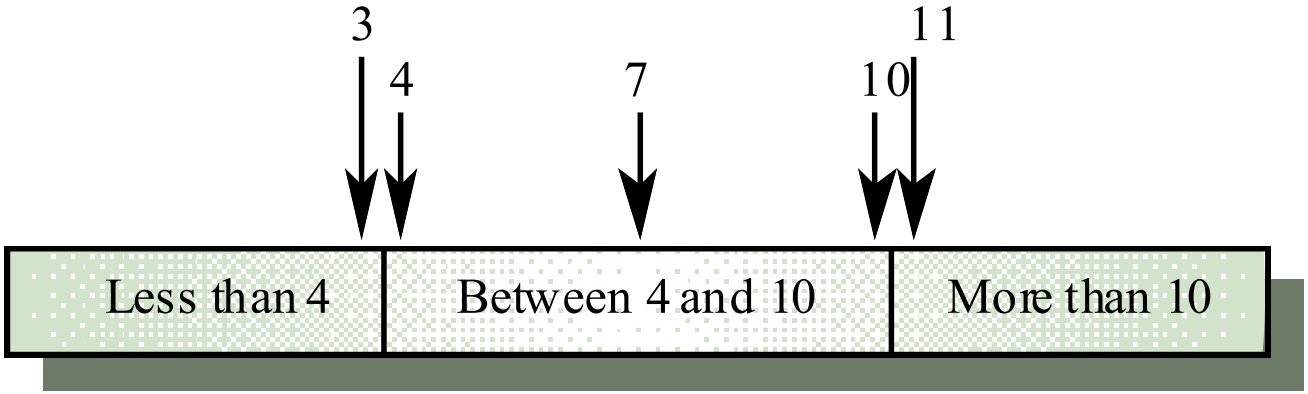
等价划分



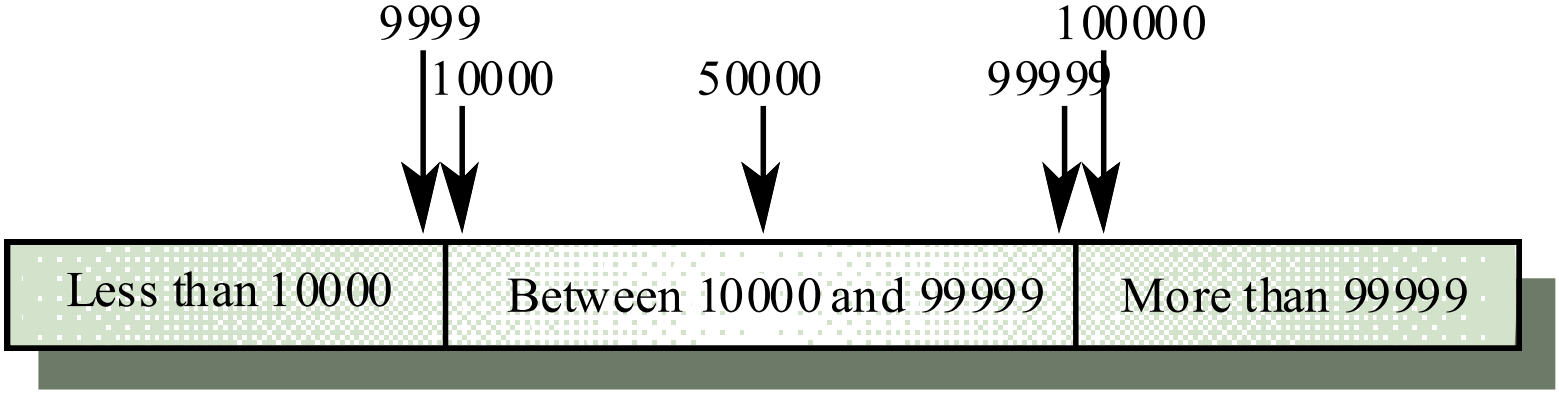
等价划分

- 将系统输入和输出划分成几个“等价集合”
  - 如果输入是 10,000～99,999的5位数整数,
    那么等价划分就是 <10,000, 10,000-99,999,>99,999
- 在这些集合的边界处选择测试用例
  - 00000, 09999, 10000, 99999, 100000,100001

等价划分



Number of input values



Input values

搜索例程描述

procedure Search (Key : ELEM ; T: ELEM_ARRAY;
Found : **in out** BOOLEAN; L: **in out** ELEM_INDEX) ;

Pre-condition

-- the array has at least one element
T'FIRST <= T'LAST

Post-condition

-- the element is found and is referenced by L
(Found and T (L) = Key)

or

-- the element is not in the array
(**not** Found **and**
not (**exists** i, T'FIRST <= i <= T'LAST, T (i) = Key))

