软件质量保证与软件测试

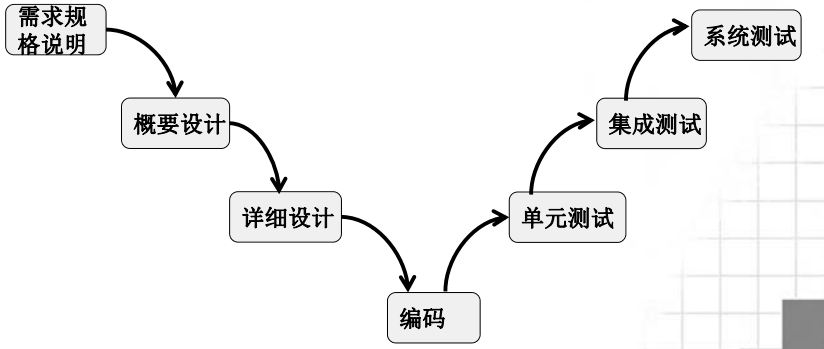
--集成测试

主讲教师:

Email:

软件测试的层次

- 单元测试——理解的最好
- 集成测试——理解的最差
- 系统测试——理解的较好



单元测试、集成测试、系统测试的差别

测试类型	对象	目的	测试依据	测试方法
单元测试	模块内部的程序	消除局部模块的逻辑和功能上的错误/缺陷	模块逻辑设计,模块外部说明	白盒测试 黑盒测试
集成测试	模块间的集成和调用关系	找出和软件设计相关的程序结构,模块调用关系,模块间接口方面的问题	程序结构、概要设计文档	结合使用白盒测试和黑盒测试.较多采用黑盒测试构造用例
系统测试	整个系统,包括硬件、人员等	对整个系统进行一系列的整体、有效性测试	系统结构设计，目标说明书，需求说明书	黑盒测试

引子

- 1999年9月，火星气象轨道人造卫星的使命，在经过41周4.16亿英里的成功飞行之后， 这颗卫星在就要开始进入火星轨道时消失了
- 卫星的缺陷本来可以通过集成测试查出：洛克希德·马丁太空科学家使用的是英制(磅)加速度数据，而喷气推进实验室采用公制(牛顿)加速度数据进行计算

主要内容

- 集成测试概述
- 集成测试的主要方法
  - 基于分解的集成
  - 基于调用图的集成
  - 基于路径的集成
- 案例研究

集成测试概述

- 定义：
  1. 集成经过单元测试的各组件评估它们之间交互的测试过程
  2. 根据实际情况对程序模块采用适当的的集成测试策略组装起来，对系统的接口以及集成后的功能进行正确性检验的测试工作
  3. 集成测试是构造软件体系结构的系统化技术，同时也是进行一些旨在发现与接口相关的错误的测试；其目标是利用已通过单元测试的构件建立设计中描述的程序结构

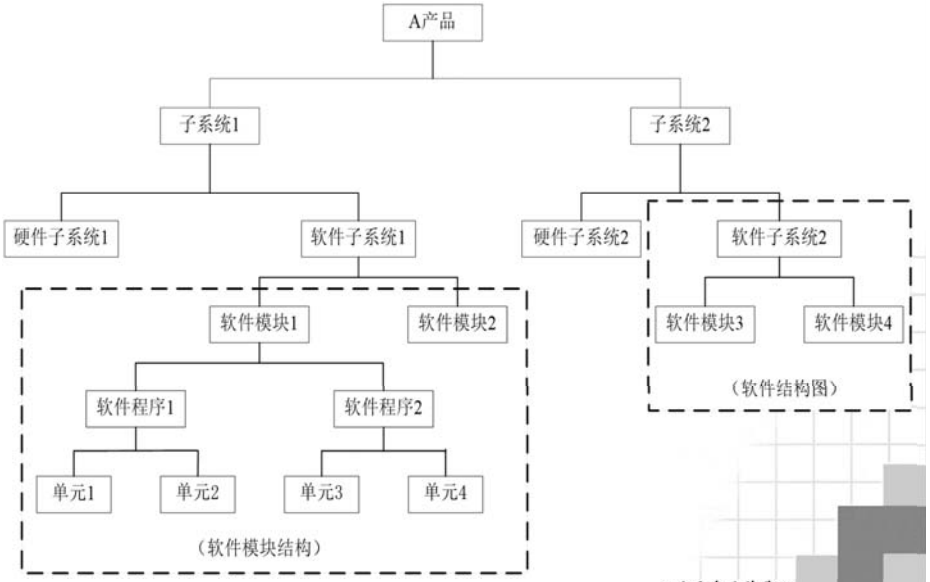
集成测试概述

- 目的：
  - 在把各个模块连接起来的时候，穿越模块接口的数据是否会丢失
  - 一个模块的功能是否会对另一个模块的功能产生不利的影响
  - 各个子功能组合起来，能否达到预期要求的父功能
  - 全局数据结构是否有问题
  - 单个模块的误差累积起来，是否会放大，以至达到不能接受的程度

集成测试概述

- 原则：
  - 所有公共接口都要被测试到
  - 关键模块必须进行充分的测试
  - 集成测试应当按一定的层次进行
  - 集成测试的策略选择应当综合考虑质量、成本和进度之间的关系
  - 集成测试应当尽早开始，并以总体设计为基础
  - 在模块与接口的划分上，测试人员应当和开发人员进行充分的沟通
  - 当接口发生修改时，涉及的相关接口必须进行再测试
  - 测试执行结果应当如实的记录
  - 集成测试应根据集成测试计划和方案进行，不能随意测试
  - 项目管理者应保证审核测试用例

示例



集成测试的层次

- 对于传统软件来说，按集成粒度不同，可以把集成测试分为3个层次，即：
  - 模块内集成测试
  - 子系统内集成测试（模块间集成）
  - 子系统间集成测试
- 对于面向对象应用系统来说，按集成粒度不同，可以把集成测试分为2个层次：
  - 类内集成测试
  - 类间集成测试

集成测试策略

- 驱动器（Driver）：用来模拟待测模块的上级模块。驱动器在集成测试中接受测试数据，将相关的数据传送给待测模块，启动待测模块，并打印出相应的结果
- 桩（Stub）：也称为存根程序，用以模拟待测模块工作过程中所调用的模块。桩由待测模块调用，它们一般只进行很少的数据处理，例如打印入口和返回，以便于检验待测模块与下级模块的接口
- 增量式集成
- 非增量式集成

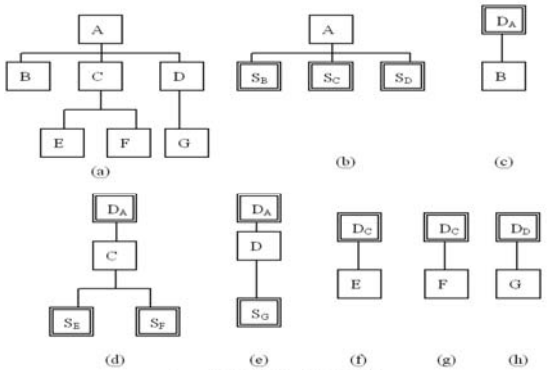
非增量式集成（大爆炸集成）

- 非渐增式集成方法首先对每个子模块进行测试（即单元测试），然后将所有模块全部集成起来一次性进行集成测试
- 把所有通过单元测试的模块一次性集成到一起进行测试，不考虑组件之间的互相依赖性及可能存在的风险
- 目的是尽可能缩短测试时间，使用最少的测试用例验证系统

非增量式集成（大爆炸集成）

- 示例：
  - A配备三个桩模块
  - B、E、F、G配备了驱动模块
  - C、D分别配备了桩模块和驱动模块

分别完成了7个模块的单元测试后，然后一下子集成这7个模块



非增量式集成测试示例

大爆炸集成

- 优点：
  - 可以并行调试所有模块
  - 需要的测试用例数目少
  - 测试方法简单、易行
- 缺点：
  - 不能充分对各个模块之间的接口进行充分测试
  - 不能很好的对全局数据结构进行测试
  - 如果一次集成的模块数量多，集成测试后可能会出现大量的错误。另外，修改了一处错误之后，很可能新增更多的新错误，新旧错误混杂，给程序的完善带来很大的麻烦
  - 即使集成测试通过，也会遗漏很多错误

