北京科技大学
University of Science and Technology Beijing

数据流测试

数据流测试

- 数据流测试是一种考虑变量的定义和使用情况的路径测试技术，和数据流图没有关系
  - 变量的定义（def）：如果语句n对变量v的值进行了修改，则称n为变量v的定义语句，记做DEF(v,n)
  - 变量的使用（use）：如果语句n对变量v的值进行了访问，则称n为变量v的使用语句，记做USE(v,n)
    - 如果对变量的访问出现在谓词中，则成为谓词使用
    - 如果对变量的访问出现在计算中，则成为计算使用

数据流测试

- 定义-使用路径（def-use path）：
  - 如果程序（图）中存在一条路径p，对某个变量v，p的起点m为DEF(v,m)，p的终点n为USE(v,n)，则称p为关于v的定义-使用路径
- 定义-清除路径（def-clear path）：
  - 如果程序（图）中的一条路径p是关于变量v的定义-使用路径，且p中除了起点外，不存其它关于v的定义语句（定义节点），则称p是关于变量v的定义-清除路径

数据流测试的覆盖指标

- 设T是程序图G中的一个路径集合，V表示程序中的所有变量
  - 全定义：对于任意v∈V，如果T包含从v的所有定义节点到v的一个使用节点的定义清除路径（不要求覆盖所有v的使用节点）
  - 全使用：对于任意v∈V，如果T包含从v的所有定义节点到v的所有使用节点的定义清除路径 但不笛卡尔积
    - 注意不是求v的所有定义节点到v的所有使用节点的笛卡尔积，因为有些路径是不可实现的