搜索例程 – 输入划分

- 符合前置条件的输入
- 不符合前置条件的输入
- Key元素在数组中的输入
- Key元素不在数组中的输入

序列的测试划分

- 用一个单一数值的序列来测试软件
- 在不同的测试中使用不同规模的多个序列
- 让第一个、中间一个和最后一个得到测试
- 用一个长度为0的序列测试

搜索例程 – 输入划分

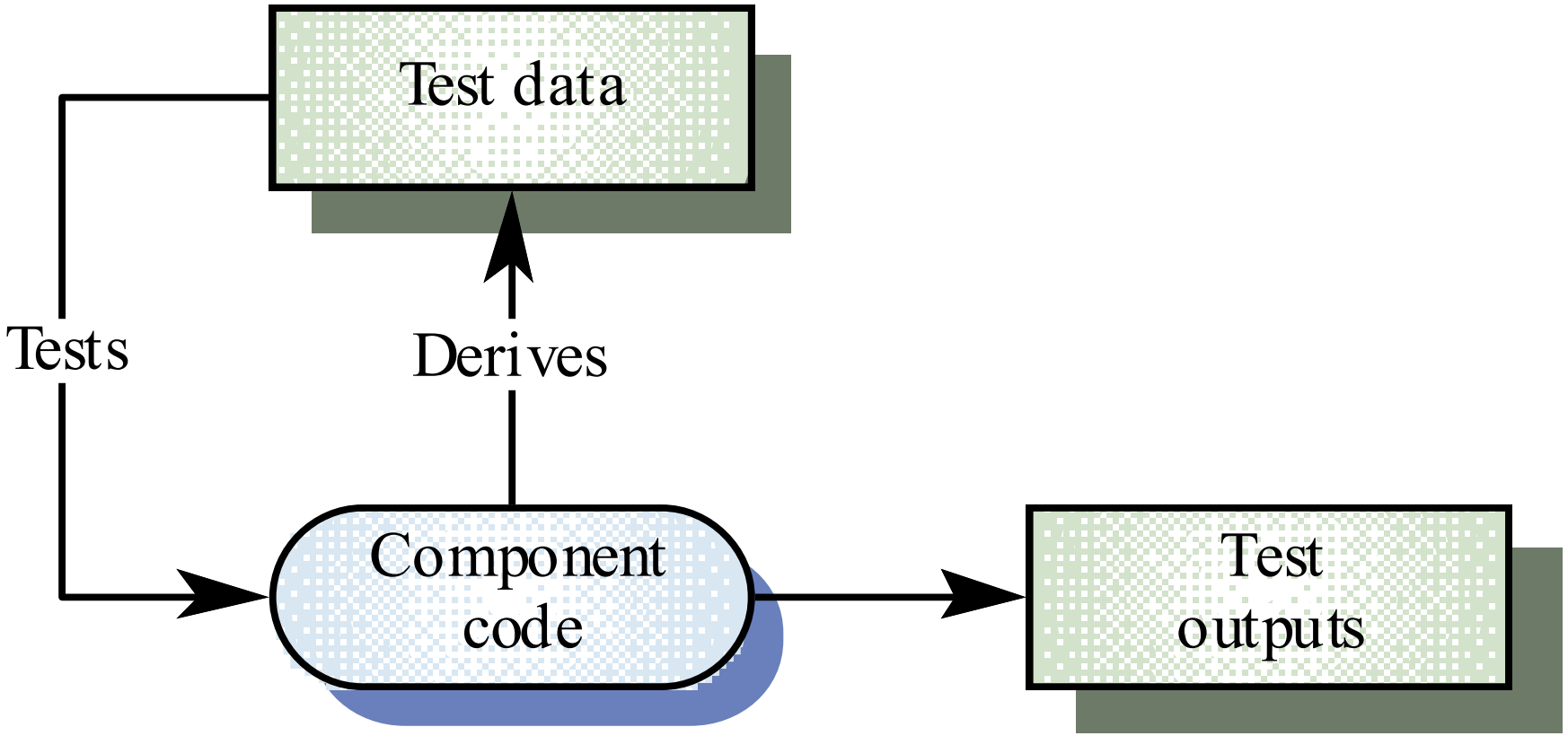
Array	**Element**
Single value	In sequence
Single value	Not in sequence
More than 1 value	First element in sequence
More than 1 value	Last element in sequence
More than 1 value	Middle element in sequence
More than 1 value	Not in sequence

Input sequence (T)	**Key** (Key)	**Output** (Found, L)
17	17	true, 1
17	0	false, ??
17, 29, 21, 23	17	true, 1
41, 18, 9, 31, 30, 16, 45	45	true, 7
17, 18, 21, 23, 29, 41, 38	23	true, 4
21, 23, 29, 33, 38	25	false, ??

结构化测试

- 也叫做“白盒测试”
- 测试用例来自程序结构。程序的知识用来识别附加的测试用例。
- 目标是让所有的程序语句执行一遍
(不是所有路径的组合)

白盒测试



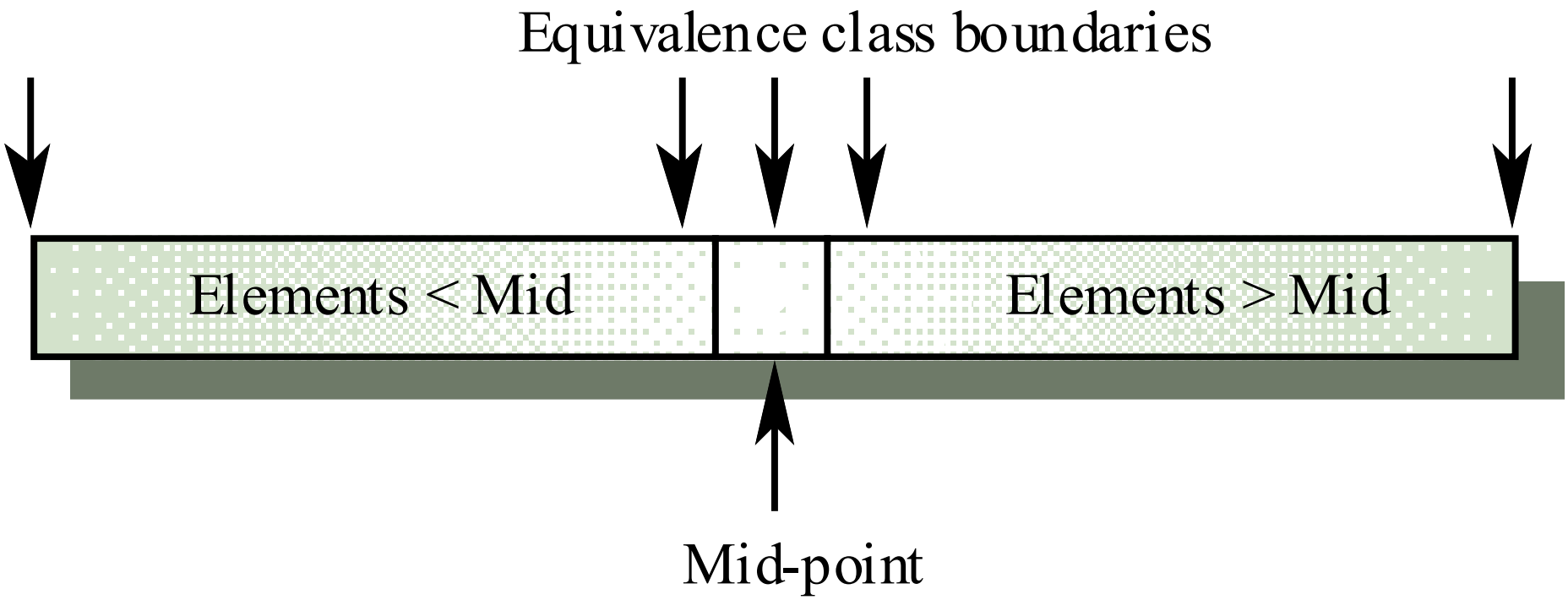
```
class BinSearch {

// This is an encapsulation of a binary search function that takes an array of
// ordered objects and a key and returns an object with 2 attributes namely
// index - the value of the array index
// found - a boolean indicating whether or not the key is in the array
// An object is returned because it is not possible in Java to pass basic types by
// reference to a function and so return two values
// the key is -1 if the element is not found

        public static void search ( int key, int [] elemArray, Result r )
        {
                int bottom = 0 ;
                int top = elemArray.length - 1 ;
                int mid ;
                r.found = false ; r.index = -1 ;
                while ( bottom <= top )
                {
                        mid = (top + bottom) / 2 ;
                        if (elemArray [mid] == key)
                        {
                                r.index = mid ;
                                r.found = true ;
                                return ;
                        } // if part
                        else
                        {
                                if (elemArray [mid] < key)
                                        bottom = mid + 1 ;
                                else
                                        top = mid - 1 ;
                        }
                } //while loop
        } // search
} //BinSearch
```