增量式集成

- 程序以小增量的方式逐步进行构造和测试，这样错误易于分离和纠正，更易于对接口进行彻底测试，而且可以运用系统化的测试方法
- 传统的增量测试策略分为自顶向下集成、自底向上集成以及三明治集成

主要内容

- 集成测试概述
- 集成测试的主要方法
  - 基于分解的集成
  - 基于调用图的集成
  - 基于路径的集成
- 案例研究

基于分解的集成

- 在讨论集成测试时，测试方法都基于采用树或文字形式来表示的功能分解。这类讨论不可避免地要深入到将要集成的模块的顺序
  - 自顶向下集成（从树顶开始向下）
  - 自底向上集成（从树底开始向上）
  - 三明治集成（前两种方法的某种组合）

自顶向下集成

- **从顶层开始，采用同设计顺序一样的思路对被测系统进行测试，一般集中于顶层的组件，然后逐步测试处于底层的组件，被上层单元调用的下层单元以桩出现**
- **自顶而下的集成方式**
  - **深度优先**

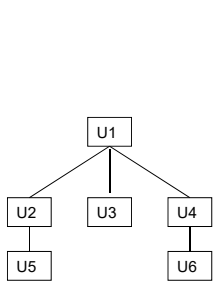
    从最顶层单元开始，持续向下到下一层，选择一个分支，自顶而下一个一个的集成这条分支上的所有单元，直到最底层，然后转向另一个分支，重复这样的集成操作直到所有的单元都集成进来
  - **广度优先**

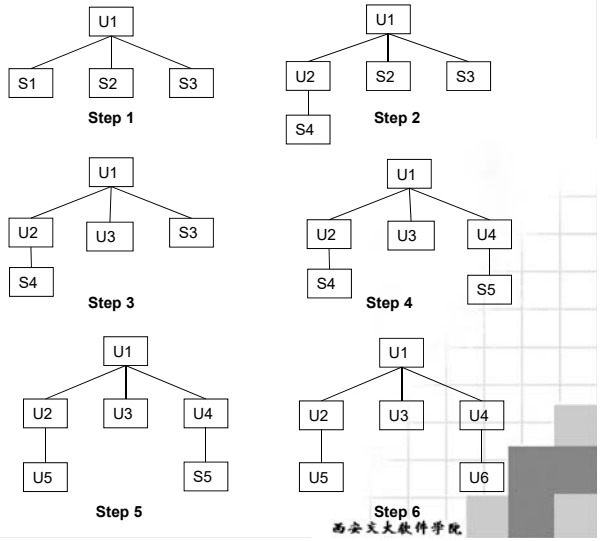
    从最顶层单元开始，持续向下到下一层， 一个个完成下一层上所有单元集成后，再转向下面一层，重复这样的集成操作直到所有的单元都集成进来
  - **其他方式**

自顶向下集成

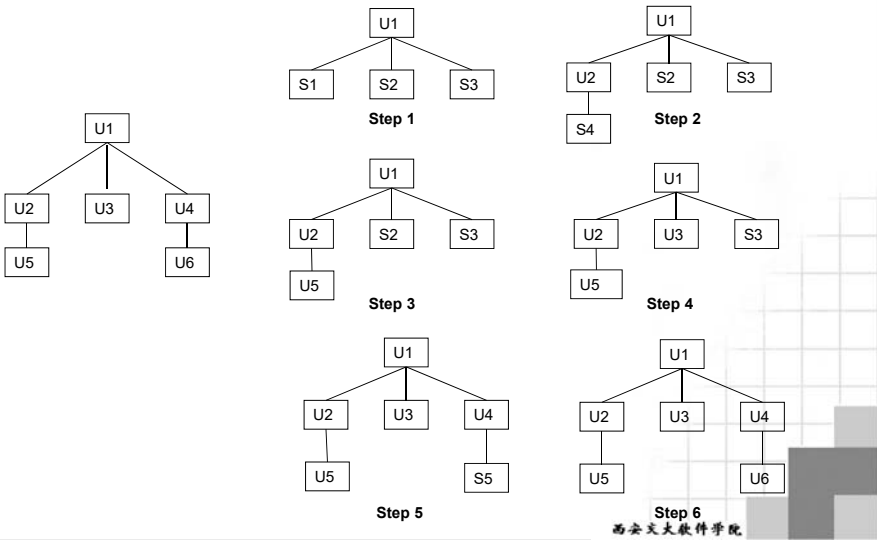
- 集成测试的步骤：
  - 1.主程序（主控模块）作为测试驱动器，把对主控模块进行单元测试时引入的被调用模拟子模块用实际模块替代
  - 2.依照所选用的模块集成方式（深度优先、广度优先、或其他方式），下层的被调用模拟子模块**一次一个**地被替换为真正的模块
  - 3.在每个模块被集成时，都必须立即进行一遍测试
  - 4.重复第2步，直到所有节点被集成进来

示例---广度优先集成



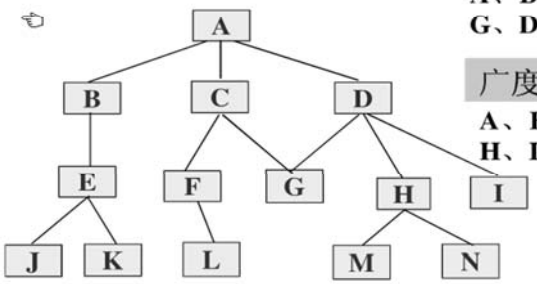


示例---深度优先集成



补充案例

- 自顶向下集成
  - 广度优先
  - 深度优先



深度优先：

A、B、E、J、K、C、F、L、G、D、H、M、N、I

广度优先：

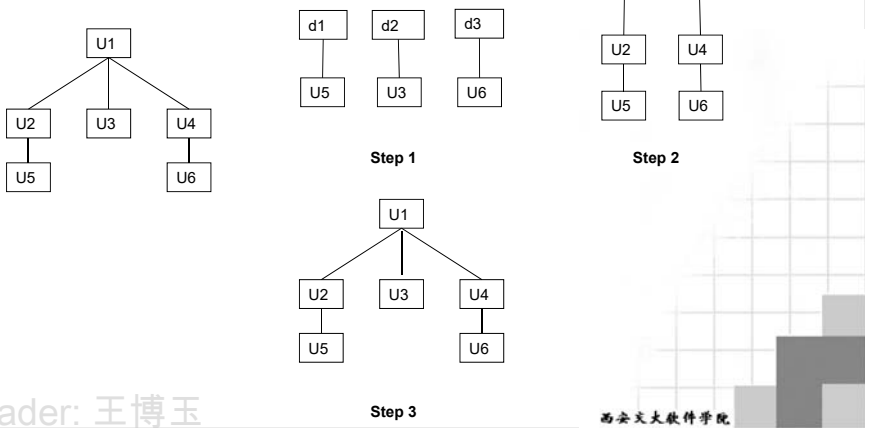
A、B、C、D、E、F、G、H、I、J、K、L、M、N

自底向上集成

- 自底向上集成是自顶向下顺序的“镜像”，不同的是，桩由模拟功能分解树上一层单元的驱动器模块代替
- 首先从分解树的叶开始，并用特别编写的驱动器测试
- 驱动器中的一次性代码比桩中的少

示例

- 从最底层U5,U3,U6开始，开发3个驱动模块d1,d2,d3调用它们
- 用U5集成U2，U6,U4被d4,d5代替
- 将所有单元集成在一起



两种集成方法的比较

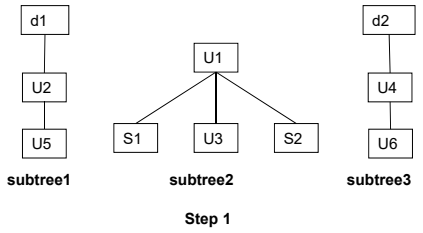
	◆优点	◆缺点
自顶向下测试	◆ 较早地验证了主要控制和判断点 ◆ 按深度优先可以首先实现和验证一个完整的软件功能 ◆ 功能较早证实，带来信心 ◆ 只需一个驱动，减少驱动器开发的费用 ◆ 支持故障隔离	◆桩的开发量大 ◆底层验证被推迟 ◆底层组件测试不充分
自底向上测试	◆对底层组件行为较早验证 ◆工作最初可以并行集成，比自顶向下效率高 ◆减少了桩的工作量 ◆支持故障隔离	◆驱动的开发工作量大 ◆对高层的验证被推迟 ◆设计上的错误不能被及时发现

