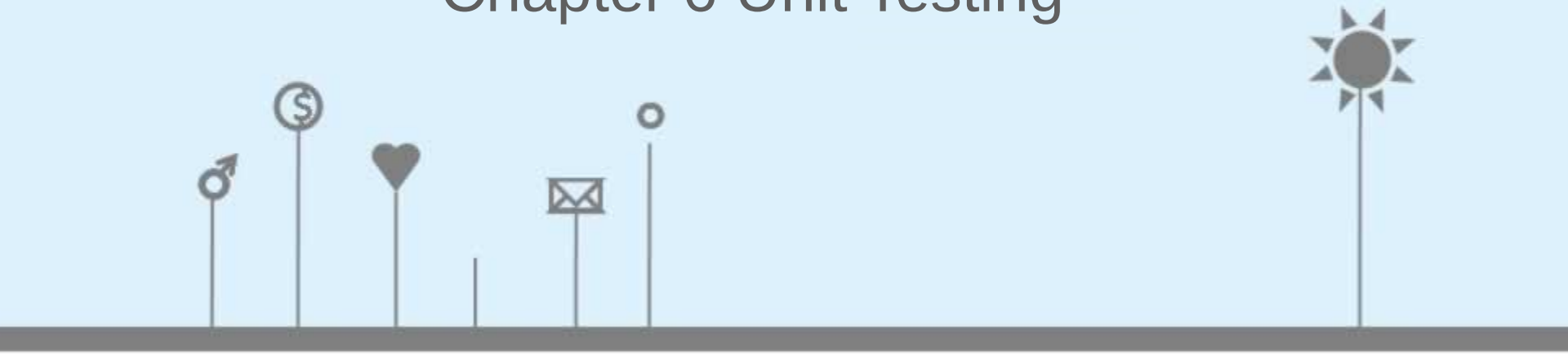
Software College Northeastern University