二叉搜索 (Java)

二叉搜索 – 等价划分

- 前置条件满足，被搜索元素在数组中
- 前置条件满足，被搜索元素不在数组中
- 前置条件不满足，被搜索元素在数组中
- 前置条件不满足，被搜索元素不在数组中
- 输入数组只有单个数值
- 输入数组有偶数个数值
- 输入数组有奇数个数值

二叉搜索的等价划分


Equivalence class boundaries
Elements < Mid
Elements > Mid
Mid-point

二叉搜索 – 测试用例

Input array (T)	**Key** (Key)	**Output** (Found, L)
17	17	true, 1
17	0	false, ??
17, 21, 23, 29	17	true, 1
9, 16, 18, 30, 31, 41, 45	45	true, 7
17, 18, 21, 23, 29, 38, 41	23	true, 4
17, 18, 21, 23, 29, 33, 38	21	true, 3
12, 18, 21, 23, 32	23	true, 4
21, 23, 29, 33, 38	25	false, ??

路径测试

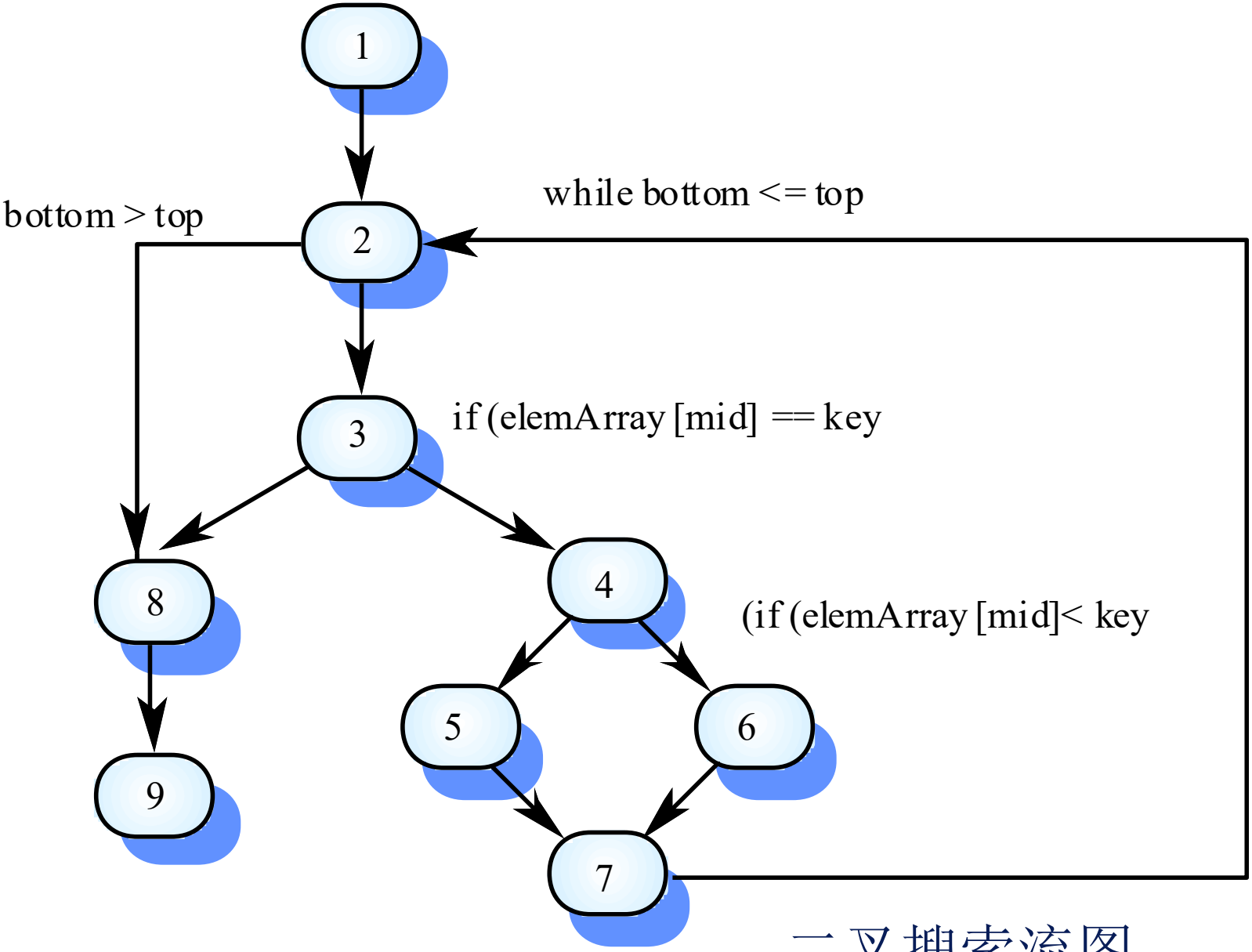
- 路径测试的目标是保证程序的每条路径都至少执行一次
- 路径测试首先需要一个程序流图，节点表示程序无分支的序列，边代表控制流
- 条件语句是程序流图中的节点