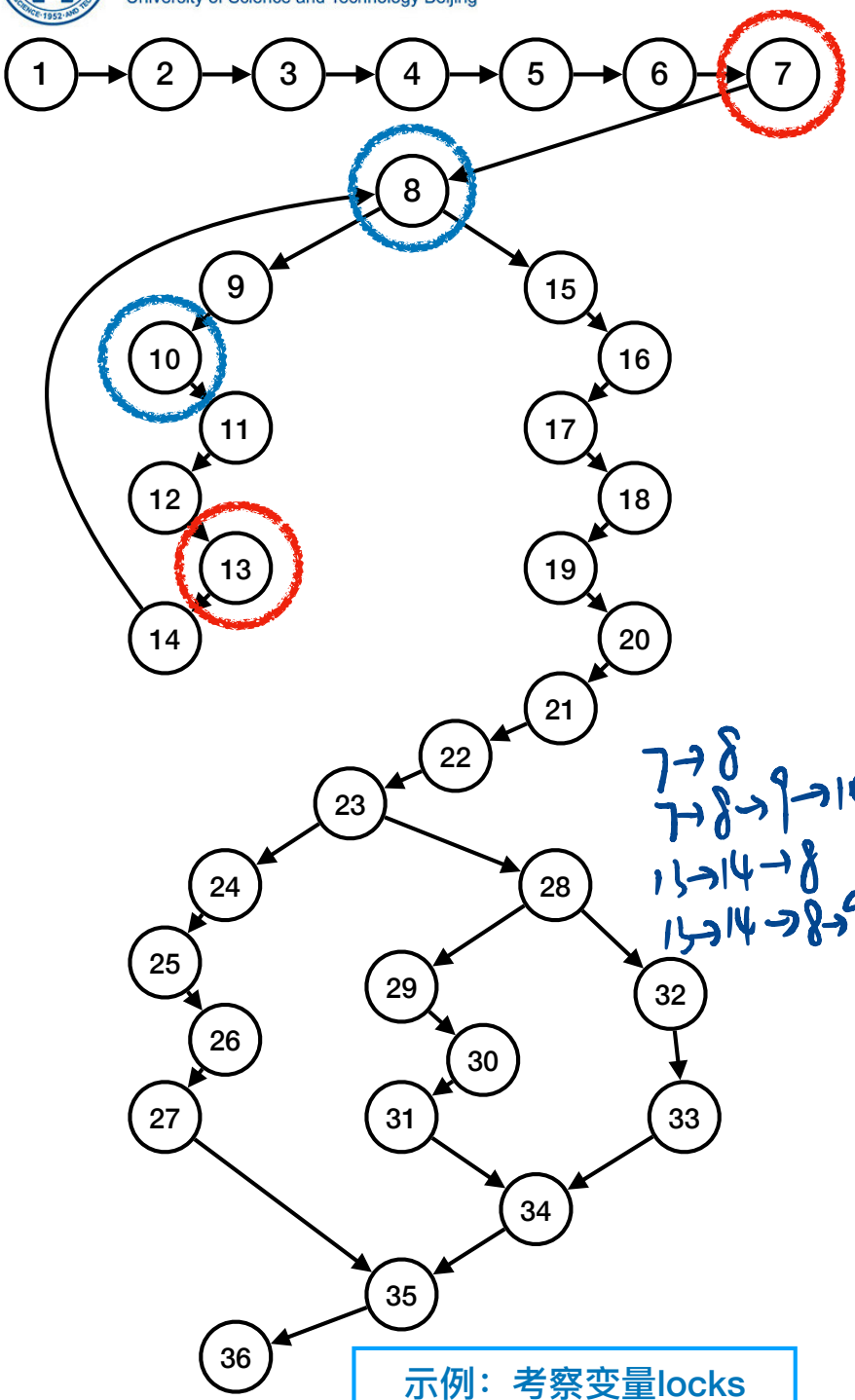
数据流测试的覆盖指标

使用

- 设T是程序图G中的一个路径集合，V表示程序中的所有变量

  - 全计算使用-部分谓词使用：对于任意v∈V，如果T包含从v的所有定义节点到v的所有计算使用节点的定义清除路径（若v的一个定义没有计算使用，那么T至少包含一条到v的谓词使用的定义清除路径）

  - 全谓词使用-部分计算使用：对于任意v∈V，如果T包含从v的所有定义节点到v的所有谓词使用节点的定义清除路径（若v的一个定义没有谓词使用，那么T至少包含一条到v的计算使用的定义清除路径）



示例：考察变量locks

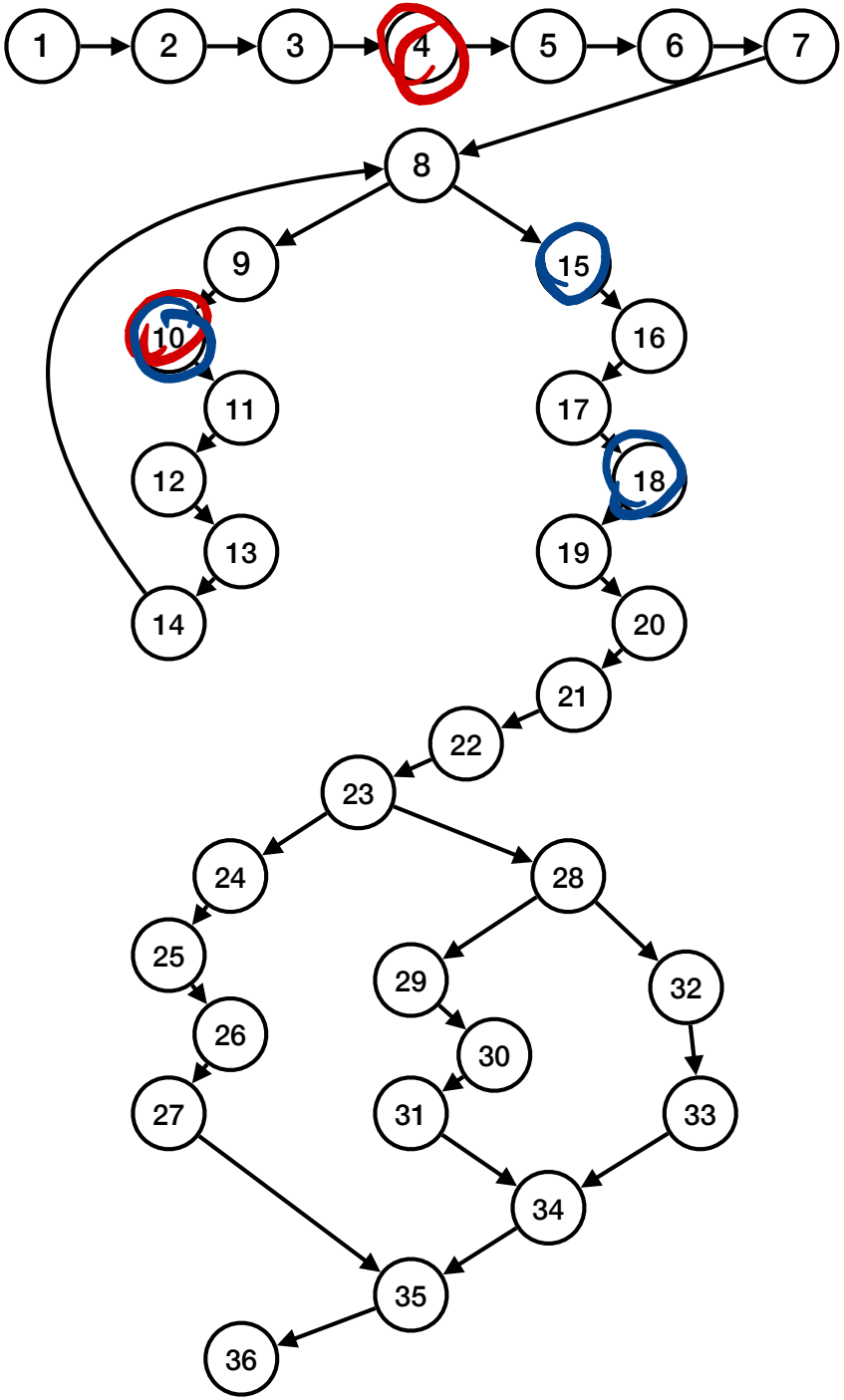
```
Program Commission()
 1. lockPrice = 45.0
 2. stockPrice = 30.0
 3. barrelPrice = 25.0
 4. totalLocks = 0
 5. totalStocks = 0
 6. totalBarrels = 0
 7. input(locks)
 8. WHILE NOT (locks=-1)
 9.   input(stocks, barrels)
10.   totalLocks = totalLocks + locks
11.   totalStocks = totalStocks + stocks
12.   totalBarrels = totalBarrels + barrels
13.   input(locks)
14. ENDWHILE
15. print('Locks sold:', totalLocks)
16. print('Stocks sold:', totalStocks)
17. print('Barrels sold:', totalBarrels)
18. lockSales = lockPrice * totalLocks
19. stockSales = stockPrice * totalStocks
20. barrelSales = barrelPrice * totalBarrels
21. sales = lockSales + stockSales + barrelSales
22. print('Total sales:', sales)
23. IF sales>1800.0
24.   THEN
25.     commission = 0.10 * 1000.0
26.     commission = commission + 0.15 * 800.0
27.     commission = commission + 0.2 * (sales - 1800.0)
28.   ELSE IF sales > 1000.0
29.     THEN
30.       commission = 0.10 * 1000.0
31.      commission = commission + 0.15 * (sales - 1000.0)
32.    ELSE
33.      commission = 0.10 * sales
34.    ENDIF
35. ENDIF
36. print('Commission is $:', commission)
```

