Session 12
Performing Tests

chengbaolei@suda.edu.cn

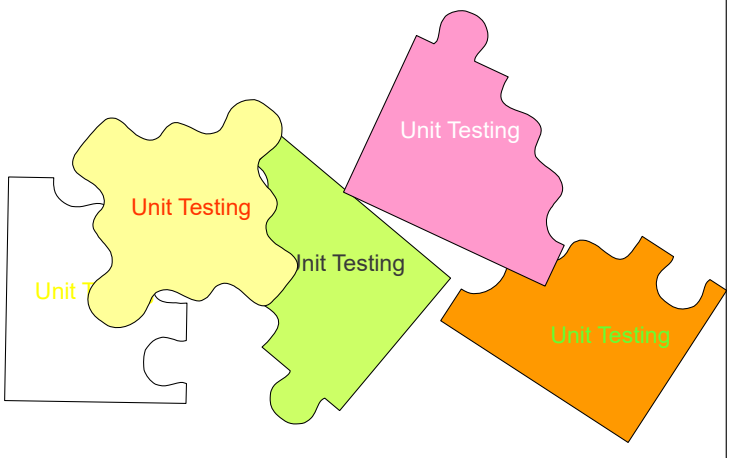
In this session, you will learn:

- Unit testing
- Integration testing
- System testing
- Acceptance testing
- Regression testing

- Most development teams take the incremental view of testing.
  - Unit testing
    - Verification of isolated software units
  - Integration testing
    - Verification of the interaction among software units
  - System testing
    - Verification of the behavior of a whole system
- Customers: acceptance testing

U-I-S-A

判断：
单元测试是指，对软件中的最小可测试单元，在与程序其他部分不隔离的情况下，进行检查和验证的工作。

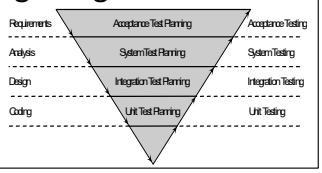


What is Unit Testing

- **Unit testing** is a software development process in which the smallest testable parts of an application, called units, are individually and independently checked for proper operation.
- A Unit may be
  - a program, a function, a procedure, a method, etc.

What is Unit Testing

- Purpose of unit testing：
  - Discover the errors introduced during coding process.
  - Validate whether code is consistent with the design.
  - Trace the implementation of requirements and design.
  - Discover the errors among design and requirement.



单元测试的目标和任务

- 目标: 单元模块被正确编码
- 信息能否正确地流入和流出单元
- 在单元工作过程中，其内部数据能否保持其完整性，包括内部数据的形式、内容及相互关系不发生错误，全局变量在单元中的处理和影响
- 为限制数据加工而设置的边界处，能否正确工作
- 单元的运行能否做到满足特定的逻辑覆盖

How to do Unit Testing

- Unit Testing
  - Static testing
    - It is primarily syntax checking of the code and/or manually reviewing the code or document to find errors.
    - This type of testing can be used by the developer who wrote the code, in isolation.
    - Peer reviews(同行评审), walkthroughs(走查) and inspections(审查) are also used.
  - Dynamic testing
    - Design and execute test cases
    - Tools: Junit, C++ Test, unittest(PyUnit), Pytest…

静态测试的常见方式不包括：
A. 同行评审
B. 走查
C. 审查
D. 执行测试用例

D

Peer reviews(同行评审)

- 一次检查少于200～400行代码
- 努力达到一个合适的检查速度：300～500LOC/hour
- 有足够的时间、以适当的速度、仔细地检查，但不宜超过60～90分钟
- 在复审前，代码作者应该对代码进行注释
- 使用检查表（checklist）能改进双方（作者和复审者）的结果
- 验证缺陷是否真正被修复