Software Quality Assurance and Testing

Chapter 6 Unit Testing



wuchenni@qq.com

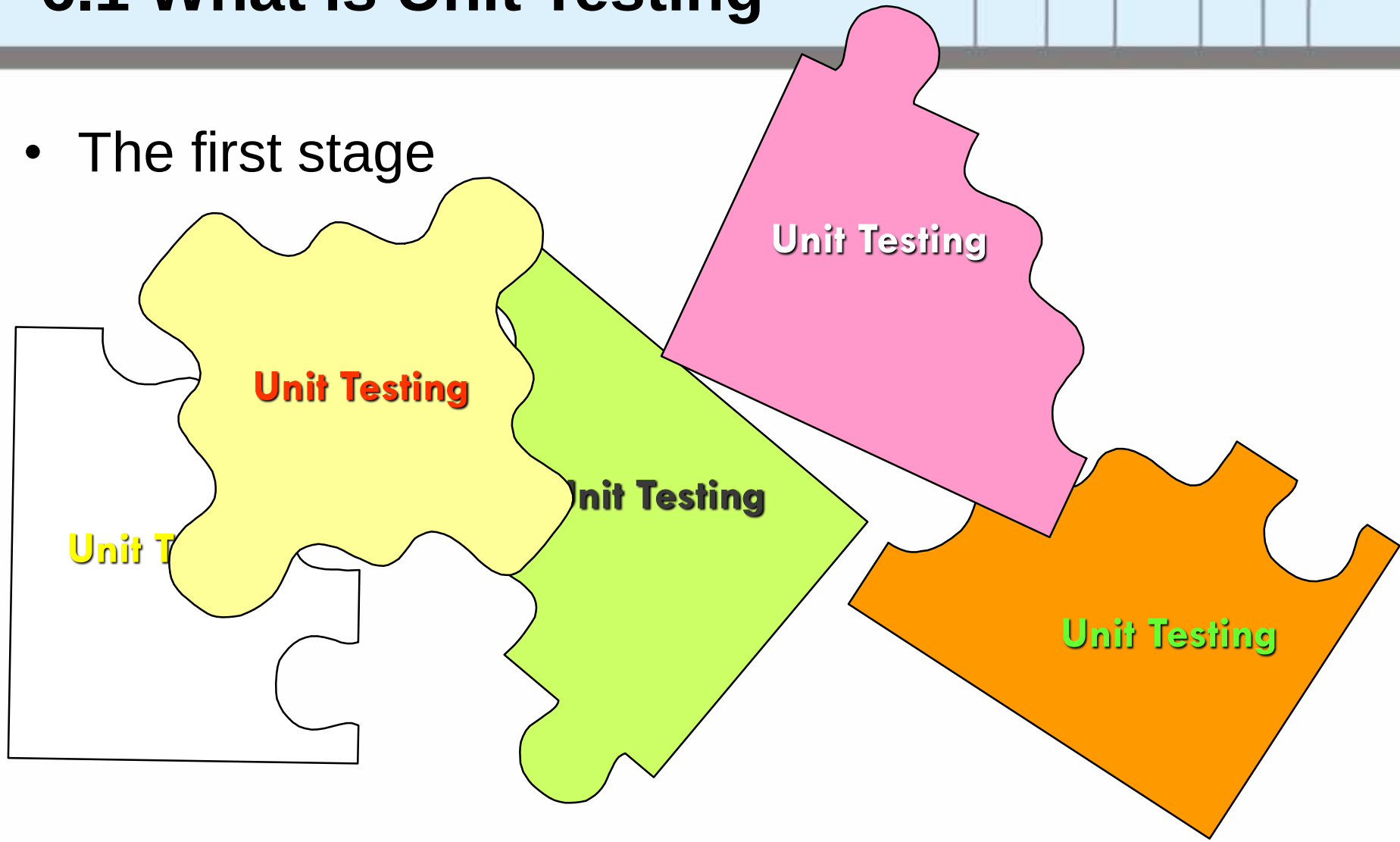
Contents

Chapter 6

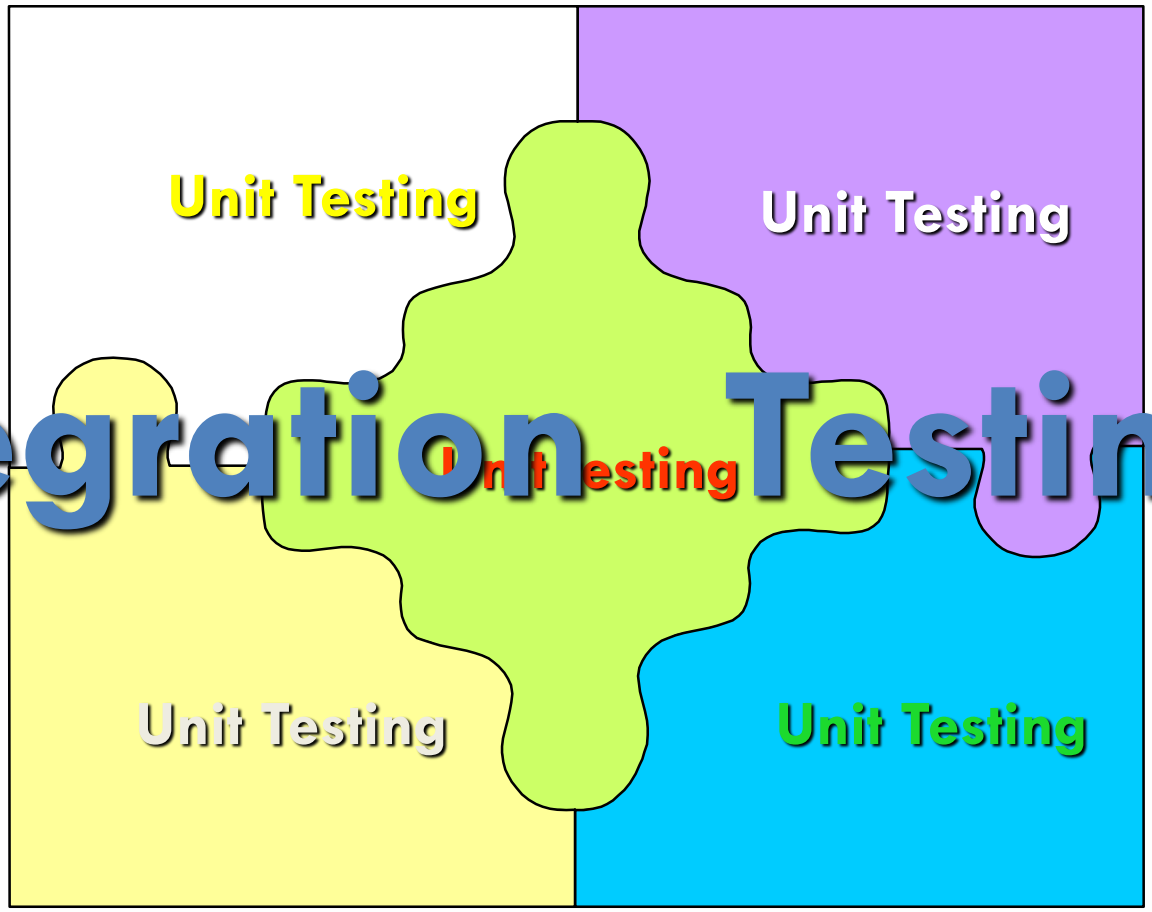
Unit Testing

6.1 What is Unit Testing

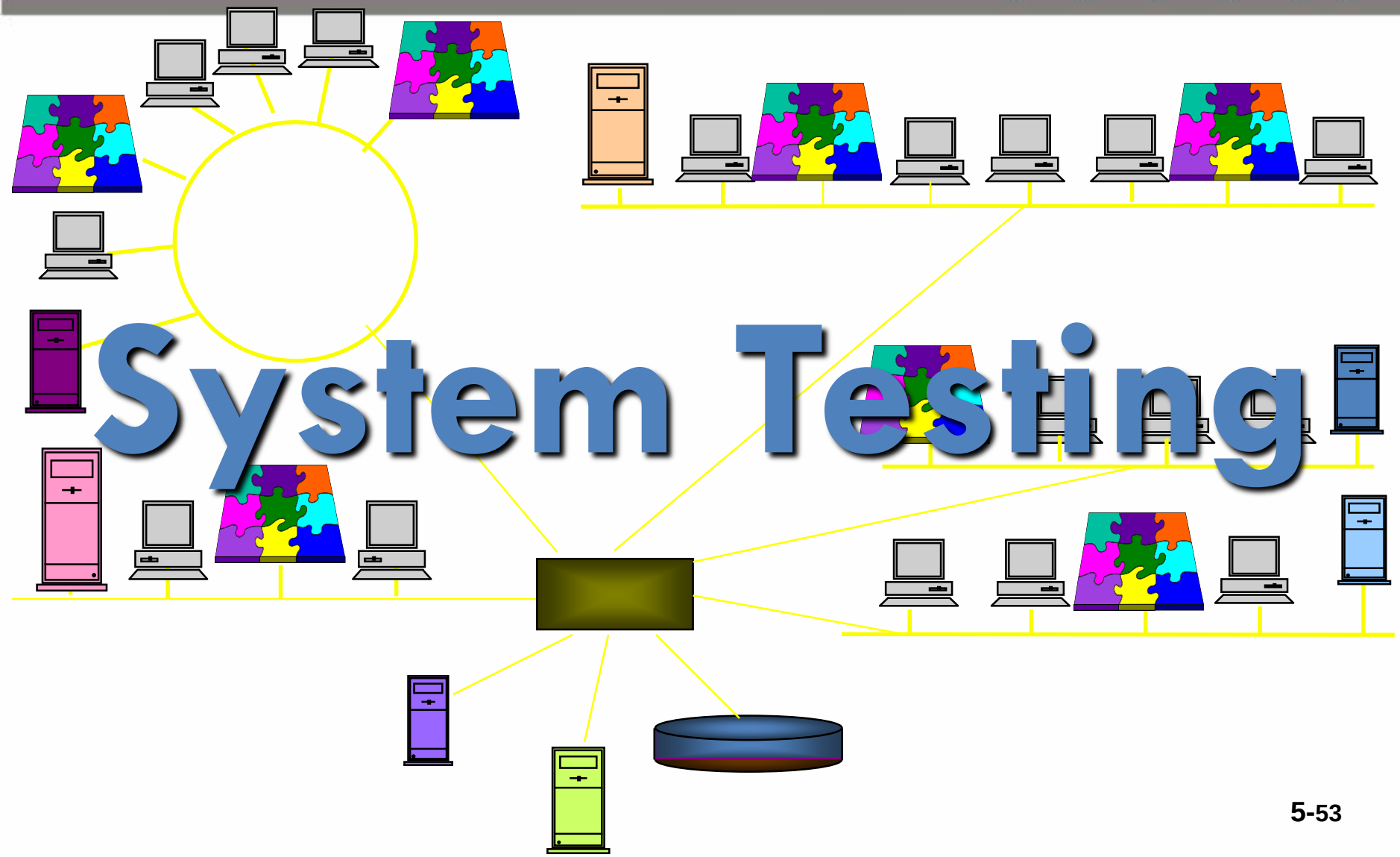
- The first stage



6.1 What is Unit Testing



6.1 What is Unit Testing



6.1 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 scrutinized（审查） for proper operation.
  - A Unit is the smallest unit of software design
  - Software unit must be apart from other parts of the program and test independently.
  - A Unit a specific function or a class method and also a module or a small set of modules