根据lock（枪机）、stock（枪托）、barrel（枪管）的销售数量计算销售额并计算comission（佣金）的程序

def-use path: 7→8, 7→10
13→14→8
使用 13→14→8→9→10
...

7→8
7→8→9→10
13→14→8
13→14→8→9→10

①③ 全定义：7→8, 13→14→8
全使用：1, 4
全谓词：1, 3
全计算 2, 4



定义清除路径

课堂练习：考察变量totalLocks

```
Program Commission()
 1. lockPrice = 45.0
 2. stockPrice = 30.0
 3. barrelPrice = 25.0
 4. totalLocks = 0
 5. totalStocks = 0
 6. totalBarrels = 0
 7. input(locks)
 8. WHILE NOT (locks=-1)
 9.   input(stocks, barrels)
10.   totalLocks = totalLocks + locks
11.   totalStocks = totalStocks + stocks
12.   totalBarrels = totalBarrels + barrels
13.   input(locks)
14. ENDWHILE
15. print('Locks sold:', totalLocks)
16. print('Stocks sold:', totalStocks)
17. print('Barrels sold:', totalBarrels)
18. lockSales = lockPrice * totalLocks
19. stockSales = stockPrice * totalStocks
20. barrelSales = barrelPrice * totalBarrels
21. sales = lockSales + stockSales + barrelSales
22. print('Total sales:', sales)
23. IF sales>1800.0
24.   THEN
25.     commission = 0.10 * 1000.0
26.     commission = commission + 0.15 * 800.0
27.     commission = commission + 0.2 * (sales - 1800.0)
28.   ELSE IF sales > 1000.0
29.     THEN
30.       commission = 0.10 * 1000.0
31.       commission = commission + 0.15 * (sales - 1000.0)
32.     ELSE
33.       commission = 0.10 * sales
34.     ENDIF
35. ENDIF
36. print('Commission is $:', commission)
```

2条

4→15

4→18

10→15

10→18

基于模型的测试

基于模型的测试

汽车、航空航天

- 应用场景
  - 在系统测试阶段，待测程序（即整个系统）过于复杂，以至于无法利用之前介绍的技术进行测试，这时可以使用基于模型的测试技术
- 基本思想
  - 构造待测系统的模型（可以是分析、设计模型等）
  - 根据系统模型产生测试用例
  - 如果模型刻画了系统的（部分）行为，则可以此作为测试断言