Best Practices for Peer Code Review（www.SmartBearSoftware.com）

Unit Testing Strategies

- Driver： is used to simulate superior module of tested module. It receives testing data, transmits related data to tested module, starts tested module and prints corresponding results.
- Stub： is used to simulate the calling modules in tested module. Generally, they only process few data.

```
1  //Program 1
2  #include <iostream.h>
3  void get_input(x& cost, int& y);
4  int main( )
5  {
6      double a;
7      int b;
8      char ans;
9      do
10     {
11         get_input(a, b);
12         cout.setf(ios::fixed);
13         cout.setf(ios::showpoint);
14         cout.precision(2);
15         cout << "a is " << a << endl;
16         cout << "b is " << b << endl;
17
18         cout << "Test again?"
19              << " (Type y for yes or n for no): ";
20         cin >> ans;
21         cout << endl;
22     } while (ans == 'y' || ans == 'Y');
23     return 0;
24 }
25 //Program 2
26 void function_under_test(int& x, int& y) {
27   ...
28   p = price(x);
29   ...
30 }
31 double price(int x) {return 10.00;}
```

Driver

Function under test

Stub

前置条件和后置条件

前置条件示例：
Contract.Requires(x != null);

后置条件示例：
Contract.EnsuresOnThrow<T>(this.F > 0);
Contract. Result<T>()

http://msdn.microsoft.com/zh-cn/library/dd264808.aspx

```
public string ReadAllText(string path)
{
      // Pre-conditions...
      if (path == null)
            throw new ArgumentNullException(...);
      if (path.Length == 0)
            throw new ArgumentException(...);
      if (IsOnlyWhitespace(path) ||
            ContainsInvalidCharacters(path))
            throw new ArgumentException(...);
      if (path.Length >
            GetMaximumFilePathLength())
            throw new PathTooLongException(...);

      // Open file and read and return contents
            as string...
      object contents = GetFileContents(path);

      // Post-conditions
      if (!contents is string)
            throw
                  new InvalidFileTypeException(...);
      if (string.IsNullOrEmpty(
            (string) contents))
            throw new EmptyFileException(...);
}
```

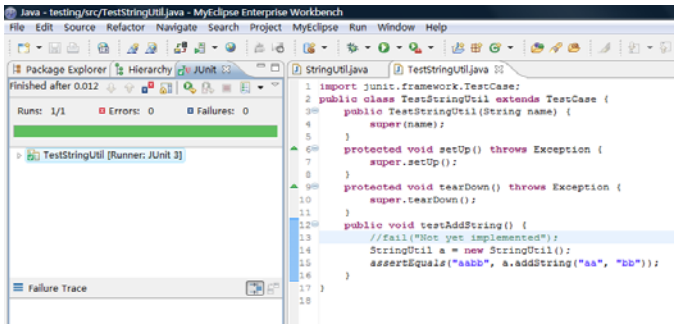
单元测试常用工具简介

- 代码规则/风格检查工具
- 内存资源泄漏检查工具
- 代码覆盖率检查工具
- 代码性能检查工具

单元测试工具列表

类别	工具
C 语言	C++ Test、CppUnit、QA C/C++、CodeWzard、Insure++6.0
Java 语言	Jtest、Junit、Jmock、EasyMock、MockRunner
JUnit 扩展框架	TestNG 、JWebUnit 和 HttpUnit
GUI （功能）	JFCUnit、Marathor
通用的	Rexelint、Splint、McCabe QA、CodeCheck、GateKeeper
.NET	.TEST、Nunit
Data Object，DAO	DDTUnit，DBUnit
EJB	MockEJB 或者 MockRunner
Servlet，Struts	Cactu，StrutsUnitTest
XML	XMLUnit
内嵌式系统等	Logiscope、JTestCase