    - In Java, a unit is a class or a class method.
    - In C, a unit is a function or sub processes.

6.1 What is Unit Testing

6.1 What is Unit Testing

- Basic attributes of unit testing：
  - –Clear function
  - –Specifications definition
  - –Clear Interface definition

6.1 What is Unit Testing

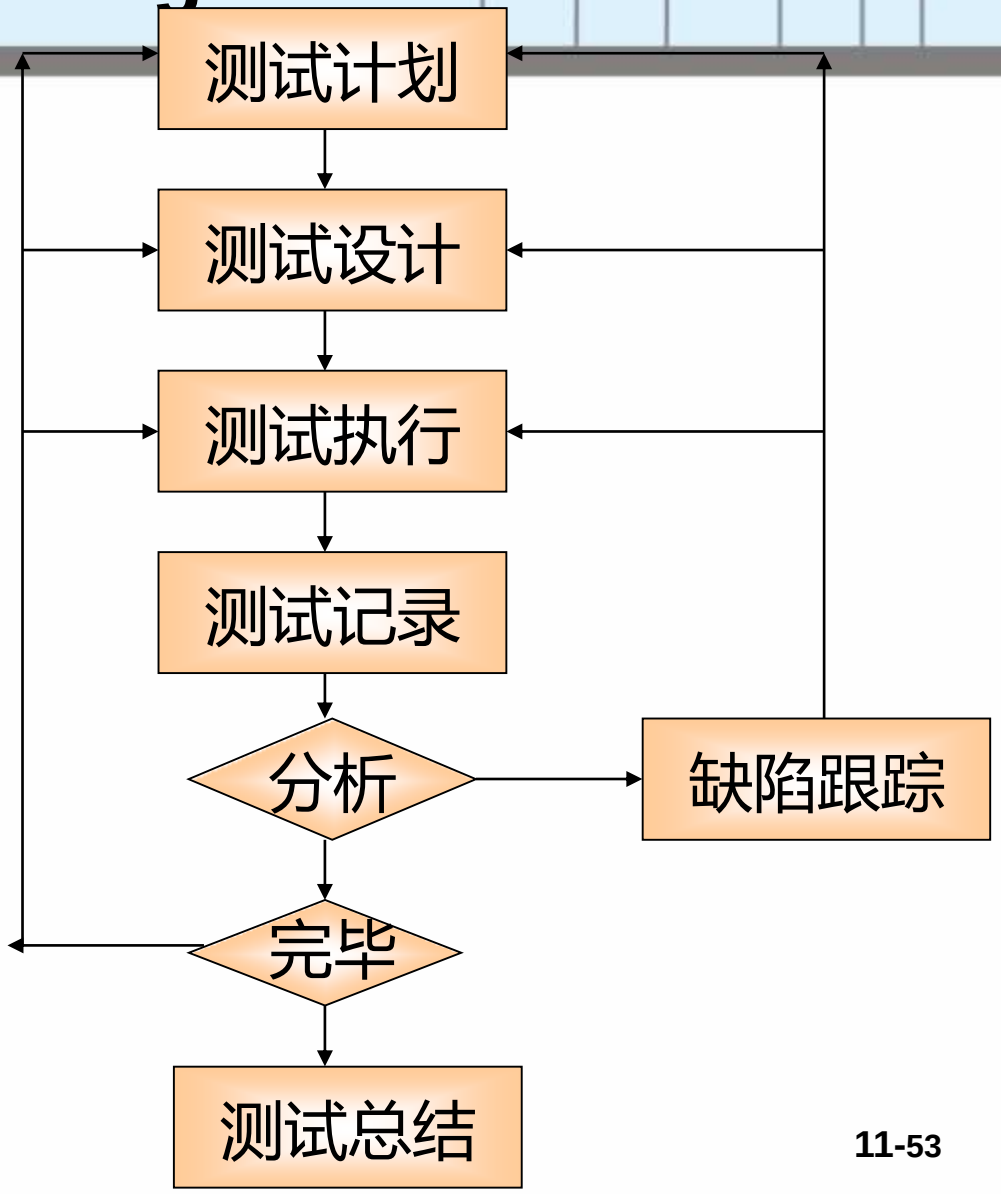
- Purpose of unit testing：
  - Validate whether code is consistent with the design.
  - Trace the implementation of requirements and design.
  - Discovery the errors among design and requirement.
  - Discovery the errors introduced during coding process.