核心点！
连接分析/设计和测试阶段

基于模型的测试

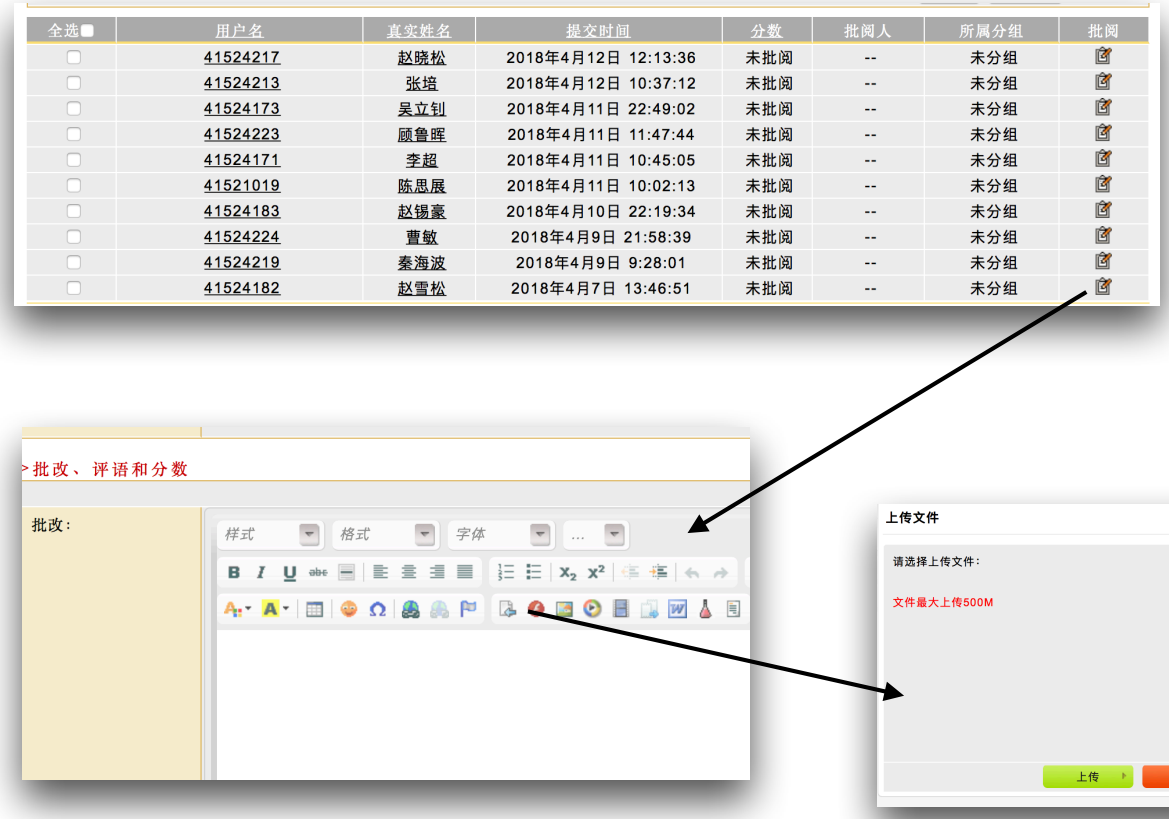
- 如何进行系统抽象
  - 业务流程模型/活动图
  - 状态机模型
  - 数据流模型
  - 用况模型
  - 类图
  - ……