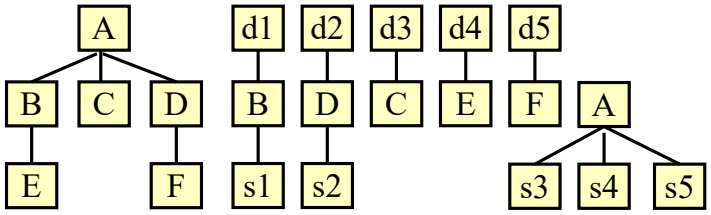
在Eclipse中JUnit应用举例

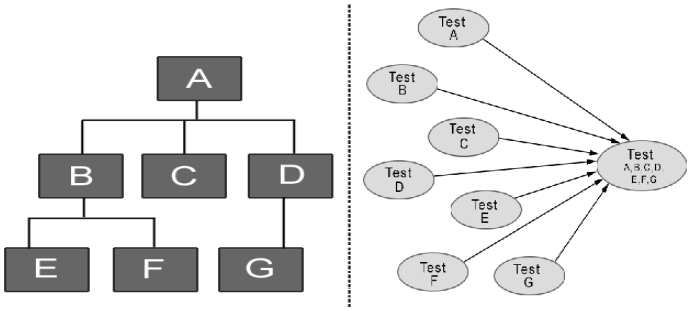


http://www.junit.org/apidocs/overview-summary.html

Integration Testing Strategy

- Big Bang integration
  - It is a non-incremental integration method, it integrates all system components together at one time, not considering the dependence of components and the possible risk.
- Strategy:





先对每一个子模块进行测试（单元测试阶段）

然后将所有模块一次性的全部集成起来进行集成测试 。

Integration Testing Strategy

- Advantage
  - Integration testing can be completed rapidly and only few stubs and drivers are needed.
  - Several testers can work in the parallel way, and human and material resources utilization is higher.
- Disadvantage
  - localization and debug more difficultly when error found.
  - Many interface errors can not be found until system testing.

Integration Testing Strategy

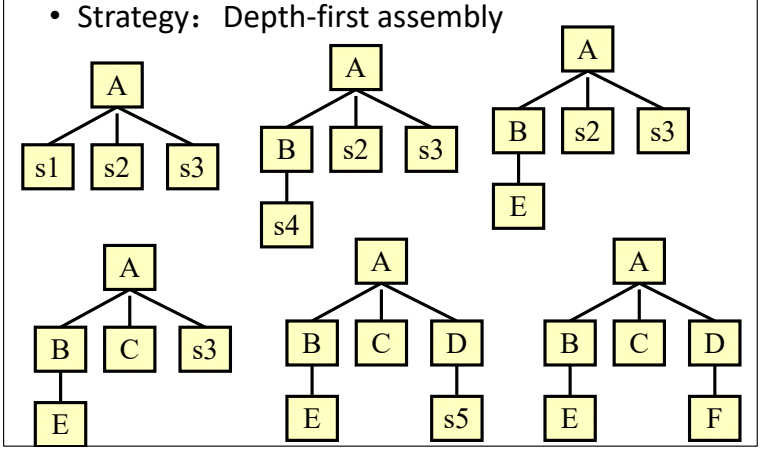
- Scope
  - Existing system with only minor modifications
  - Small systems with adequate unit testing
  - System made from certified high quality reusable components

Integration Testing Strategy

- Top-down integration
  - （1）Focus on the top level components firstly, then gradually test the bottom of components.
  - （2）Depth-first and breadth-first strategy can be used.
  - （3）Conduct regression testing, exclude possible errors caused by integrated.
  - （4） All modules are integrated into the system then finish testing, otherwise turn to （2）.

Integration Testing Strategy

- Strategy： Depth-first assembly



Integration Testing Strategy

集成测试策略				
类型	非渐增式集成	渐增式集成		
基本方法	先进行单元测试，再将所有模块一起进行集成测试。	把程序划分成小段来构造和测试		
		自顶向下	自底向上	三明治
特点	需要的用例少，比较简单，效率较高；但不能处理复杂的程序，而且不容易一次成功。	比较容易定位和改正错误，对接口可以进行更彻底测试。		