6.1 What is Unit Testing

- Attention:
  - Target: ensure that module was properly coding
  - Basis: detailed specification
  - Process: after design, script development, implementation, debugging and analysis results a process
  - Executives: by program developers and tester to complete together.
  - Test methods: mainly used white-box testing, auxiliary black box testing methods
  - How to evaluate: through all the unit test cases without serious defects of code.

6.1 What is Unit Testing

- 单元测试的过程
  - 1、在详细设计阶段完成单元测试计划
  - 2、建立单元测试环境，完成测试设计和开发
  - 3、执行单元测试用例，并且详细记录测试结果
  - 4、判定单元测试是否通过
  - 5、提交单元测试报告



6.1 What is Unit Testing

- Benefits (Why do we need unit testing?)
  - The goal of unit testing is to isolate each part of the program and show that the individual parts are correct.
  - A unit test provides a strict, written contract that the piece of code must satisfy.
  - Unit tests find problems early in the development cycle.

6.1 What is 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.
  - Dynamic testing

6.1 What is Unit Testing

- Unit Testing
  - Static testing
    - Coding standards and specifications
      - C++ Code standards of writing
    - Code reviews, inspections and walkthroughs are also used.
      - Walkthroughs
      - Inspections

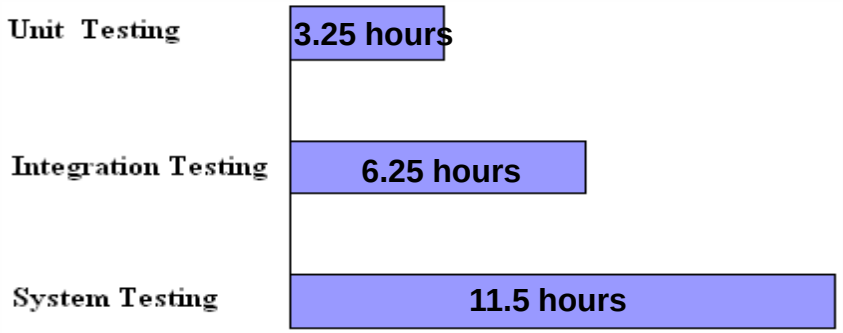
6.1 What is Unit Testing

- 走查审查比对表

项目	走查	审查
准备	通读设计和编码	应准备好需求描述文档、程序设计文档、程序源代码清单、代码编码标准和代码缺陷检查表
形式	非正式会议	正式会议
参加人员	开发人员为主	项目组成员包括测试人员
主要技术方法	无	缺陷检查表
注意事项	限时、不要现场修改代码	限时、不要现场修改代码
生成文档	会议记录	静态分析错误报告
目标	代码标准规范、无逻辑错误	代码标准规范、无逻辑错误

6.1 What is Unit Testing

- **Misunderstanding about unit testing**
  - Unit testing wastes so much time.
  - Unit testing only prove what has been done by codes.
  - Because I'm a great programmer , I do not need conduct unit testing.
  - In any case，integration testing will seize all bugs.
  - Low cost efficiency.