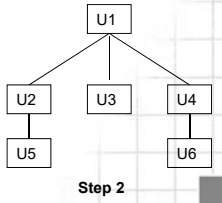
三明治集成

- 三明治集成测试是将自顶向下测试与自底向上测试两种模式有机结合起来，采用并行的自顶向下、自底向上集成方式，形成的方法。
- 三明治集成测试更重要的是采取持续集成的策略
- 优点：桩和驱动的开发工作都比较小
- 缺点：作为大爆炸集成的后果，在一定程度上增加了定位缺陷的难度

示例

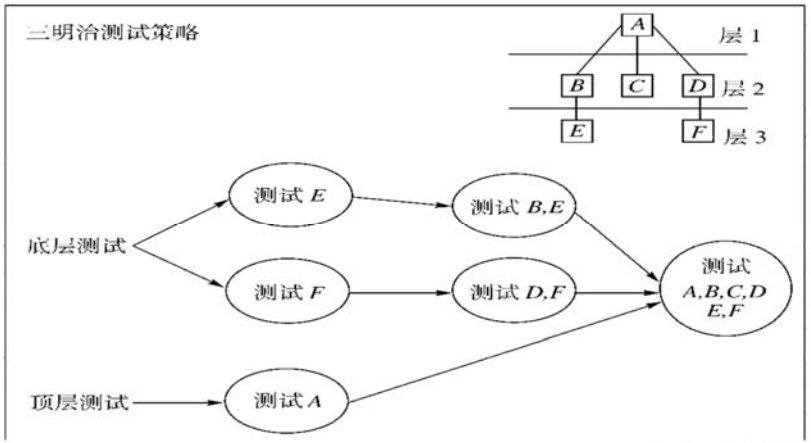
- 基于功能树，选择完全分支/子分支作为集成单元，这里选择了3个子树
  - 为了测试U2和U5的集成，开发一个驱动器d1
  - 开发两个桩S1和S2测试U1和U3的集成
  - 为了测试U4和U6，开发一个驱动器d2
- 将所有的测试子树集成在一起





示例

- 另一种三明治集成方式



基于分解的集成总结

- 优点：
  - 直觉上很清晰，容易根据分解树跟踪
  - 缺陷容易定位
- 缺点：
  - 要开发桩和驱动器
  - 重新测试需要较大工作量
- 给定分解树所需的集成测试过程个数：
  - 集成测试个数 = 节点 – 叶 + 边

练习 --- SATM系统

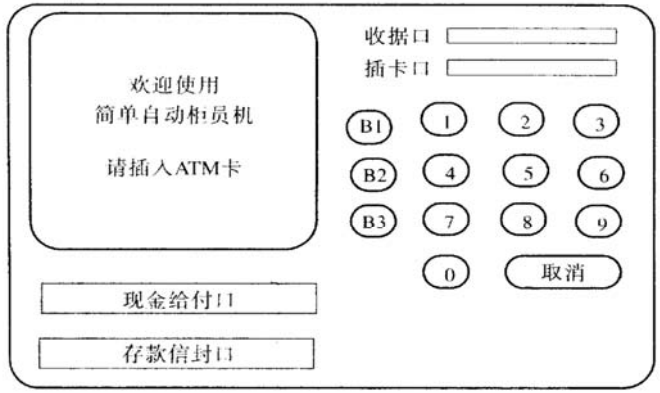
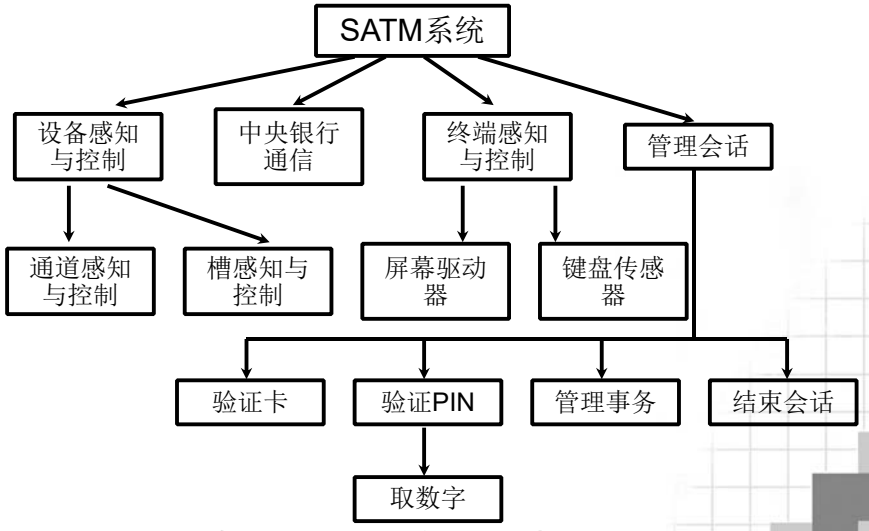


图2-3 SATM终端

SATM系统的功能分解树



SATM系统的扩展功能分解树

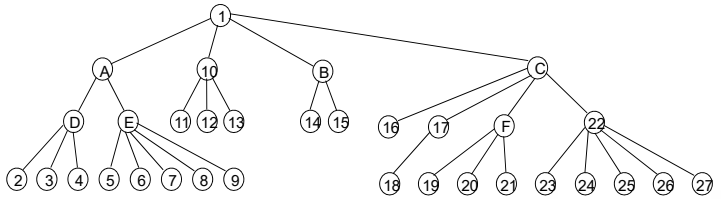
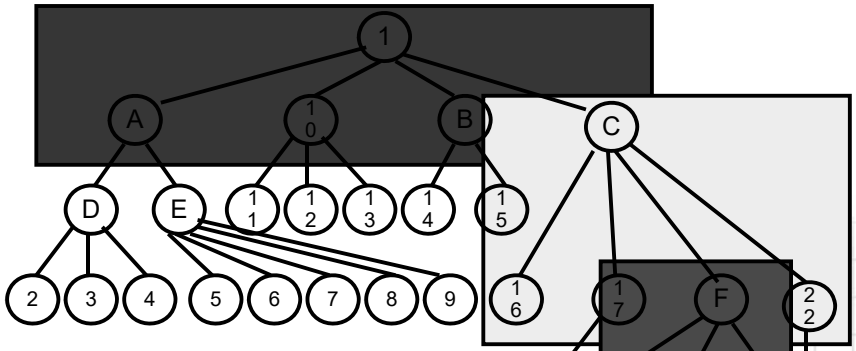


Table 1: SATM Units and Abbreviated Names

1	1	SATM系统
A	1. 1	设备感知与控制
D	1. 1. 1	通道感知与控制
2	1. 1. 1. 1	取通道状态
3	1. 1. 1. 2	控制通道
4	1. 1. 1. 3	给付现金
E	1. 1. 2	槽感知与控制
5	1. 1. 2. 1	检查ATM卡槽
6	1. 1. 2. 2	取存款槽状态
7	1. 1. 2. 3	控制ATM卡传送器
8	1. 1. 2. 4	控制信封传送器
9	1. 1. 2. 5	读ATM卡磁条
10	1. 2	中央银行通信
11	1. 2. 1	取PAN的PIN
12	1. 2. 2	取账户状态
13	1. 2. 3	发出每日事务处理
B	1. 3	终端功能与控制
14	1. 3. 1	屏幕驱动器
15	1. 3. 2	键盘感知器
C	1. 4	管理会话
16	1. 4. 1	检验ATM卡
17	1. 4. 2	检验PIN
18	1. 4. 2. 1	取PIN
F	1. 4. 3	关闭会话
19	1. 4. 3. 1	新事务处理请求
20	1. 4. 3. 2	打印收据
21	1. 4. 3. 3	发送本地事务处理
22	1. 4. 4	管理事务处理
23	1. 4. 4. 1	取事务处理类型
24	1. 4. 4. 2	取账户类型
25	1. 4. 4. 3	报告余额
26	1. 4. 4. 4	处理存款
27	1. 4. 4. 5	处理取款