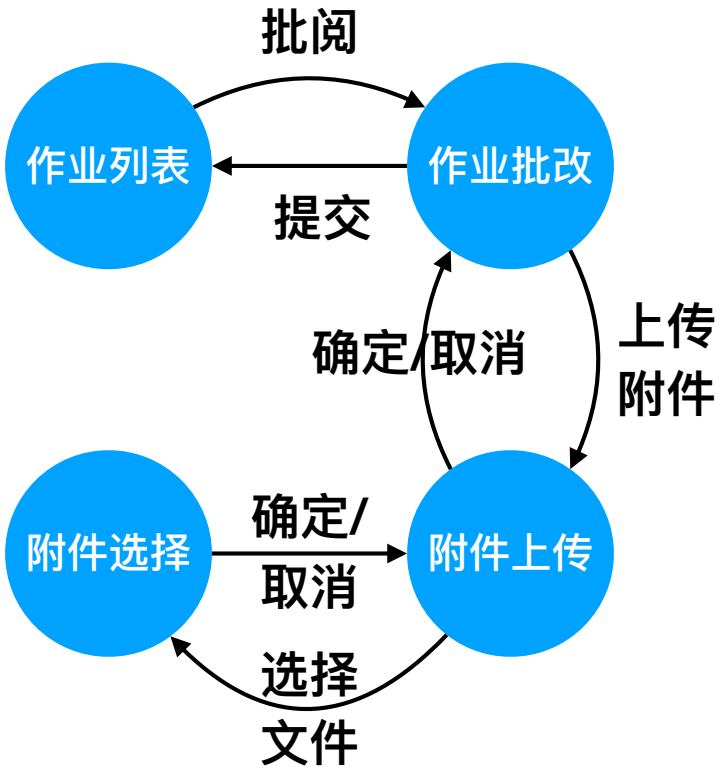
如何生成测试用例需要根据不同的模型定义不同的覆盖准则

基于模型的测试

- 例子：GUI测试

- 不同UI之间的切换可以表示成一个状态机模型

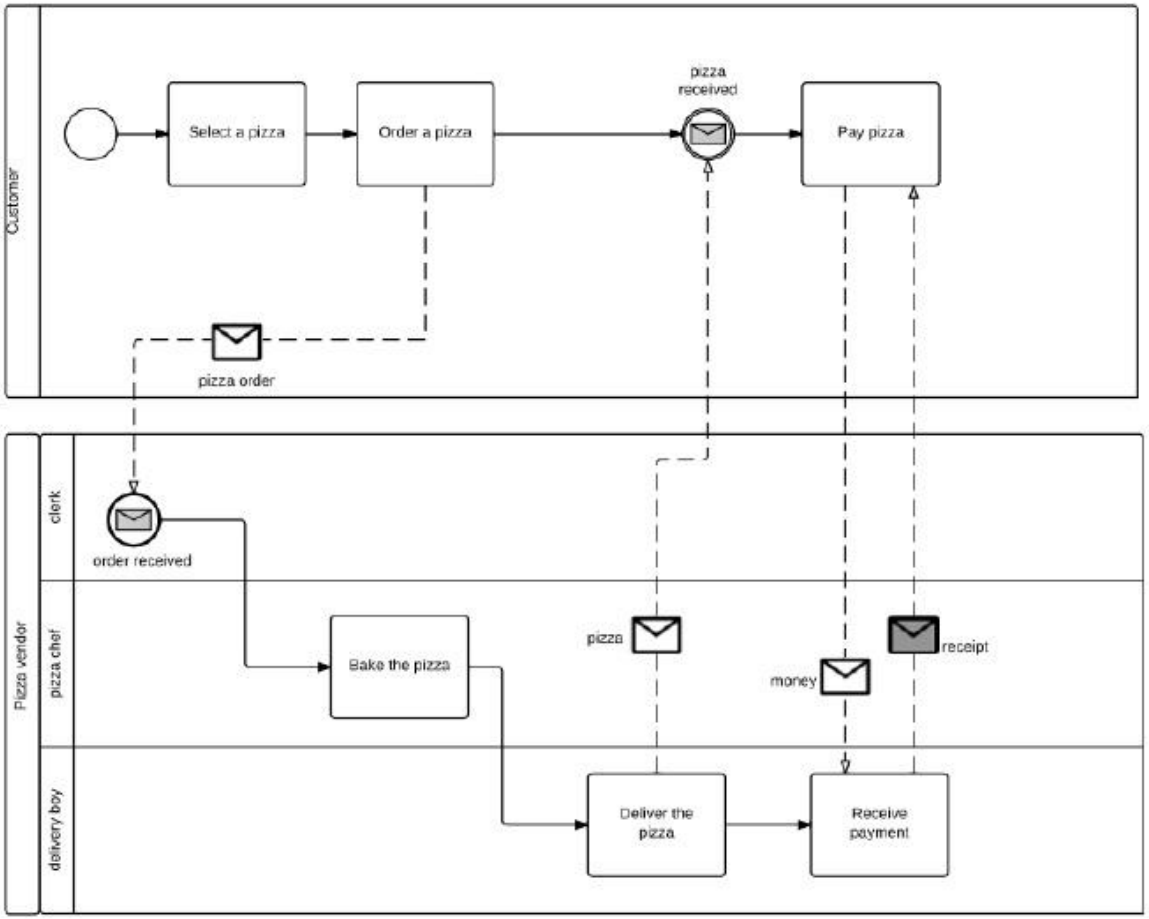




基于模型的测试

- 例子：Web服务测试
  - 一个Web服务系统实现的业务流程