Integration Testing Strategy

渐增式集成			
名称	自顶向下集成	自底向上集成	三明治集成
方法	从主控模块开始，沿着程序控制层次向下移动，逐渐把各模块组合起来。（深度优先或广度优先）	从软件结构最底层的模块开始组装和测试，不需要桩模块。	混合增量式测试策略，综合了自顶向下和自底向上两种集成方法。
优点	可以在早期实现软件的一个完整功能。	可以并行集成，对被测模块可测性要求比自顶向下集成策略低。	桩模块和驱动模块的开发工作都比较小。
缺点	没有底层返回来真实数据流。	驱动模块开发量大，整体设计的错误发现较晚，集成到顶层时将变得越来越复杂。	增加了缺陷的定位难度，目标层在集成前测试不充分。

改进的三明治集成

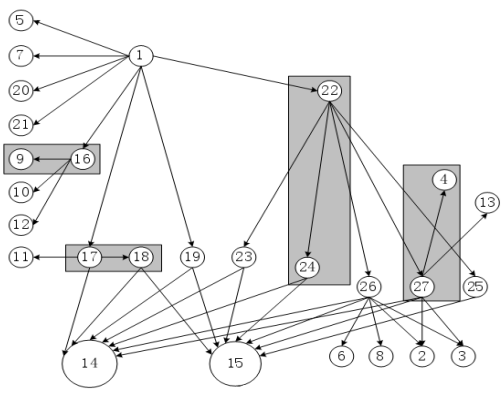
Integration Testing Strategy

- 基于分解的集成—更关注结点
  - Big bang（*大爆炸*）integration
  - Top-down *（自顶向下）*integration
  - Bottom-up *（自底向上）*integration
  - Sandwich *（三明治）*integration
- 基于调用图的集成—更关注边
  - 成对集成
  - 相邻集成

基于调用图的集成

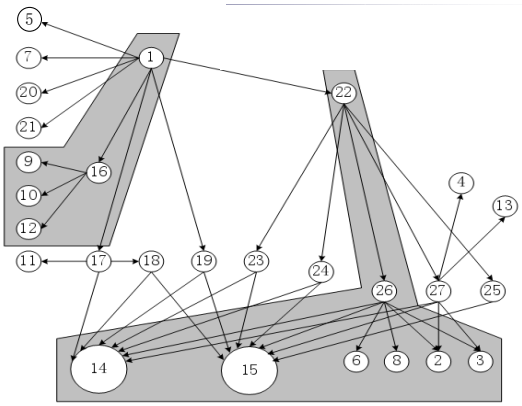
- 成对集成
  - 免除桩/驱动模块的开发工作，使用实际的代码
  - 为避免大爆炸集成，限制在调用图的一对单元上
  - 对调用图中的每条边有一个集成测试过程
- 相邻集成
  - 借用拓扑学中的邻接概念
  - 在有向图中，结点邻居包括所有直接前驱结点和所有直接后继结点
  - 对应结点的桩和驱动模块集合

成对集成示例



40次集成测试会话

相邻集成示例



高频集成

- 使用高频集成测试需要具备一定的条件:
  - 可以持续获得一个稳定的增量，并且该增量内部已被验证没有问题;
  - 大部分有意义的功能增加可以在一个相对稳定的时间间隔（如每个工作日）内获得;
  - 测试包和代码的开发工作必须是并行进行的，并且需要版本控制工具来保证始终维护的是测试脚本和代码的最新版本;
  - 必须借助于使用自动化工具来完成。