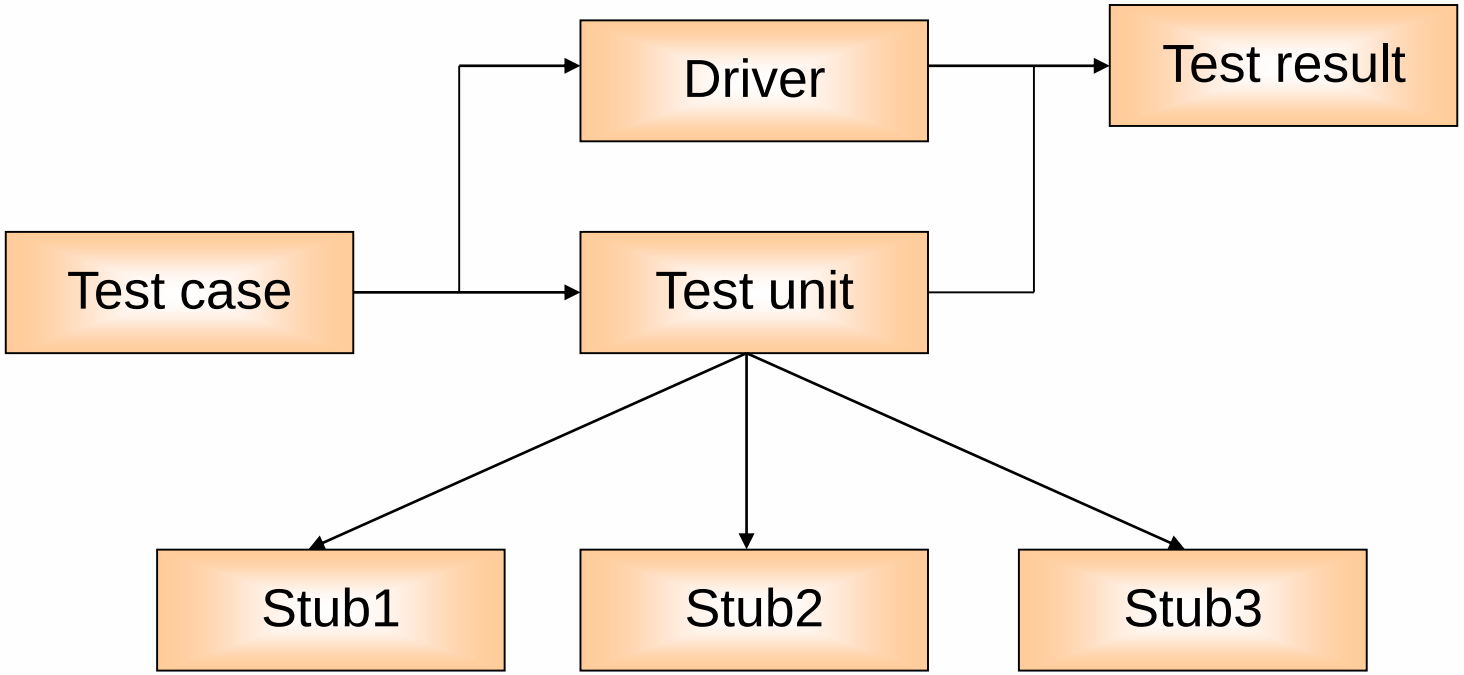


6.2 Unit Testing Strategies

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

6.2 Unit Testing Strategies

- Why do we need stub modules and driver modules?