程序流图

- 描述程序的控制流。每个分支表示为独立的路径，循环用一个返回到条件节点的箭头表示
- 是计算环路复杂性的基础
- 环路复杂性 = 边数 – 节点数 +2

环路复杂性

- 测试所有语句的最少测试用例数等于环路复杂性
- 环路复杂性等于程序中的条件数
- 应谨慎使用。并不意味着测试是足够的。
- 虽然所有的路径都执行到，但不是所有的路径的组合都执行到


1
2
3
4
5
6
7
8
9
bottom > top
while bottom <= top
if (elemArray [mid] == key
(if (elemArray [mid]< key

二叉搜索流图

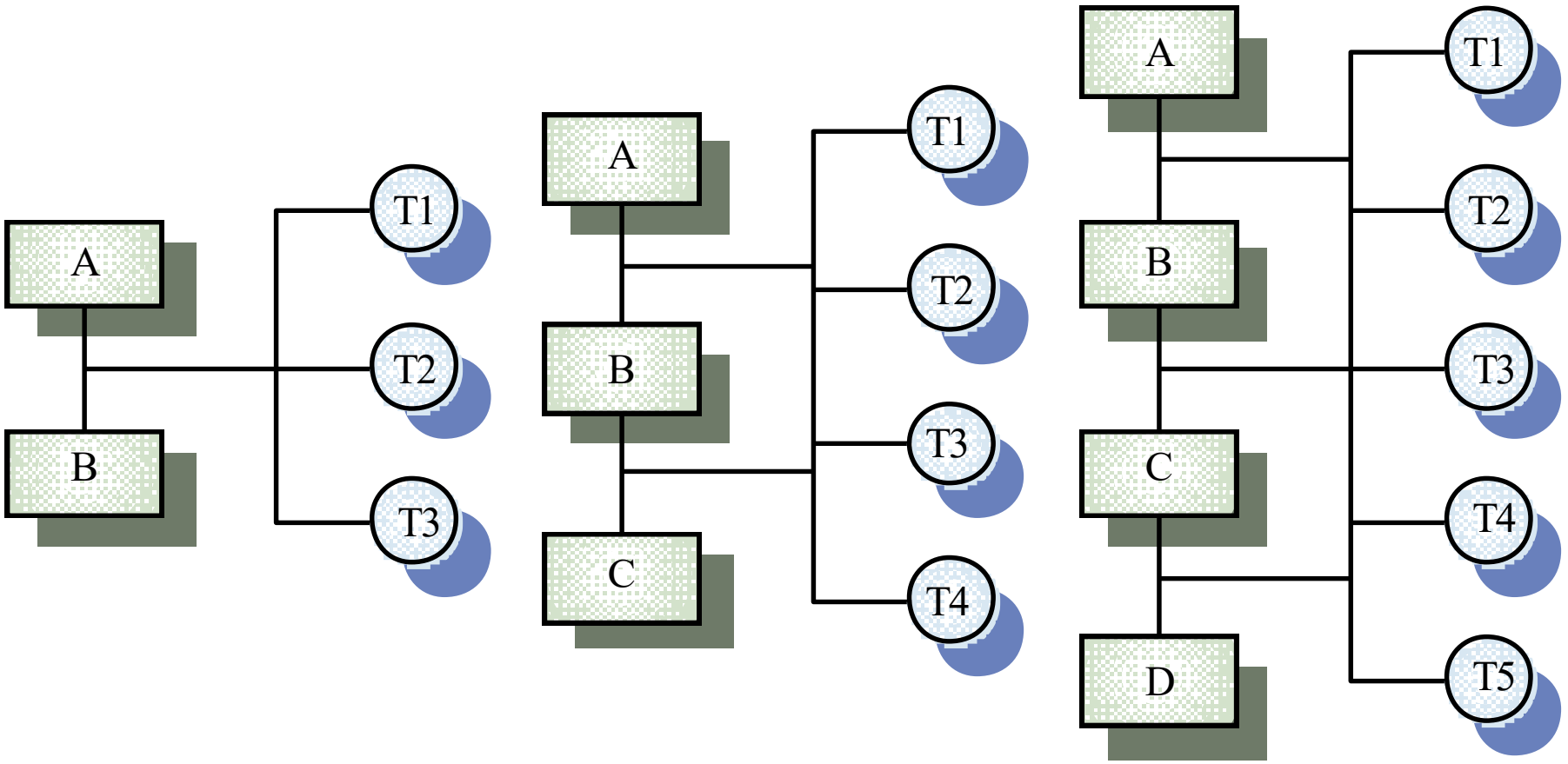
独立的路径

- 1, 2, 3, 8, 9
- 1, 2, 3, 4, 6, 7, 2
- 1, 2, 3, 4, 5, 7, 2
- 1, 2, 3, 4, 6, 7, 2, 8, 9
- 测试用例必须使这些路径都能执行到
- 一个动态的程序分析器可以用来检查路径是否被执行到