验证系统是否实现了业务流程

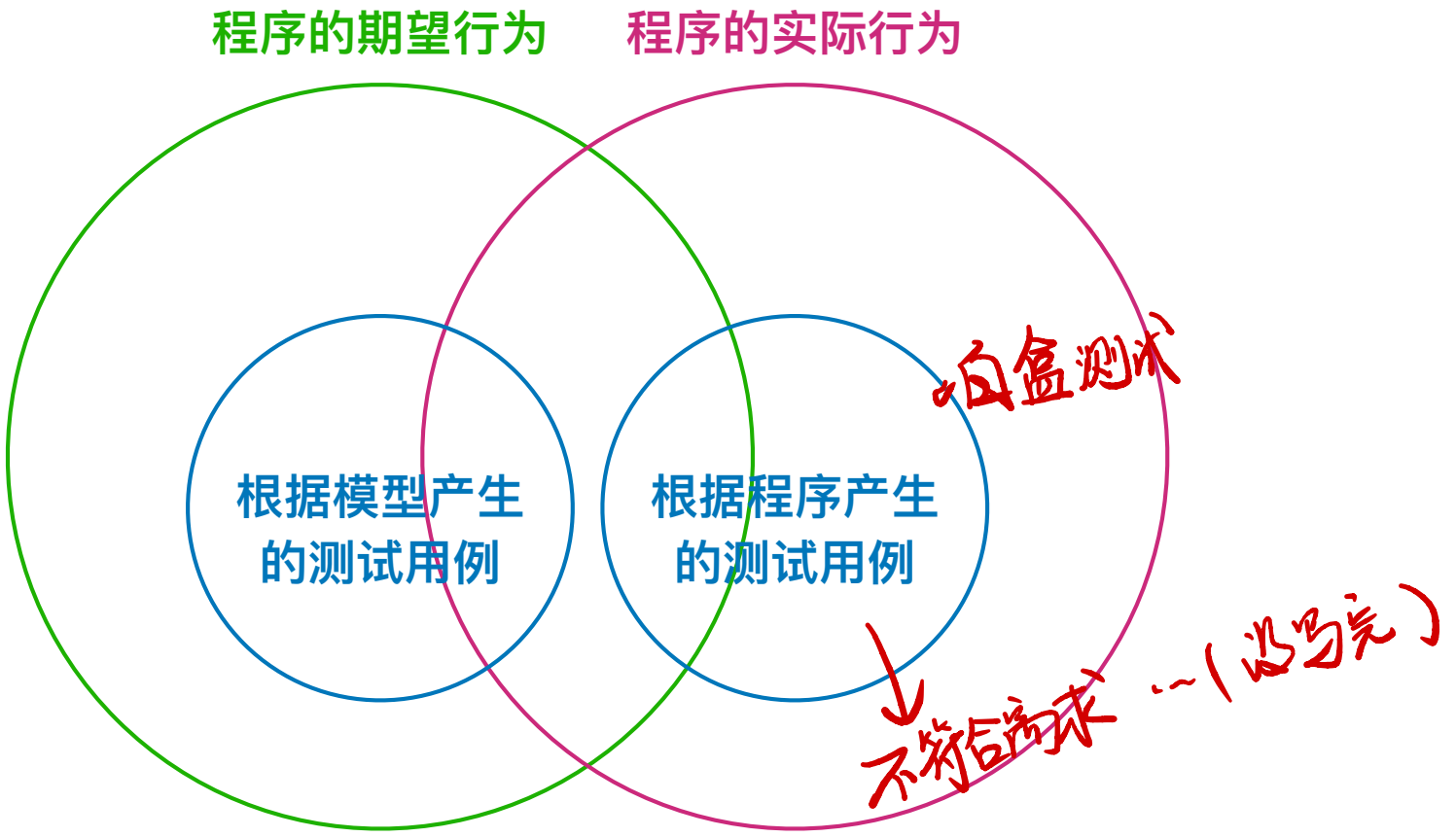


比较和讨论

- 路径测试
  - 抽象方法：程序图
  - 测试用例生成方法：根据路径覆盖准则产生测试用例
- 基于模型的测试
  - 抽象方法：软件模型
  - 测试用例生成方法（以状态机模型为例）：根据状态机路径覆盖准则产生测试用例

区别在哪里?