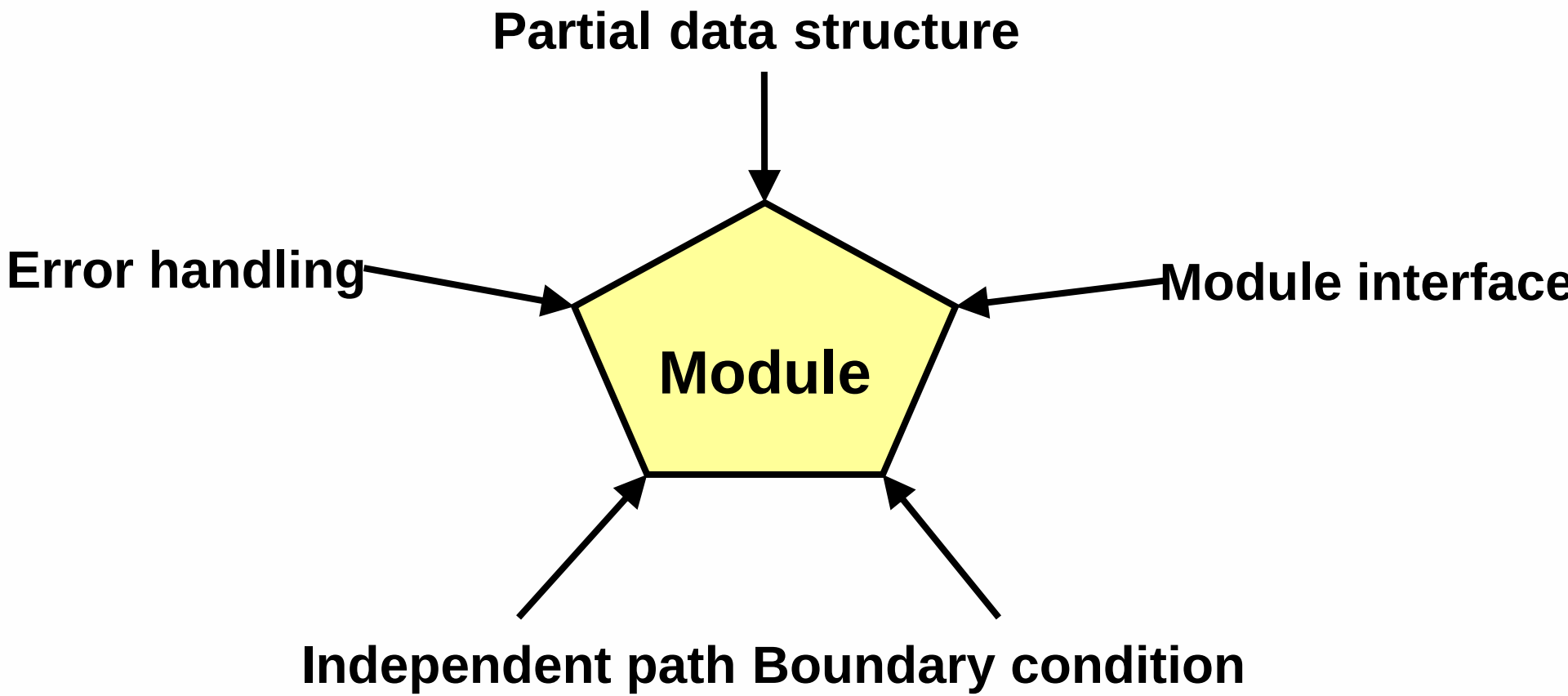
Unit test environment

6.2 Unit Testing Strategies

- Top-down unit testing strategy
  - Test the top level firstly, take the unit called by top level as stub，and then test the second level, take the unit that has been tested as driver module sequentially .
- Bottom-up unit testing strategy
  - Test the bottom module firstly, set up driver module for it, and then test the upper module, take the module has been tested as stub sequentially.
- Isolation testing
  - Not consider the relationship between modules, design sub and driver module for every module.

6.3 Unit Testing Analysis

- **Considered aspect of unit testing:**



6.4 Unit Testing Guidelines

- 实施单元测试的时候, 如果没有一份经过实践证明的详细规范, 很难掌握测试的 “度”, 范围太小施展不开, 太大又侵犯 “别人的” 地盘. 上帝的归上帝, 凯撒的归凯撒, 给单元测试念念紧箍咒不见得是件坏事, 反而更有利于发挥单元测试的威力, 为代码重构和提高代码质量提供动力。
- 这份文档来自 Geotechnical, 是一份非常难得的经验准则. 你完全可以以这份准则作为模板, 结合所在团队的经验, 整理出一份内部单元测试准则。

6.4 Unit Testing Guidelines

- 1. Keep unit tests small and fast
- 2. Unit tests should be fully automated and non-interactive
- 3. Make unit tests simple to run
- 4. Measure the tests
- 5. Fix failing tests immediately
- 6. Keep testing at unit level
- 7. Start off simple

6.4 Unit Testing Guidelines

- 8. Keep tests independent
- 9. Keep tests close to the class being tested
- 10. Name tests properly
- 11. Test public API
- 12. Think black-box
- 13. Think white-box
- 14. Test the trivial cases too

6.4 Unit Testing Guidelines

- 15. Focus on execution coverage first
- 16. Cover boundary cases
- 17. Provide a random generator
- 18. Test each feature once
- 19. Use explicit asserts
- 20. Provide negative tests
- 21. Design code with testing in mind

6.4 Unit Testing Guidelines

- 22. Don't connect to predefined external resources
- 23. Know the cost of testing
- 24. Prioritize testing
- 25. Prepare test code for failures
- 26. Write tests to reproduce bugs
- 27. Know the limitations

6.5 How to Use JUnit

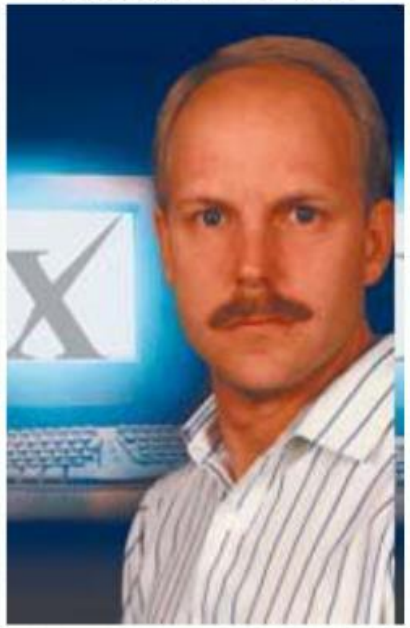
- Tools of Unit test
  - Unit tests generally use the method of white-box testing.
  - Unit test tools according to the different language has its corresponding version.
    - The typical tools - xUnit family:
      - JUnit —— Java
      - CppUnit —— C++
      - NUnit —— C# (. Net)
      - ……

6.5 How to Use JUnit

Junit的诞生

Erich Gamma

Kent Beck



junit

JUnit.org

- 设计模式开创者之一
- Eclipse Java 总设计师

- 软件开发方法学大师
- 极限编程创始人

6.5 How to Use JUnit

- JUnit is a unit testing framework for the Java programming language.
- JUnit has been important in the development of test-driven development, and comprises a family of unit testing frameworks collectively known as xUnit that originated with SUnit.
- JUnit is linked as a JAR at compile-time; the framework resides under packages *junit.framework* for JUnit 3.8 and earlier and under *org.junit* for JUnit 4 and later.