集成测试

- 测试组件集成完成的系统或子系统
- 集成测试应该是黑盒测试，测试用例是基于系统描述
- 主要困难是定位错误
- 增量式集成测试可以减轻这个问题

增量式集成测试



Test sequence 1

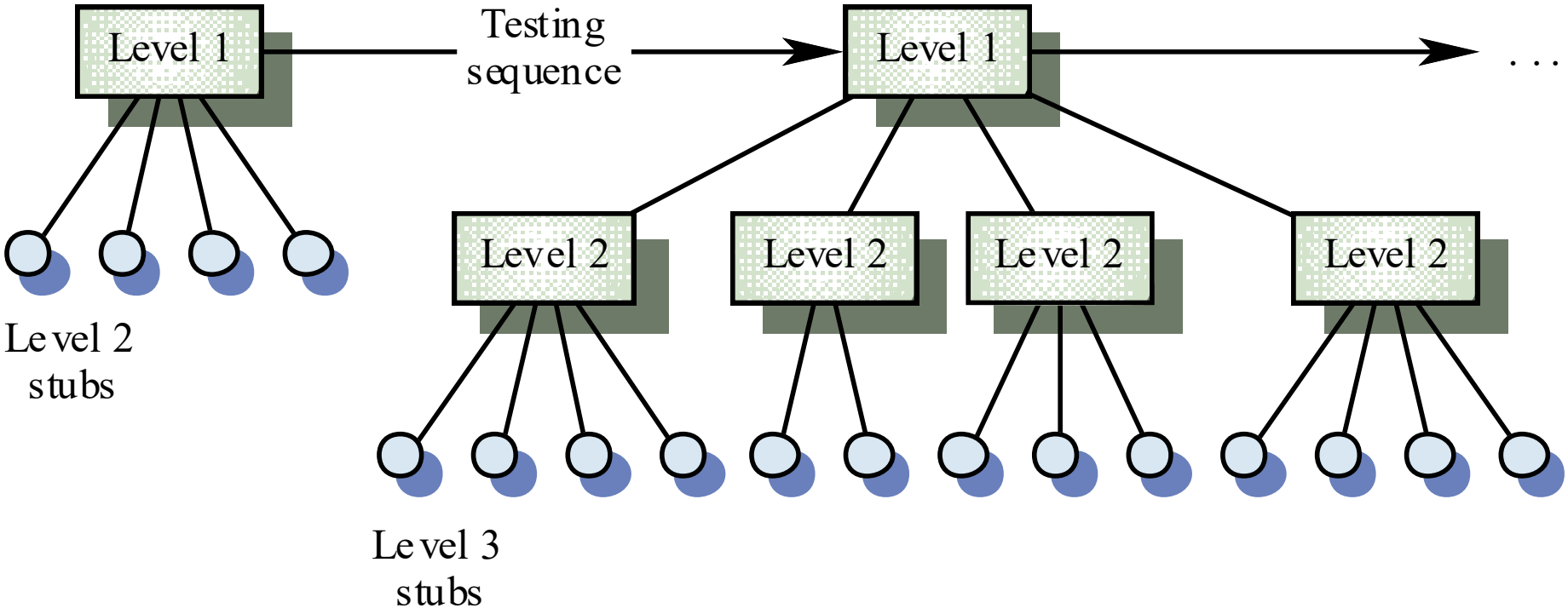
Test sequence 2

Test sequence 3

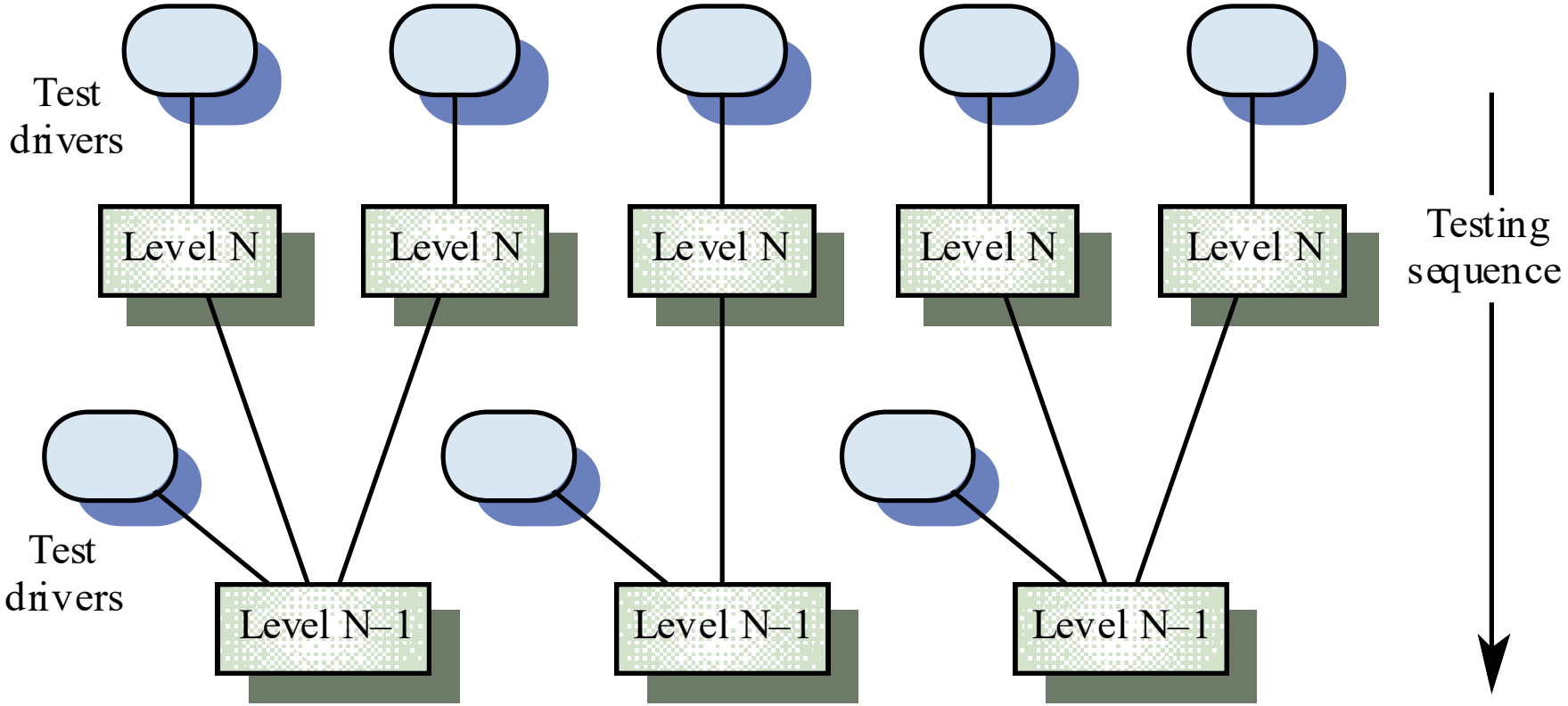
集成测试的方法

- 自顶向下测试
  - 从高层系统开始，从上至下集成组件，需要程序“桩”
- 自底向上测试
  - 一层层集成组件，直到产生整个系统
- 实践中，大多数系统集成是这两种方式的结合