SATM -- 自顶向下集成



- 先对节点1进行测试，将A、10、B、C以桩的形式集成进来；
- 如果测试通过，则依次将这四个桩替换为实际代码，这里假定先替换节点C
- 然后对节点C进行集成测试，
- 依次进行，直到所有节点全集成进来

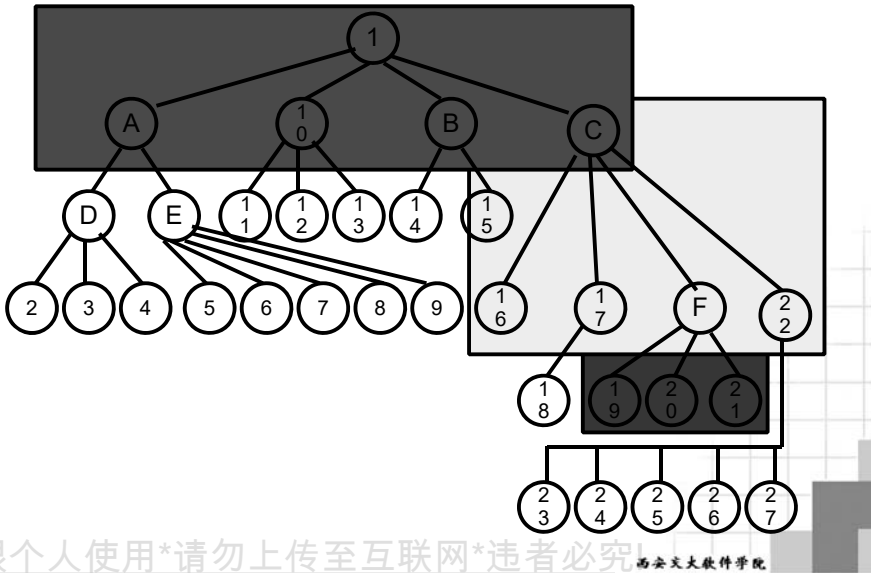
SATM-----桩示例

- 一般来说，测试人员不得不开发桩
- 桩示例（取PAN的PIN、键盘感知器）

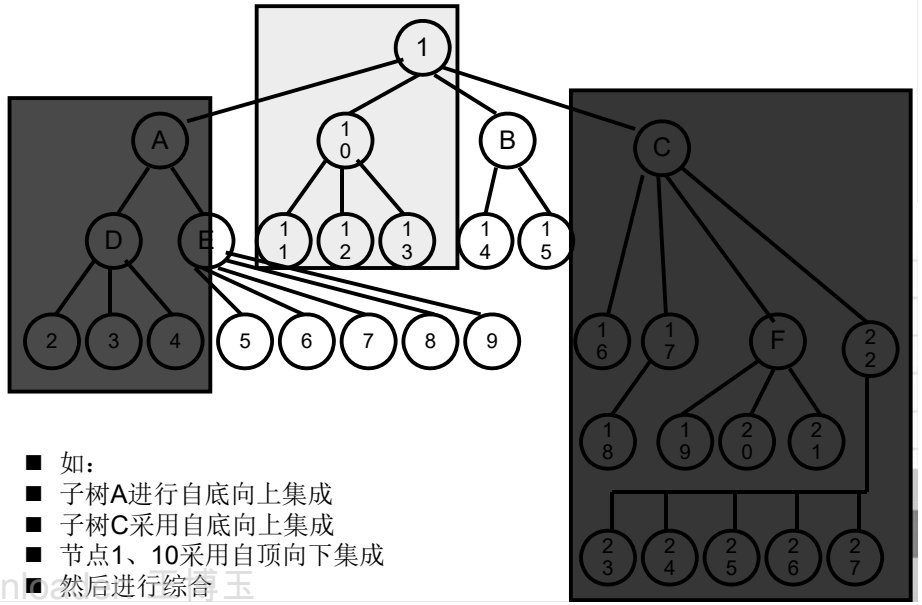
```
Procedure Get PINforPAN(PAN， Expected PIN) STUB
        If PAN = '1123' Then Expected PIN := '8876'
        If PAN = '1234' Then Expected PIN := '8765'
        If PAN = '8746' Then Expected PIN := '1253'
End

Procedure KeySensor (KeyHit) STUB
        data: KeyStrokes STACK OF '8 ‘, '8 ‘, '7 ‘, 'cancel'
        KeyHit = POP (KeyStrokes)
End
```


SATM----自底向上集成



SATM----三明治集成



- 如：
- 子树A进行自底向上集成
- 子树C采用自底向上集成
- 节点1、10采用自顶向下集成
- 然后进行综合

主要内容

- 集成测试概述
- 集成测试的主要方法
  - 基于分解的集成
  - 基于调用图的集成
  - 基于路径的集成
- 案例研究

基于调用图的集成

- 基于分解集成的缺点之一是以功能分解树为基础，不能反映软件的行为特征
- **基于调用图的集成方法有：**
- **1.成对集成**
- **2.相邻集成**

SATM系统的调用图

- 单元调用图是一种有向图，节点表示程序单元，边表示对应程序。即：如果单元A调用单元B，则从单元A到单元B有一条有向边
- 该图是SATM调用图，对于内度和外度很高的节点，集成测试很重要

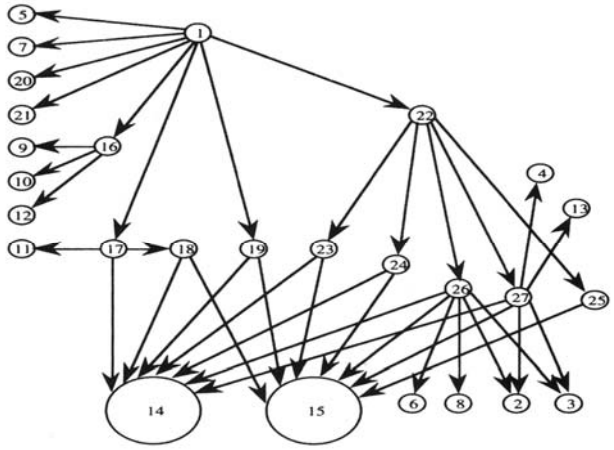


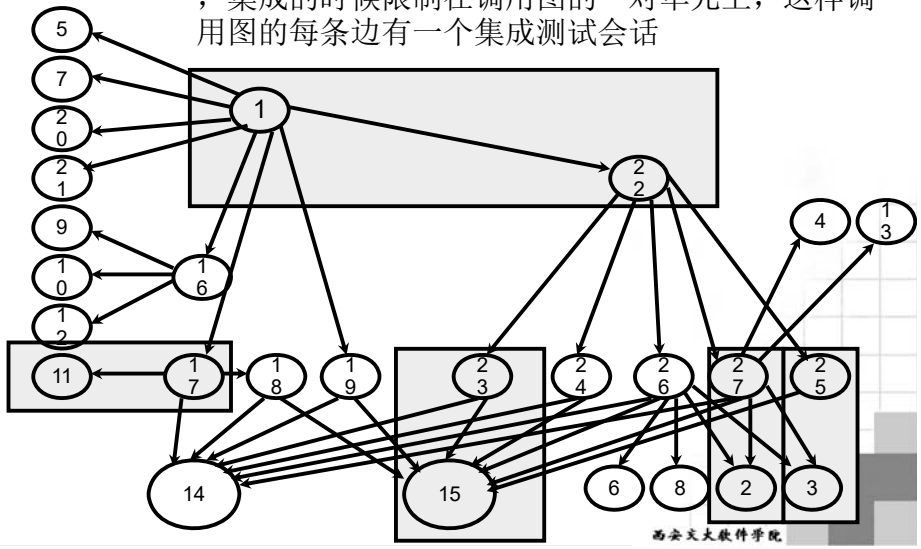
图13-2 SATM调用图

SATM调用图的邻接矩阵

	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1				X		X							X		X	X		X	X	X	X					
2																										
3																										
4																										
5																										
6																										
7																										
8																										
9																										
10																										
11																										
12																										
13																										
14																										
15																										
16								X	X		X															
17										X			X				X									
18													X	X												
19													X	X												
20																										
21																										
22													X	X								X	X	X	X	X
23													X	X												
24													X	X												
25														X												
26	X	X			X		X					X	X	X												
27	X	X	X		X		X					X	X	X												