6.5 How to Use JUnit

- JUnit features include:
  - Assertions for testing expected results
  - Test fixtures（测试装置） for sharing common test data
  - Test suites for easily organizing and running tests
  - Graphical and textual test runners

6.5 How to Use JUnit

- Test Fixtures
  - A test fixture is the set of preconditions or state needed for a test to run. It's known as a test context.
- Test Suites
  - A test suite is a set of tests that all share the same fixture.

6.5 How to Use JUnit

- Test Execution
  - The execution of an individual unit test proceeds as follows:

```
setup();
...
/* Body of test */
...
teardown();
```

6.5 How to Use JUnit

- Assertions
  - An assertion is a function or macro that verifies the behavior of the unit under test. Failure of an assertion typically throws an exception, aborting the execution of the current test.

6.5 How to Use JUnit

- Assertion methods

Method	Description
assertEquals(a,b)	Test if a is equal to b
assertFalse(a)	Test if a is false
assertNotSame(a, b)	Test if a and b do not refer to the identical object
assertNull(a)	Test if a is null
assertSame(a,b)	Test if a and b refer to the identical object
assertTrue(a)	Test if a is true

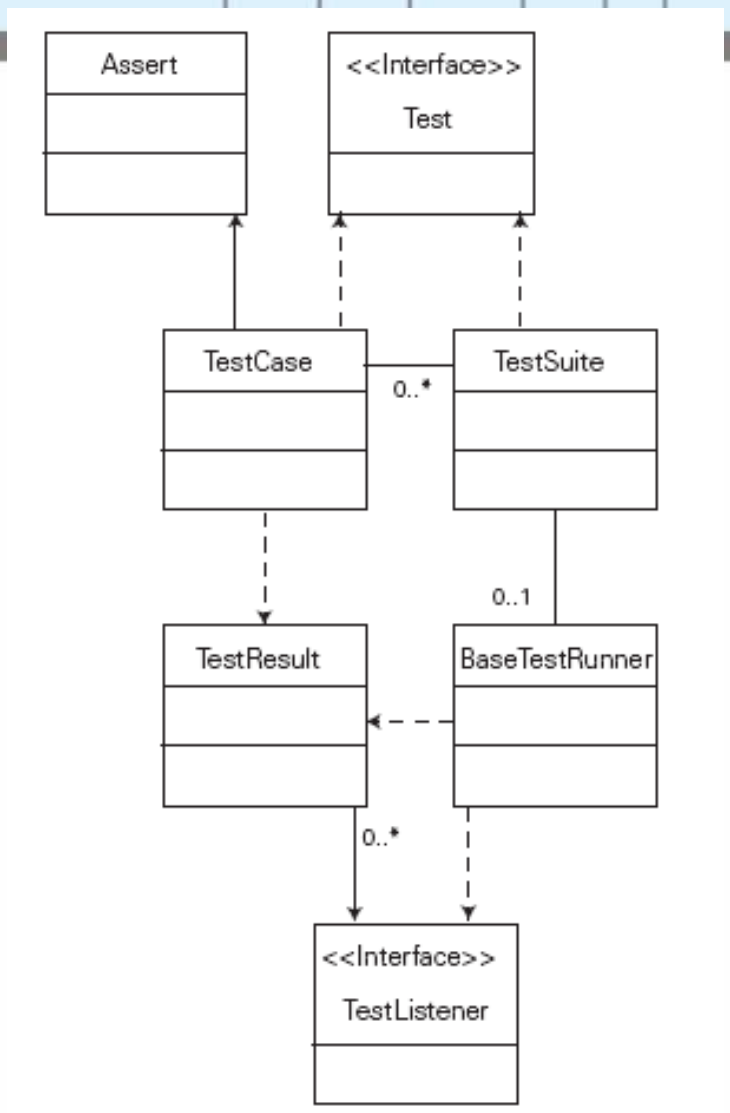
Class/Interface	Responsibilities
Assert	An assert method is silent when its proposition succeeds but throws an exception if the proposition fails
TestResult	A TestResult collects any errors or failures that occur during the test
Test	A Test can be run and passed a TestResult
TestListener	A TestListener is apprised of events that occur during a test, including when the test begins and ends, along with any errors or failures
TestCase	A TestCase defines an environment (or fixture) that can be used to run multiple tests
TestSuite	A TestSuite runs a collection of test cases, which may include other test suites, it's a composite of Tests
BaseTestRunner	A test runner is a user interface for launching tests. The test runners are designed to execute your tests and provide you with statistics regarding the outcome

6.5 How to Use JUnit

- Criteria for test methods
  - The method must be declared public.
  - The method must return nothing (void).
  - The name of the method must start with the word *test*.
  - The method can't take any arguments.