自顶向下测试


Level 1
Testing
sequence
Level 1
. . .
Level 2
stubs
Level 2
Level 2
Level 2
Level 2
Level 3
stubs

自底向上测试


Test drivers
Level N
Level N
Level N
Level N
Level N
Testing sequence
Test drivers
Level N–1
Level N–1
Level N–1

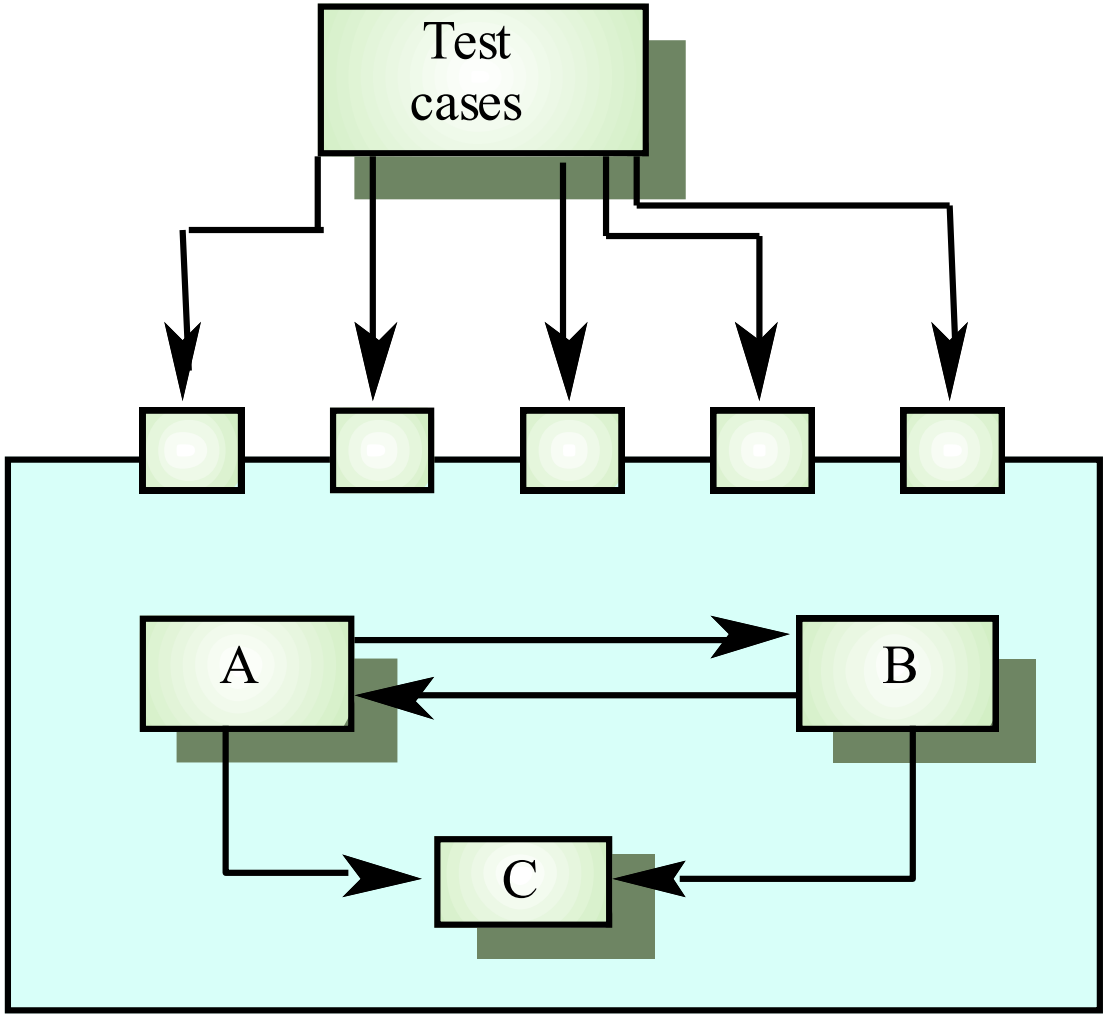
测试方法比较

- 体系结构的有效性
  - 自顶向下的测试容易发现体系结构中的错误
- 系统演示
  - 自顶向下集成测试允许在开发早期阶段有一个有限功能的演示系统
- 测试的执行
  - 自底向上的集成测试通常较容易
- 测试的观察
  - 两者都有问题。需要额外的代码来观察测试结果。

接口测试

- 当模块或子系统集成生成更大的系统时，就要做接口测试
- 目标是检测那些由于接口错误或无效的接口假设造成的系统缺陷
- 对面向对象开发特别重要，因为对象是由接口定义的。

接口测试



接口类型

- 参数接口
  - 数据从一个过程传递到另一个
- 共享内存接口
  - 过程之间共享的内存块
- 过程接口
  - 子系统封装了一组过程，可以被其它子系统调用
- 消息传递接口
  - 子系统从其它子系统请求服务

接口错误

- 接口误用
  - 组件调用另外组件时接口使用出错，例如：参数顺序错误
- 接口误解
  - 对被调用组件的行为进行了错误的假设，没有预期的行为
- 时机错误
  - 调用者和被调用者的速度不同，访问过期的数据

接口测试的一般准则

- 检查代码并明确列出对外部组件的调用。参数选择紧靠取值范围的边缘。
- 当有指针从接口传递时，一定要用空指针参数来测试接口
- 在过程接口系统中，设计一些容易引起组件失败的测试
- 在消息传递系统中使用强度测试
- 在共享内存系统中，设计一种测试，使组件激活的次序改变

强度测试

- 运行系统超过它的最大设计负荷。超负荷运行通常能暴露系统的缺陷
- 强度测试能测试系统的失败行为。系统失败不应是灾难性的，不应引起不可接受的服务或数据丢失
- 强度测试对分布式系统特别有用，网络超负荷的情况下会引起严重的系统性能下降。