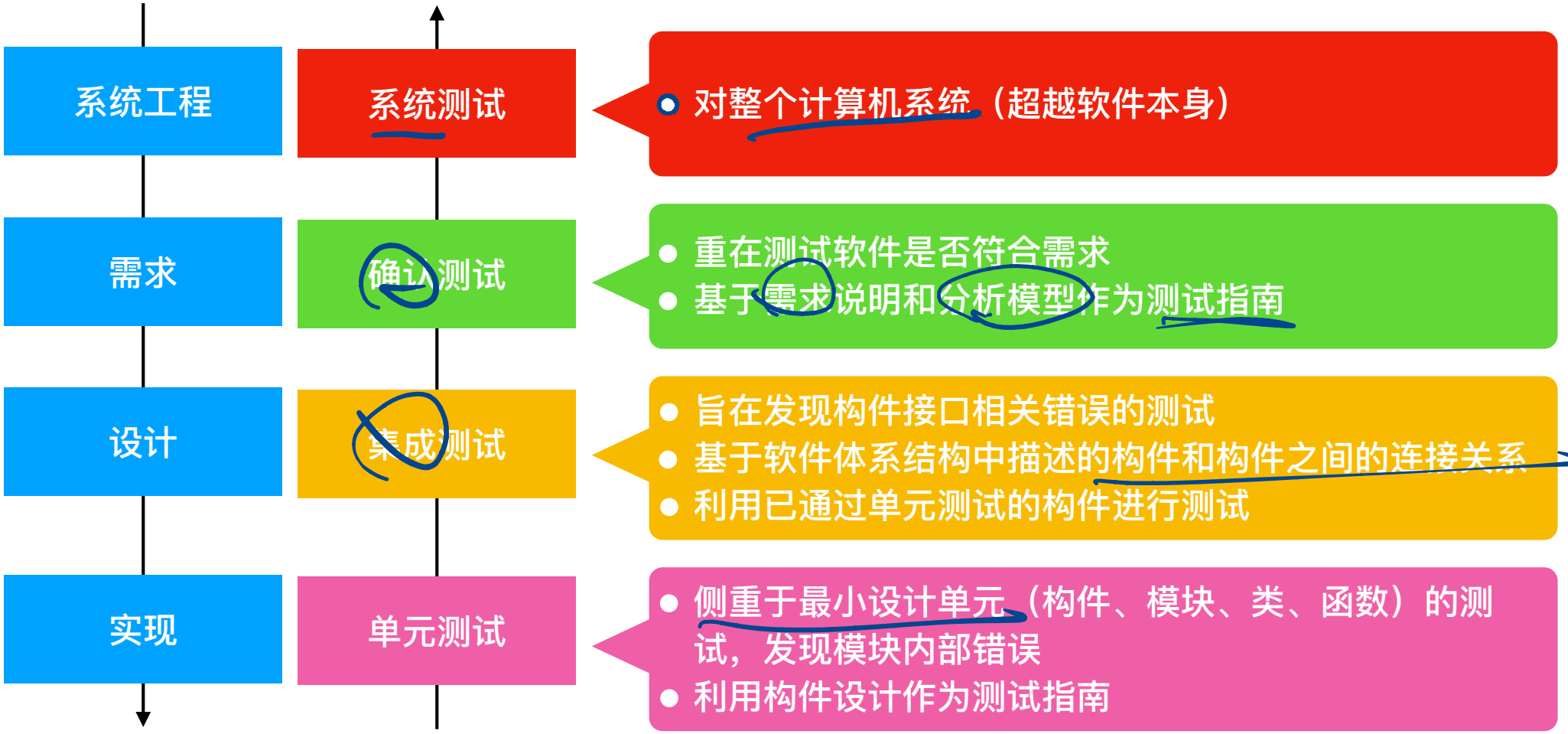
两种测试方式的比较



北京科技大学
University of Science and Technology Beijing

测试计划和组织
（策略）

测试的策略


系统工程
系统测试
对整个计算机系统（超越软件本身）
需求
确认测试
重在测试软件是否符合需求
基于需求说明和分析模型作为测试指南
设计
集成测试
旨在发现构件接口相关错误的测试
基于软件体系结构中描述的构件和构件之间的连接关系
利用已通过单元测试的构件进行测试
实现
单元测试
侧重于最小设计单元（构件、模块、类、函数）的测试，发现模块内部错误
利用构件设计作为测试指南

单元测试

1. 测试穿越模块接口的数据流
   - 对通过模块接口的数据流进行测试
     - 参数的数目、次序、属性等；
     - 是否修改了只作输入用的变元；
     - 全局变量的定义和用法在各个模块中是否一致
2. 局部数据结构
   - 设计测试方案，以便发现局部数据说明、初始化、默认值等方面的错误。
3. 重要的执行通路 (路径测试标准)
   - 选择最有代表性、最可能发现错误的执行通路进行测试
     - 不可能进行穷尽测试

单元测试

4. 边界条件

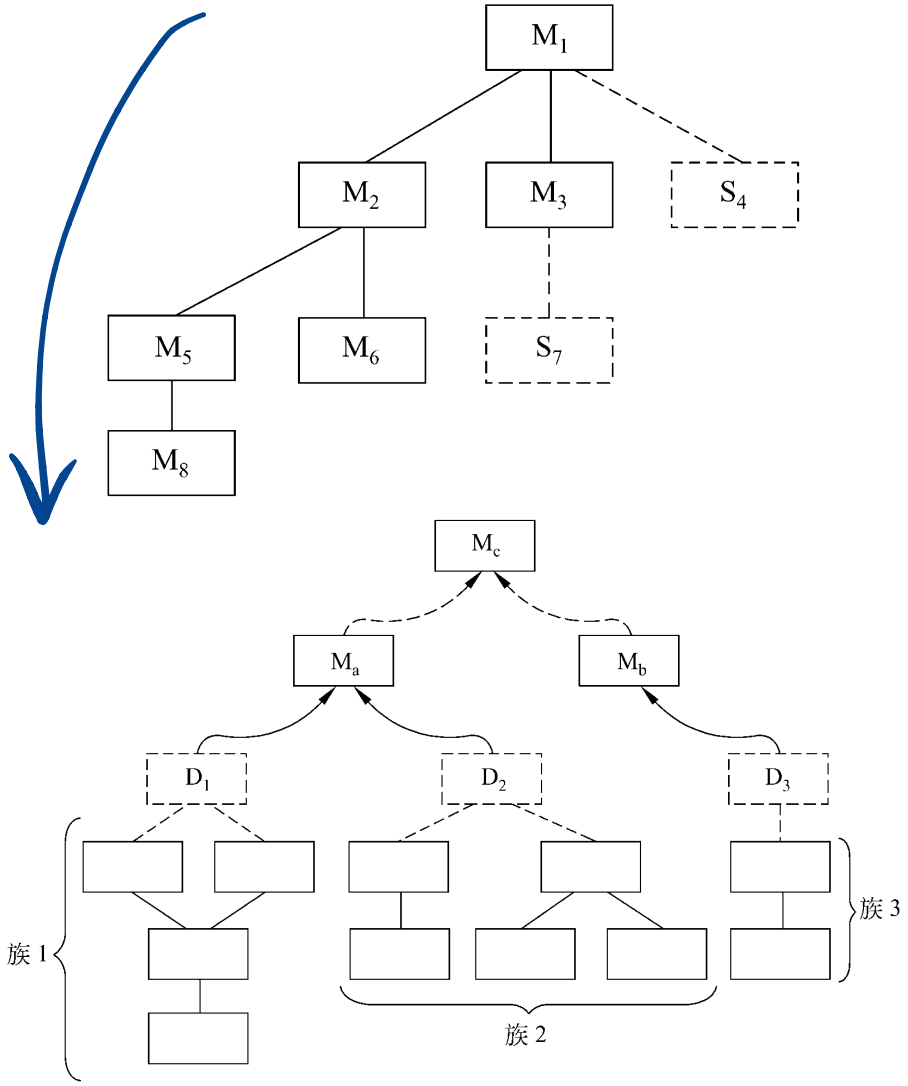
- 软件常常在它的边界上失效
  - 例如，处理n元数组的第n个元素时，或做到i次循环中的第i次重复时，往往会发生错误