成对集成

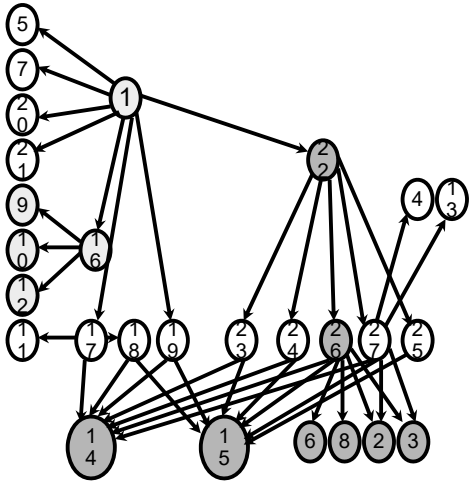
基本思想：免除（减少）桩/驱动模块开发的工作量，集成的时候限制在调用图的一对单元上，这样调用图的每条边有一个集成测试会话



相邻集成

基本思想：找出邻居进行集成

邻居包括所有直接前驱节点和所有直接后继节点



节点（邻居）	前驱	后继
16	1	9, 10, 12
17	1	11, 14, 18
18	17	14, 15
19	1	14, 15
23	22	14, 15
24	22	14, 15
26	22	14, 15, 6, 8, 2, 3
27	22	14, 15, 2, 3, 4, 13
25	22	15
22	1	23, 24, 26, 27, 25
1	–	5, 7, 20, 21, 16, 17, 19, 22

相邻集成邻居数量

- **内部节点=节点-（源节点+汇节点）**
- **邻居=内部节点+源节点**

合并得到：

邻居 = 节点 – 汇节点

- 相邻集成可大大降低集成测试会话数量(从40降至11)，并且避免了桩和驱动器的开发
- 邻居本质上是前面介绍过的三明治 (稍有不同，因为邻居的基本信息是调用图，不是分解树)
- 与三明治集成的共同之处:相邻集成测试具有“中爆炸”集成的缺陷隔离困难

基于调用图集成总结

- **优点：**
  - **偏离了纯结构基础，转向行为基础，因此底层假设是一种改进**
  - **减少开发桩/驱动器的数量**
  - **与以构建和合成为特征的开发匹配得很好**
    - 例如，邻居序列可以用于定义构建
- **缺点：**
  - **缺陷隔离问题，**尤其是对有大量邻居的情况
  - 清除缺陷后，意味着以前测试过的包含已变更代码的邻居，都需要重新进行测试

主要内容

- 集成测试概述
- 集成测试的主要方法
  - 基于分解的集成
  - 基于调用图的集成
  - 基于路径的集成
- 案例研究

基于路径的集成

- 将集成测试的侧重点由测试单独开发并通过测试的单元之间的接口，转移到这些单元的交互上，即它们的“协同功能”上。接口是结构性的，而交互是功能性的。

新概念与扩展概念

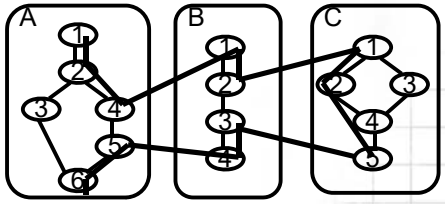
- 定义：
  - 源节点：程序执行开始或重新开始处的语句片段
    - 单元中的第一个可执行语句显然是源节点
    - 源节点还会出现在紧接转移控制到其他单元的节点之后
  - 汇节点：程序执行结束处的语句片段
    - 程序中的最后一个可执行语句显然是汇节点
    - 转移控制到其他单元的节点也是汇节点

新概念与扩展概念

- 定义：
  - 模块执行路径：以源节点开始、以汇节点结束的一系列语句，中间没有插入汇节点
  - 消息：一种程序设计语言机制，通过这种机制一个单元将控制转移给另一个单元
  - MM-路径：穿插出现模块执行路径和消息的序列
    - 基本思想是，描述包含在单独单元之间控制转移的模块执行路径序列。这种转移是通过消息完成的，因此MM-路径永远是可行执行路径，并且这些路径要跨越单元边界。在经过扩展的程序图中可以发现MM-路径，其中的节点表示模块执行路径，边表示消息

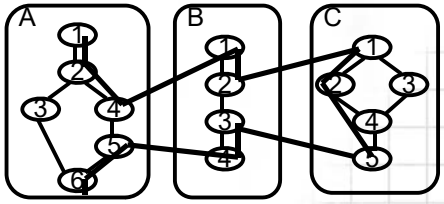
MM-路径的集成

- 实例：模块A调用模块B，模块B调用模块C
  - 模块A
    - 源节点：1、5
    - 汇接点：4、6
  - 模块B
    - 源节点：1、3
    - 汇接点：2、4
  - 模块C
    - 源节点：1
    - 汇接点：5



MM-路径的集成

- 案例：七条模块执行路径，MEP(模块号，路径编号)
  - MEP(A，1)=<1，2，3，6>
  - MEP(A，2)=<1，2，4>
  - MEP(A，3)=<5，6>
  - MEP(B，1)=<1，2>
  - MEP(B，2)=<3，4>
  - MEP(C，1)=<1，2，4，5>
  - MEP(C，2)=<1，3，4，5>

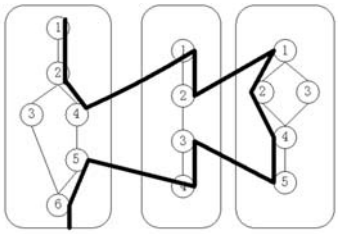


MM-路径的集成

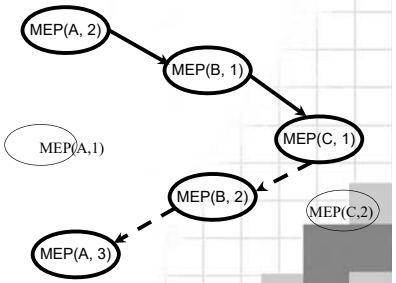
- 定义： 给定一组单元，其MM-路径图是一种有向图，图中的节点表示模块执行路径，边表示消息以及单元之间的返回
- 实线箭头表示消息，相应的返回由虚线箭头表示

9.5.2 MM-路径的集成

- 实例
  - 模块执行路径
    - MEP(A，1)=<1，2，3，6>
    - MEP(A，2)=<1，2，4>
    - MEP(A，3)=<5，6>
    - MEP(B，1)=<1，2>
    - MEP(B，2)=<3，4>
    - MEP(C，1)=<1，2，4，5>
    - MEP(C，2)=<1，3，4，5>
  - MM-路径图
    - 实线表示调用
    - 虚线表示返回



跨三个单元的 MM-路径



分析

- 模块执行路径、程序路径、DD-路径和MM-路径之间的关系：
  - 程序图是DD-路径序列
  - MM-路径是模块执行路径序列
  - DD-路径和模块执行路径之间没有简单的对应关系
  - 由于MM-路径实现超出单元边界的功能，因此确实有一种关系: MM-路径与单元的交叉。这种交叉中的模块执行路径，是模块执行路径所在单元的功能约束

分析

- MM-路径定义需要具有某种实际指导方针MM-路径有多长(“深”可能更确切)？
- 消息静止可以帮助解决这个问题
- 当到达不发送消息的节点/单元时，消息静止发生
  - 如示例中的模块C
- 也就是说，消息静止点可以作为MM路径的“终点”
- 消息静止点是MM-路径的自然端点

示例 --- SATM系统的伪代码（1）

```
1.  Main Program
2.  State = A wait Card
3.  Case State
4.  Case 1: A wait Card
5.      Screen Driver(1, null)                          msg1
6.      Watch Card Slot (Card Slot Status)              msg2
7.      Do While Card Slot Status is Idle
8.              Watch Card Slot (Card Slot Status)      msg3
9.      End While
10.     Control Card Roller (accept)                    msg4
11.     Validate Card ( Card OK，PAN)                           msg5
12.     If  Card OK
13.             Then State = A wait PIN
14.             Else Control Card Roller (eject)                msg6
15.     End If
16.     State = A wait Card
```