面向对象测试

- 被测试的组件是实例化为对象的对象类
- 比单独的功能模块粒度要大，所以白盒测试方法需要扩展到更大的粒度
- 对自顶向下集成和测试来说没有明显的“顶层”

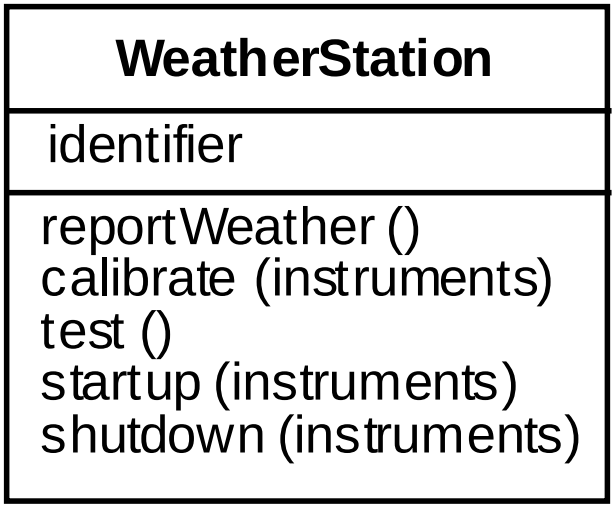
测试的层次

- 测试与对象关联的单个操作
- 测试单个对象类
- 测试对象集群
- 测试面向对象系统

对象类测试

- 完全的覆盖测试应该包括：
  - 对象中所有操作被单独隔离测试
  - 对象所有属性的设置和访问的测试
  - 对象的所有可能状态的测试
- 对象继承使得测试更加困难，因为要测试的信息不在一个地方

气象站对象接口



- 所有的操作需要测试用例
- 用一个状态模型识别用来测试的状态迁移
- 测试序列的例子
  - Startup → Waiting → Shutdown
  - Waiting → Calibrating → Testing → Transmitting → Waiting
  - Waiting → Collecting → Waiting → Summarising → Transmitting → Waiting

对象集成

- 在面向对象系统中集成的层次不明显
- 集群测试是一组协同操作的对象的集成和测试
- 要根据对象操作和系统的特征来识别对象集群

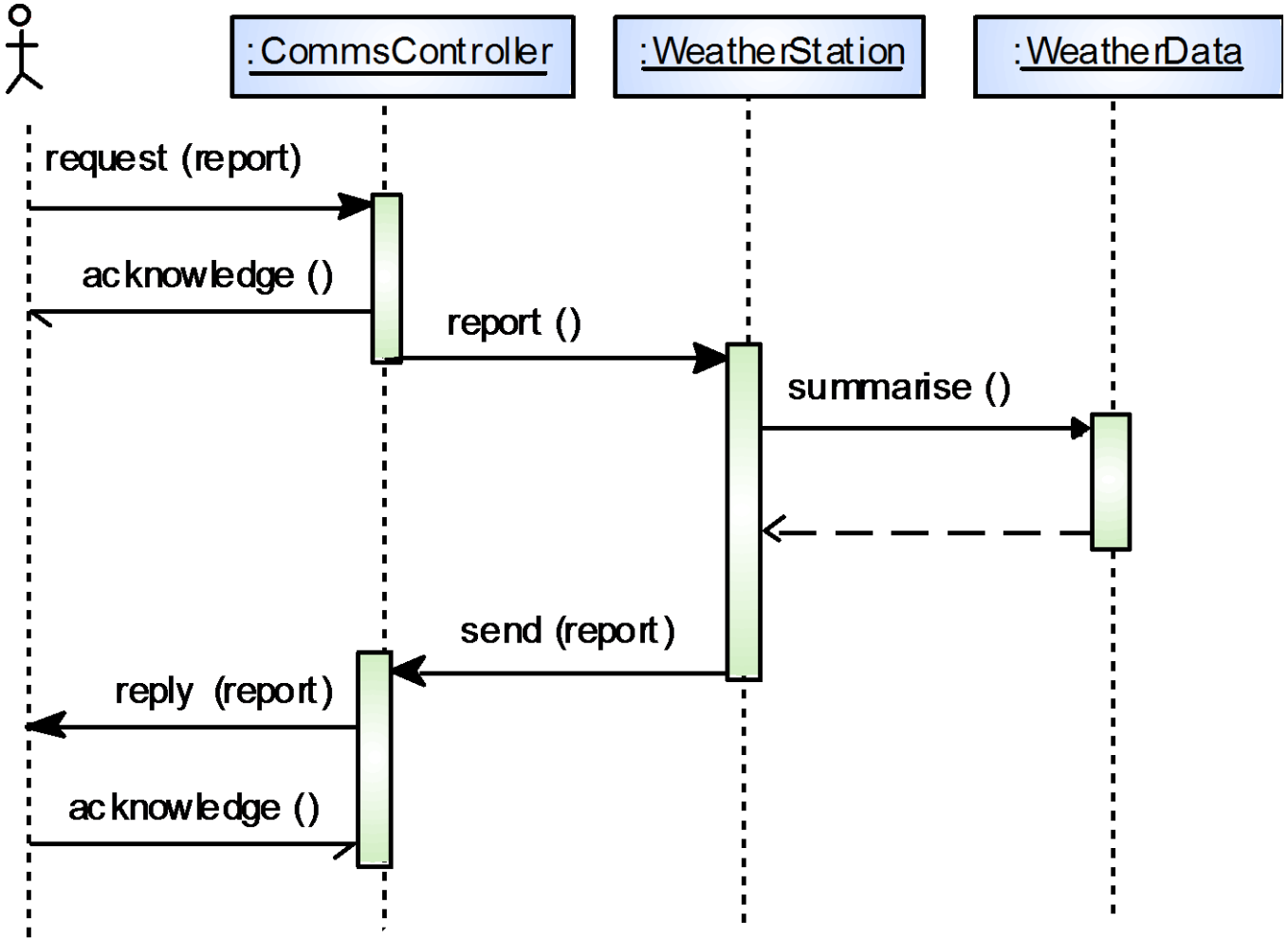
集群测试的方法

- 用例或基于场景的测试
  - 测试是基于用户和系统的交互
  - 优点是按照用户的体验来测试系统的特征
- 线程测试
  - 测试系统对于事件的响应
- 对象交互测试
  - 测试对象交互的序列，这个序列是由对象服务调用串起来的