5. 出错处理通路

- 对能够导致软件错误的情况进行测试，并检查出错后的错误处理情况
  i. 对错误的描述是难以理解的；
  ii. 记录的错误与实际遇到的错误不一致；
  iii. 在对错误进行处理之前，错误条件已经引起系统干预；
  iv. 对错误的处理不正确；
  v. 描述错误的信息不足以帮助确定造成错误的位置。

集成测试

- “一步到位”的集成 vs 增量集成
  - 自顶向下的集成测试
    - 从顶层主控模块开始测试，逐步将子模块加入到测试中来，未加入测试的子模块采用“桩子”替代
  - 自底向上的集成测试
    - 从底层的一组子模块开始测试，逐步将上层模块加入到测试中来，未加入测试的模块采用“测试驱动”替代



确认测试

- 确认测试也称为验收测试，它的目标是验证软件的有效性
  - 如果软件的功能和非工程需求如同用户所合理期待的那样，软件就是有效的
  - 需求分析阶段产生的软件需求规格说明书，准确地描述了用户对软件的合理期望，是软件有效性的标准，也是进行确认测试的基础
- 可能的结果
  - 功能和非功能性质与用户要求一致，软件是可以接受的；
  - 功能和非功能性质与用户要求有差距

确认测试

- α测试
  - 最终用户在开发者的场所进行，开发者通过观察用户的使用过程，记录错误和使用问题。
- β测试
  - 在最终用户场所执行，开发者不在场，不受开发者控制。

系统测试

- 恢复测试：通过强制系统发生错误，测试系统在崩溃后是否能够及时恢复
- 安全测试：测试系统是否能够抵抗安全攻击
- 压力测试：以反常的数量、频率或容量的方式执行系统，检查系统是否出现问题
- 性能测试：测试软件在运行时的性能表现

北京科技大学
University of Science and Technology Beijing

软件可靠性

可靠性与可用性

- 可靠性
  - 在给定时间间隔内，软件按照规格说明书的规定成功地运行的概率
  - P(0<t<T内没有失效)
- 可用性
  - 在给定时间点，软件按照规格说明书的规定成功地运行的概率
  - P(t=T时没有失效)

平均无故障时间

平均可用性 $A_{ss} = \dfrac{MTTF}{MTTF + MTTR}$

平均维护时间

估算平均无故障时间

- 平均无故障时间：MTTF

$$MTTF = \frac{1}{K(\frac{E_T}{I_T} - \frac{E_c(\tau)}{I_T})}$$

E_T/I_T一般介于 0.5×10^{-2}~2×10^{-2}

经验常数，K=200

测试之前程序中的错误数

指令长度

0到τ时刻，修复错误数

估算测试的进展

由上页方程的到：

$$E_c = E_T - \frac{I_T}{K \times MTTF}$$

这个公式可以估算：在改正了多少个错误后，软件即可达到所要求的MTTF，从而可以用来判断软件测试的充分性。

估算软件中的错误

- 植入法：

1. 植入错误：向程序中植入N个错误
2. 测试：发现M个错误，其中有P个错误是植入错误
3. 估算：软件原有错误的总数E_T

$$E_T = \frac{N}{P} \times (M - P)$$

假设：植入错误的分布和真实错误的分布一致！