SATM系统的伪代码（2）

```
17. Case 2: A wait PIN
18.     Validate PIN ( PIN ok，PAN)                     msg7
19.     If  PIN ok
20.             Then Screen Driver (2, null)            msg8
21.                     State = A wait Trans
22.             Else Screen Driver (4, null)            msg9
23.     End If
24.     State = A wait Card
25. Case 3: AwaitTrans
26.     Manage Transaction                              msg10
27.     State = CloseSession
28. Case 4: CloseSession
29.     If  New Transaction Request
30.             Then State = AwaitTrans
31.             Else Print Receipt                      msg11
32.     End If
33.     Post Transaction Local                          msg12
34.     Close Session                                   msg13
35.     Control Card Roller (eject)                     msg14
36.     State = A wait Card
37. End Case (State)
38. End. (Main program SATM)
```

SATM系统的伪代码（3）

```
1
39. Procedure Validate PIN (PIN ok, PAN)
40. Get PIN for PAN (PAN, Expected PAN)                    msg15
41. Try = First
42. Case Try of
43. Case 1: First
44.         Screen Driver (2, null)                        msg16
45.         Get PIN (Entered PIN)                          msg17
46.         If Entered PIN = Expected PIN
47.                     Then PIN ok = True
48.                     Else Screen Driver (3, null)       msg18
49.         End If
50.         Try = Second
51. Case 2: Second
52.         Screen driver (2, null)                        msg19
53.         Get PIN (Entered PIN)                          msg20
54.         If Entered PIN = Expected PIN
55.                     Then PIN ok = True
56.                     Else Screen Driver (3, null)       msg21
57.         End If
58.         Try = Third
59. Case 3: Third
60.         Screen Driver (2, null)                        msg22
61.         Get PIN (Entered PIN)                          msg23
62.         If Entered PIN = Expected PIN
63.                     Then PIN ok = True
64.                     Else Screen Driver (4, null)       msg24
65.                             PIN ok = False
66.         End If
67. End Case (Try)
68. End. (Procedure Validate PIN)
```

SATM系统的伪代码（4）

```
69. Procedure Get PIN (Entered PIN，  Cancel Hit)
70. Local Data: Digit Keys = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
71. Cancel hit = False
72. Entered PIN = null string
73. Digits Rcvd = 0
74. Do While NOT (DigitsRcvd=4 OR Cancel hit)
75. Key Sensor (Key Hit)                                   msg25
76. If Key Hit IN Digit Keys
77.         Then
78.                     EnteredPIN = EnteredPIN + Key Hit
79.                     INCREMENT (DigitsRcvd)
80.                     If Digits Rcvd = 1
81.                     Then Screen Driver (2, X---')      msg26
82.                     End If
83.                     If Digits Rcvd = 2
84.                     Then Screen Driver (2, ‘XX--')     msg27
85.                     End If
86.                     If Digits Rcvd = 3
87.                     Then Screen Driver (2, ‘XXX-')     msg28
88.                     End If
89.                     If Digits Rcvd = 4
90.                     Then Screen Driver (2, ‘XXXX')     msg29
91.                     End If
92.         Else
93.                     Cancel hit = True
94.         End If
95. End While
96. End. (Procedure Get PIN)
```

西安文大软件学院

SATM系统中的MM-路径

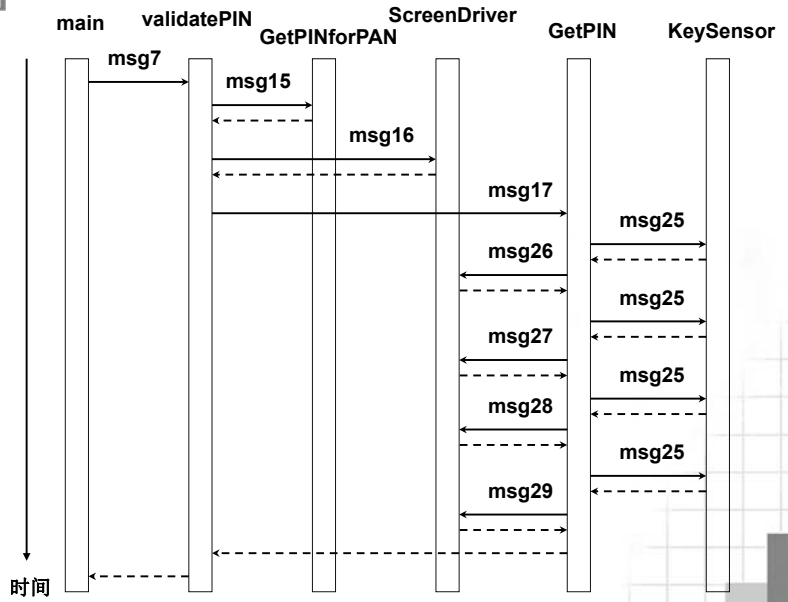
第一次尝试正确PIN输入的MM-路径

```
Main (1,2, 3, 17, 18)
msg 7
ValidatePIN (39, 40)
msg 15
GetPINforPAN (no pseudo-code given)
ValidatePIN (41, 42, 43, 44)
msg 16
ScreenDriver (no pseudo-code given)
ValidatePIN (45)
msg 17
GetPIN(69, 70, 71, 72, 73, 74, 75)
msg 25
KeySensor (no pseudo-code given)      'first digit
GetPIN (76, 77, 78, 79, 80, 81)
msg 26
ScreenDriver (no pseudo-code given)
GetPIN (82, 83, 85, 86, 88, 89, 91, 94, 95, 74, 75)
msg 25
```

SATM系统中的MM-路径(续)

```
KeySensor (no pseudo-code given)   'second digit
GetPIN (76, 77, 78, 79, 80, 82, 83, 84)
msg 27
ScreenDriver (no pseudo-code given)
GetPIN (85, 86, 88, 89, 91, 94, 95, 74, 75)
msg 25
KeySensor (no pseudo-code given) 'third digit
GetPIN (76, 77, 78, 79, 80, 82, 83, 85, 86, 87)
msg 28
ScreenDriver (no pseudo-code given)
GetPIN (88, 89, 91, 94, 95, 74, 75)
msg 25
KeySensor (no pseudo-code given))     'fourth digit
GetPIN (76, 77, 78, 79, 80, 82, 83, 85, 86, 88, 89, 90)
msg 29
ScreenDriver (no pseudo-code given)
GetPIN (91,94, 95, 74, 96)
ValidatePIN (46, 47, 50, 67, 68)
 Main(19)
```