高频集成

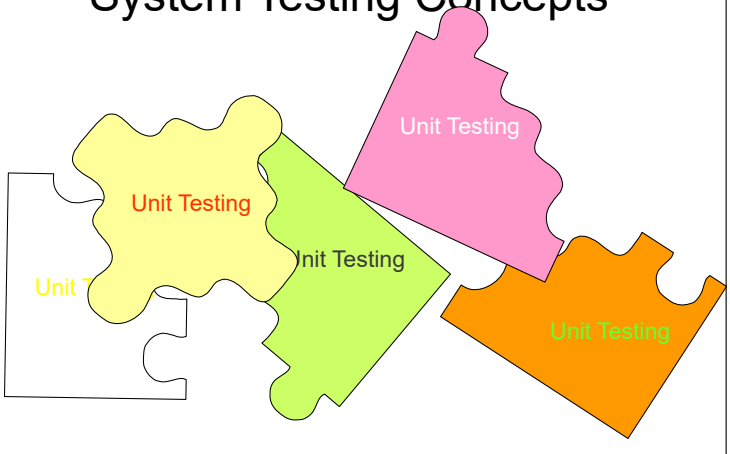
- 优点：能在开发过程中及时发现代码错误，能直观地看到开发团队的有效工程进度。
- 缺点：需要开发和维护源代码和测试包。

Principle of Integration Testing

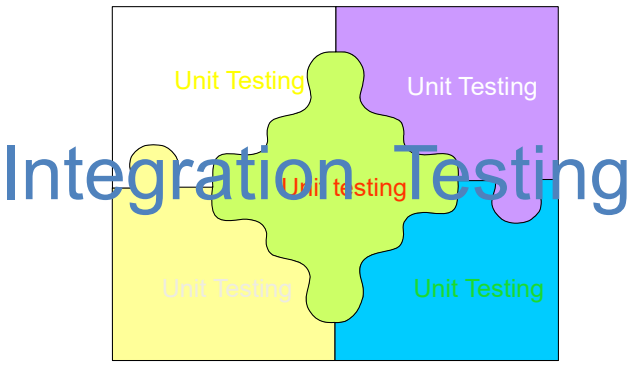
- Key modules must be tested sufficiently. 20-80
- All interfaces must be tested.
- When the interface is changed, all related interfaces should be tested by using regression testing.
- Integration testing is conducted according to plan and prevents random testing.
- Integration testing strategy should integrate the relationship among quality, cost and progress.

- Unit testing
- Integration testing
- System testing
- Acceptance testing
- Regression testing

System Testing Concepts



System Testing Concepts



Acceptance Testing

- Acceptance testing is a formal testing conducted to determine whether the software satisfies the acceptance criteria defined by the customer in the requirements phase of the SDLC.
- The end user or customer can conduct acceptance testing to validate whether or not to accept the product.
- 交付测试

Acceptance Testing

- There are two types of acceptance testing:
  - Alpha testing:
    - Alpha testing is a simulated or actual operational testing at an in-house site close to the development team.
    - This type of testing helps evaluate the software to ascertain whether or not it meets all the requirements specified in the requirements analysis phase.
  - Beta testing:
    - Beta testing involves operational testing of the software at a site away from the developers or at the customer site.

Acceptance Testing

- α测试：早期的、不稳定的软件版本所进行的验收测试，受控的实验室测试
- β测试：晚期的、更加稳定的软件版本所进行的验收测试，不受控的非实验室测试

Acceptance Testing

- β 测试的局限性：
  - β 测试通常不是专业测试人员，问题往往停留在易用性上；
  - 环境不可控，使用不当引起的问题居多；
  - 为了评价软件或获得软件而参与测试；
  - 反馈信息简单，经常无法重现。
- 众包测试/群体智能的兴起

测试步骤 (Testing life cycle)

- 制定测试计划及验收通过准则，通过客户评审
- 设计测试用例并通过评审
- 准备测试环境与数据，执行测试用例，记录测试结果
- 分析测试结果，根据验收通过准则分析测试结果，作出验收是否通过及测试评价。
  - 测试项目通过；
  - 测试项目没有通过，但存在变通方法，在维护后期或下一个版本改进；
  - 测试项目没有通过，并且不存在变通方法，需要很大的修改；
  - 测试项目无法评估或者无法给出完整的评估。此时须给出原因
- 提交测试报告

- Compare unit testing vs. integration testing vs. system testing vs. acceptance testing:
  - Level
  - Action
  - Actor
  - Method

5W+1H