基于场景的测试

- 从用例中识别场景，同时用交互图来补充说明。交互图中列出了场景所涉及的对象。
- 举例：气象站收集数据生成报告的场景

收集气象数据


:CommsController
:WeatherStation
:WeatherData
request (report)
acknowledge ()
report ()
summarise ()
send (report)
reply (report)
acknowledge ()

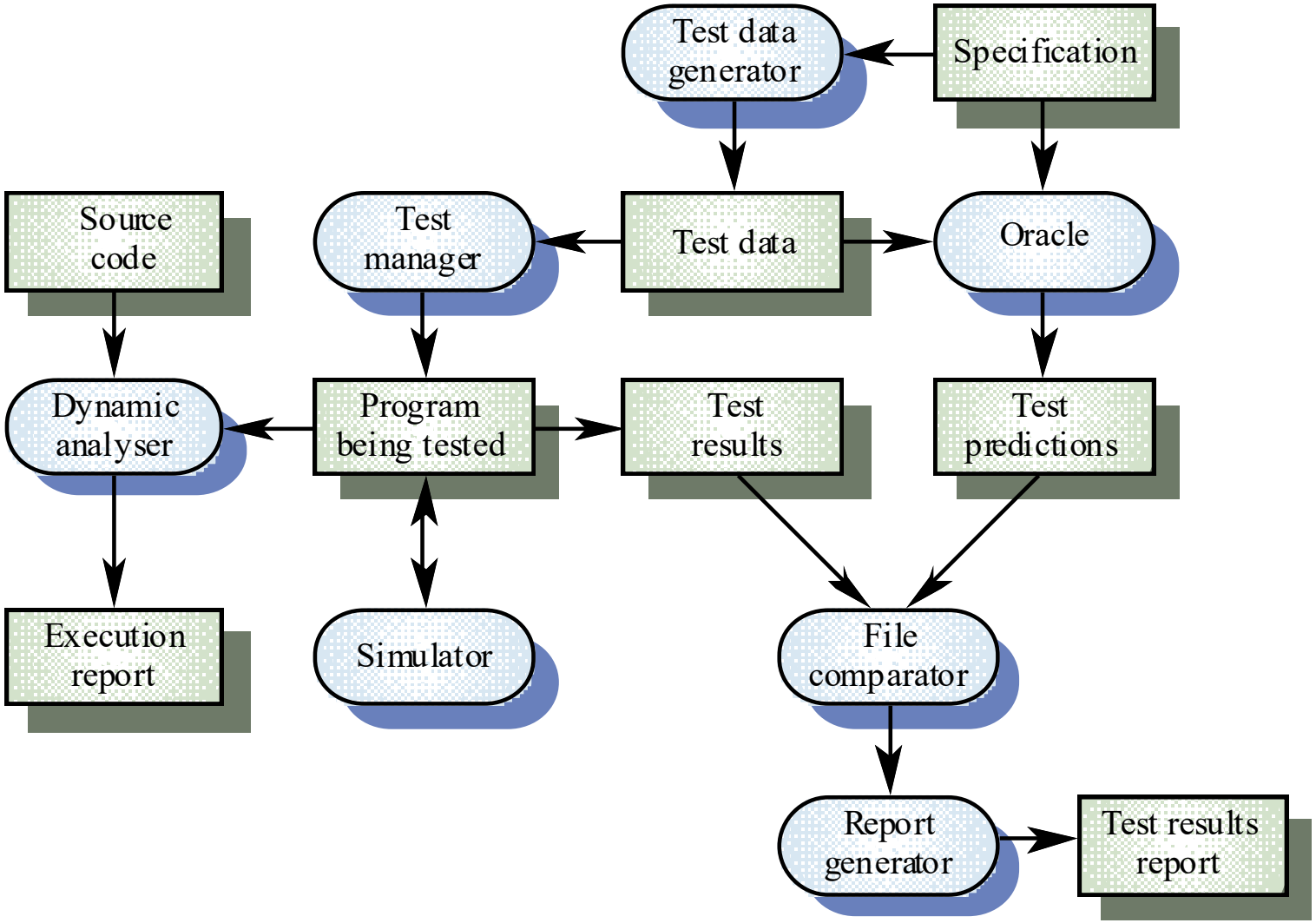
气象站测试

- 执行方法的线程
  - CommsController:request → WeatherStation:report → WeatherData:summarise
- 输入和输出
  - 输入一个报告请求，得到一个确认回复，最后返回报表
  - 可以通过产生原始数据来测试，确保数据被合适地汇总
  - 利用同样的原始数据测试WeatherData对象

测试工作平台

- 测试是一个费用昂贵的过程阶段。测试工作平台提供了一系列工具，可以减少需要的时间以及总的测试费用
- 大多数测试工作平台是开放系统，因为测试需求是随机构不同
- 难于跟封闭的设计和分析工作平台集成在一起

测试工作平台


Test data generator
Specification
Source code
Test manager
Test data
Oracle
Dynamic analyser
Program being tested
Test results
Test predictions
Execution report
Simulator
File comparator
Report generator
Test results report

测试平台的适应

- 需要为用户界面模拟器编写脚本，为测试数据生成器定义模式
- 测试输出必须手工准备，用来比较
- 需要开发专用的文件比较器

要点

- 对系统常用部分的测试要比对系统不常用的部分测试更重要。
- 等价划分是系统等价行为划分的测试用力集合
- 黑盒测试是基于系统描述
- 结构化测试识别能使所有程序路径都得到执行的测试用例

要点

- 测试覆盖度测量确保所有语句都至少执行一次
- 接口缺陷的发生是由于描述误读、误解、错误或无效的时间上的假设
- 测试对象类，要测试所有的操作、属性和状态
- 集成面向对象的系统围绕对象集群