第一次PIN正确MM-路径的UML序列图



MM-路径图

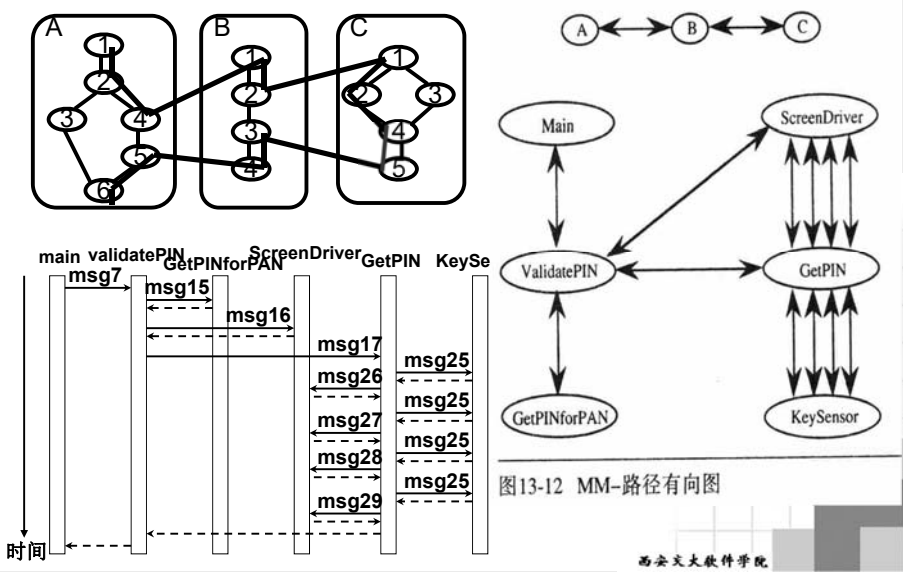


图13-12 MM-路径有向图

MM-路径复杂度

■圈复杂度公式为：

V(G) = e – n + 2p

其中， p是强连接区域的个数

对于结构化过程代码，永远有p = 1

■右图的圈复杂度分别为：

V(G) = 4 – 3 + 2 = 3

V(G) = 24 – 6 + 2 = 20

注：一个双向箭头代表两条边

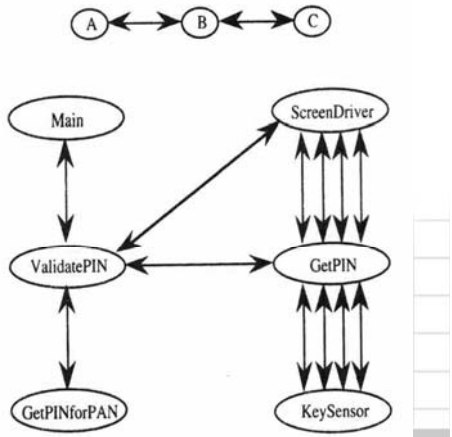


图13-12 MM-路径有向图

基于路径的集成总结

■MM-路径是功能性测试和结构性测试的一种混合

- 在表达输入和输出行动上， MM-路径是功能性的，因此所有功能性测试技术都是潜在可使用的
- 在标识方式上，特别是MM-路径图的标识方式上，它是结构性的

■优点：

- 与实际系统行为密切匹配，而不是靠基于分解和调用图集成的结构性推动
- 适用性广泛，各种软件开发模型都适用

■缺点：

- 需要更多工作量来标识MM-路径

练习

- 对于SATM系统来讲
- 1.给出三次尝试错误PIN的MM-路径
- 2.给出一次尝试，输入两个数字后，cancel的MM-路径

主要内容

- 集成测试概述
- 集成测试的主要方法
  - 基于分解的集成
  - 基于调用图的集成
  - 基于路径的集成
- 案例研究

案例研究

- 重新编写NextDate函数，将主程序的功能分解为过程和函数
- 这个“集成版本”有一点扩展：对月、日、年增加了有效性检查
- 伪代码从50行语句增长到81行语句

NextDate的伪代码(1)

```
. Main integration Next Date
   Type Date
     Month As Integer
     Day As Integer
     Year As Integer
   End Type
   Dim today As Date
   Dim tomorrow As Date
2.  Get Date(today)                          msg1
3.  Print Date(today)                        msg2
4.  tomorrow = Increment Date(today)              msg3
5.  Print Date(tomorrow)                     msg4
6. End Main
7. Function is Leap (year)  Boolean
8.  If (year divisible by 4)
9.  Then
10.         If (year is NOT divisible by 100)
11.         Then is Leap = True
12. Else
13.         If (year is divisible by 400)
14.         Then is Leap = True
15.         Else is Leap = False
16.         End If
17.         End If
18.         Else is Leap = False
19. End If
20. End (Function is Leap)
```

NextDate的伪代码(2)

```
21.  Function last Day Of month (month, year) Integer
22.  Case month Of
23.   Case 1: 1,3,5,7,8,10,12
24.           last Day Of month = 31
25.  Case 2: 4,6,9,11
26.           last Day Of month = 30
27.  Case 3: 2
28.           If (is Leap(year))                          msg5
29.           Then last Day Of month = 29
30.           Else last Day Of month = 28
31.           End If
32.   End Case
33.  End (Function last Day Of month)
34. Function Get Date (aDate) Date
      dim aDate As Date
      Function Validate Date (aDate)  Boolean 'within scope of Get Date
      dim aDate As Date
      dim day OK, month OK, year OK As Boolean
36.  If (aDate.Month > 0) AND (aDate.Month <= 12)
37.     Then month OK = True
38.     Else month OK = False
39.   End If
```


NextDate的伪代码(3)

```
40.    If  (month OK)
41    Then
42.     If(aDate.Day>0)AND(aDate.Day<=lastDayOfmonth(aDate.Month,aDate.Year))  msg6
43.  Then day OK = True
44.  Else day OK = False
45.  End If
46.    End If
47.    If (aDate.Year>1811)AND (aDate.Year<= 2012)
48.    Then year OK = True
49.     Else year OK = False
50.    End If
51.    If (month OK AND day OK AND year OK)
52.    Then Validate Date = True
53.     Else Validate Date = False
54.     End If
55.  End (Function Validate Date)
```


NextDate的伪代码(4)

```
    ‘ Get Date body begins here
56.  Do
57.           Output ("enter a month")
58.           Input (aDate.Month)
59.           Output ("enter a day")
60.           Input (aDate.Day)
61.           Output ("enter a year")
62.           Input (aDate.Year)
63.           GetDate.month = aDate.month
64.           GetDate.day = aDate.Day
65.           GetDate.Year = aDate.Year
66.  Until (Validate (aDate))                  msg7
67. End (Function Get Date)

68. Function Increment Date (aDate) Date
69.  If (aDateDay < lastDayOfmonth(aDate.Month, aDate.Year))          msg8
70.    Then aDate.Day = aDate.Day + 1
71     Else aDate.Day = 1
72.          If (aDate.Month = 12)
73.            Then aDate.Month = 1
74.                  aDate.Year = aDate.Year + 1
75.            Else aDate.Month = aDate.Month + 1
76.          End If
77.   End If
78. End (Increment Date)
79. Procedure Print Date (aDate)
80. Output ( "Day is ",  aDate.Month,  "/",  aDate.Day,  "/",  aDate.Year)
81. End (PrintDate)
```


NextDate集成版本的单元程序图

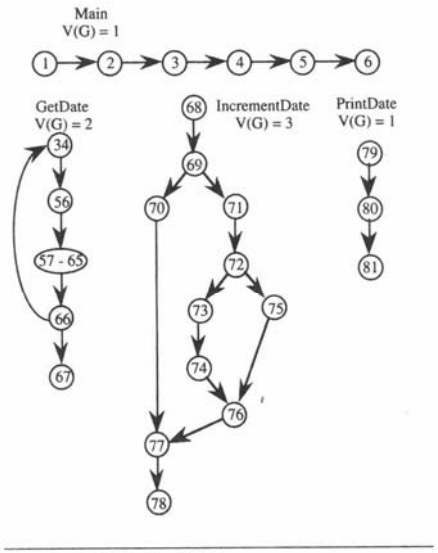


图13-13 主程序和第一层单元

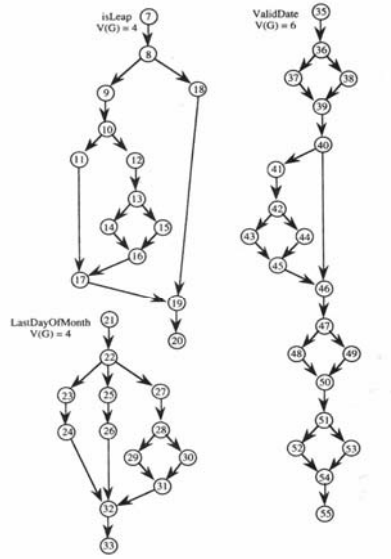


图13-14 低层单元

NextDate基于分解的集成

讨论：1.根据下面功能分解图，其基于分解的集成过程是怎样的？自底向上、自顶向下、三明治、大爆炸
2.是否可以采用基于分解的成对集成？

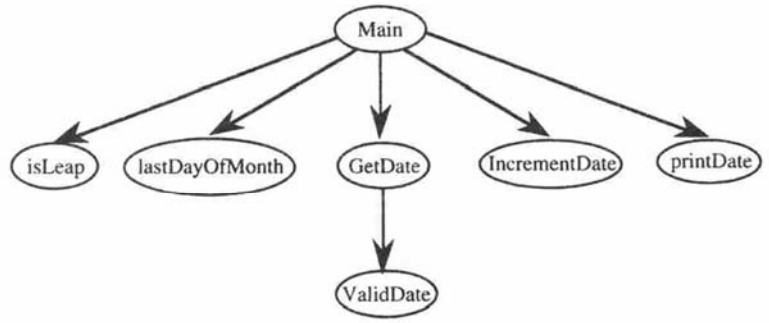


图13-15 集成版本的功能分解

NextDate基于调用图的集成

讨论：1.根据下面调用图，其基于调用图的集成过程是怎样的？成对集成、相邻集成

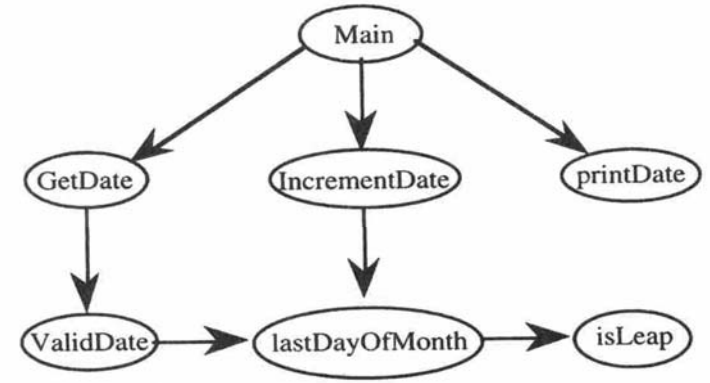


图13-16 集成版本的调用图

NextDate基于MM-路径的集成

■ 由于程序是数据驱动的，因此所有MM-路径都要从主程序开始，并回到主程序。以下是2002年5月27日的第一条MM-路径(当主程序调用PrintDate和IncrementDate时，还有其他 MM-路径)

```
Main (1,2)
        msg1
        Get Date (34,56,57,58,59,60,61,62,63,64,65,66)
            msg7
            valid Date (35,36,37,39,40,41,42)
               msg6
               1astDayOfMonth (21,22,23,24,32,33)      ’point of message quiessence
            Valid Date (43,45,46,47,48,50,51,52,54,55)
     Get Date (67)
   Main (3)
```

NextDate基于MM-路径的集成

■讨论：请给出其他MM-路径图

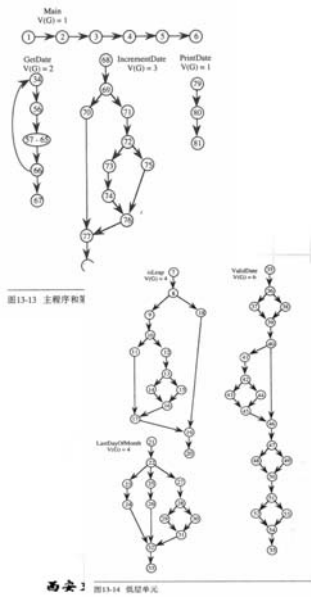
■如：2000年2月28日

■ 1995年2月28日

■ 2010年12月31日

NextDate基于MM-路径的集成

- 请注意，语句片段序列(如图13-13和13-14所示)标识出从源节点到汇节点的所有路径：在消息静止点上这是直接的，在其他单元中，节点序列对必须级联起来，构成完整的从源节点到汇节点的路径
- 需要多少条MM-路径：MM-路径集合应该覆盖单元集合中所有从源到汇节点的路径
- 如果存在循环，则要进行压缩，产生有向无环路图，因此可解决潜在无限(或非常多)条路径问题



总结

- 基于分解的集成都有哪些方法？
- 基于调用图的集成都有哪些方法？
- MM-路径是什么？

补充：集成测试过程

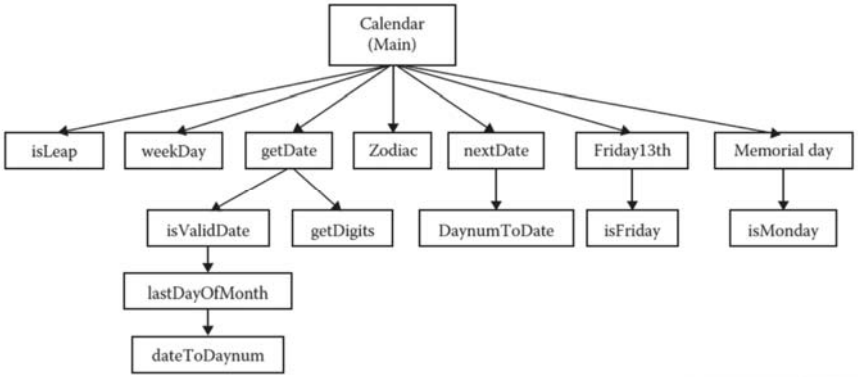
- 与单元测试类似，主要的测试活动包括
  - 集成测试计划
  - 设计集成测试用例
  - 实现测试用例
  - 搭建集成测试环境
  - 执行测试
  - 测试总结
  - 评估测试工作量
- 很多时候采用黑盒和白盒相结合被称为灰盒测试的测试方法
- 集成测试可由开发人员也可由测试人员承担

练习

- 日历程序 Calendar：
- 输入(一个日期)：mm, dd, yyyy
- 提供的功能：
  - 计算下一天NextDate
  - 计算星期几
  - 计算星座
  - 计算最近一次纪念日为5月27日的年
  - 计算最近一次13号是星期五的日期（黑色星期五）
- 用本章知识进行集成测试

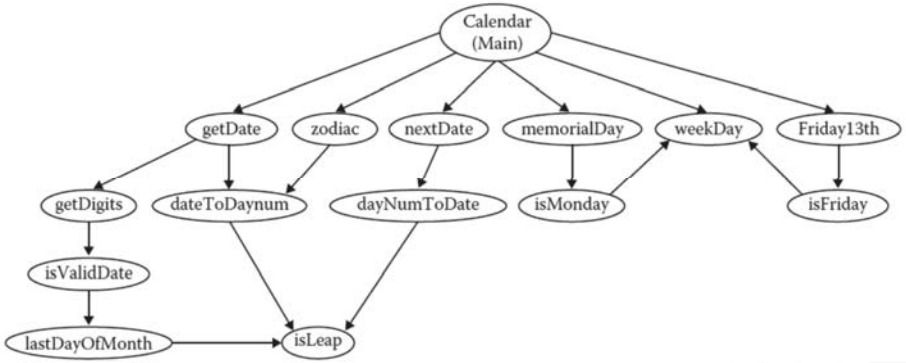
练习

- 日历程序 Calendar：
- 功能分解图



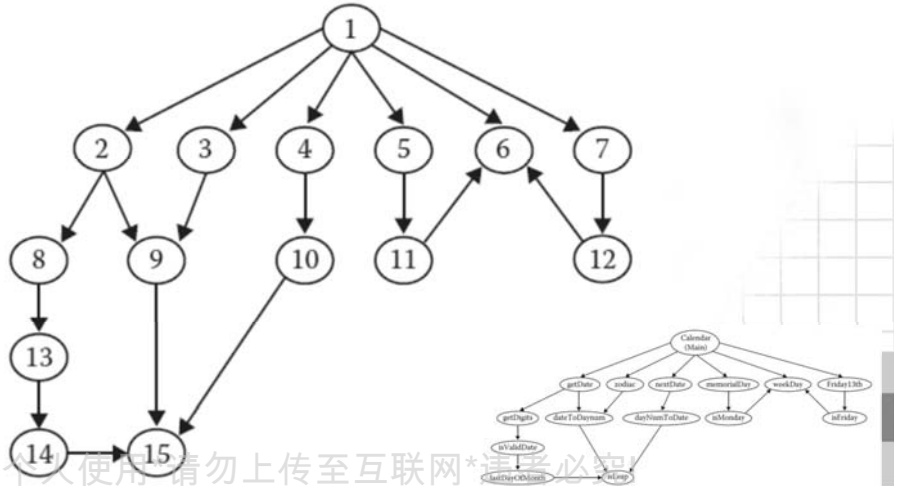
练习

- 日历程序 Calendar：
- 调用图



练习

- 日历程序 Calendar：
- 调用图，用数字表示单元



练习

- 日历程序 Ca
- 调用图中的

Neighborhoods in Calendar Program Call Graph			
Node	*Unit Name*	*Predecessors*	*Successors*
1	Calendar (Main)	(None)	2, 3, 4, 5, 6, 7
2	getDate	1	8, 9
3	zodiac	1	9
4	nextDate	1	10
5	memorialDay	1	11
6	weekday	1, 11, 12	(None)
7	Friday13th	1	12
8	getDigits	2	13
9	dateToDayNum	3	15
10	dayNumToDate	4	15
11	isMonday	5	6
12	isFriday	7	6
13	isValidDate	8	14
14	lastDayOfMonth	13	15
15	isLeap	9, 10, 14	(None)