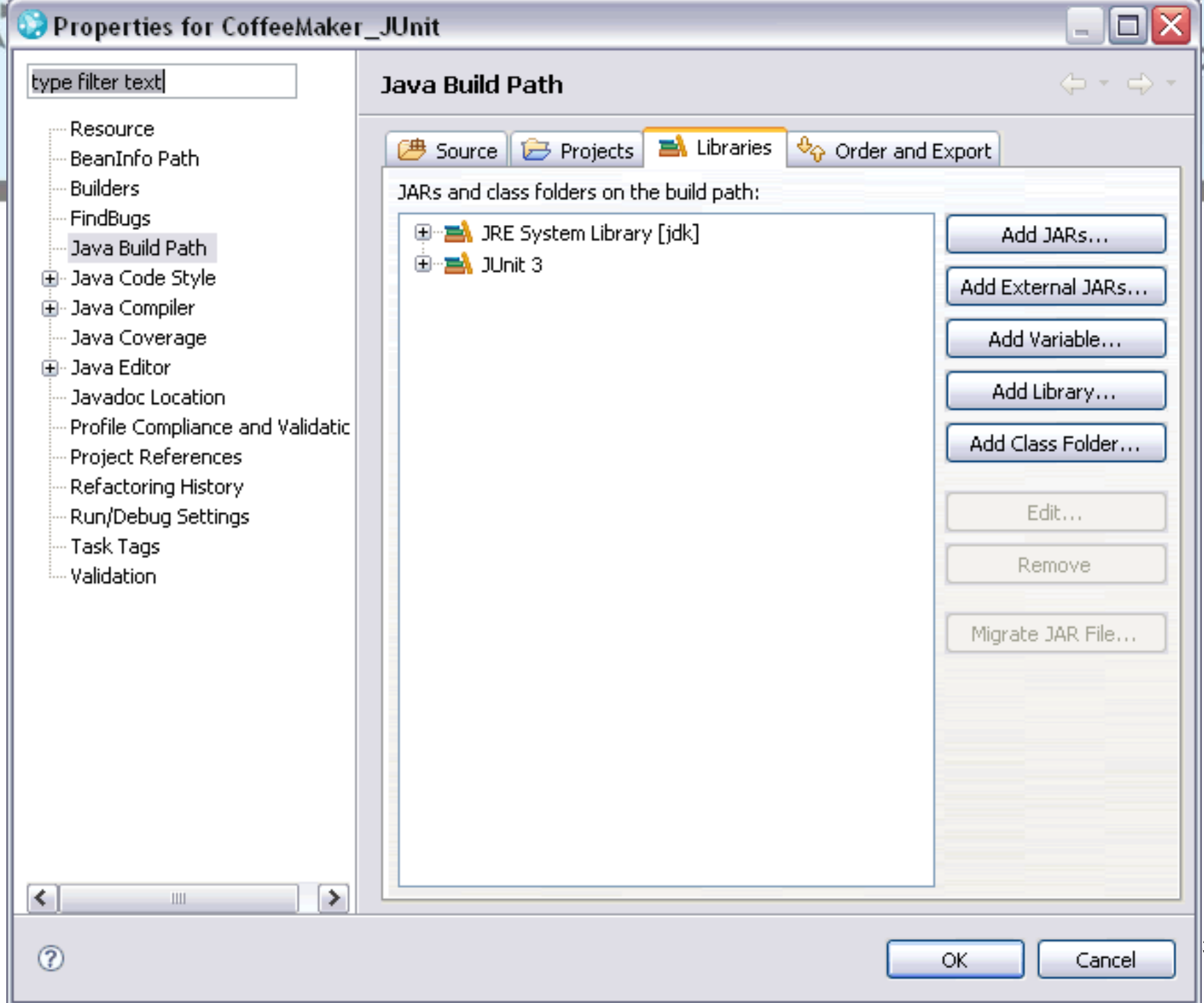
6.5 How to Use JUnit

- How JUnit run a test
  - call TestRunner first
  - then create a TestSuite object
  - load TestResult and TestListener
  - individual TestCase will run
  - Assert will throw exception if test goes wrong
  - Interface Test will pass the Test



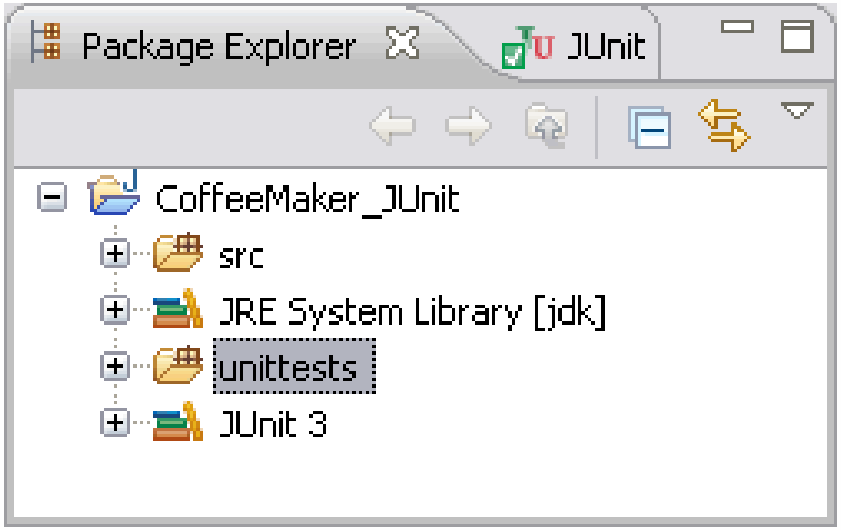
6.5 How to Use JUnit

- JUnit installation
  - Check JUnit version running in Eclipse
    - Right click on the project and select ***Properties***. In the left column select ***Java Build Path*** then check the details of ***Libraries***
  - Import another JUnit version into Eclipse
    - Open Eclipse, select ***Project*** and select ***Properties***. In the left column select ***Java Build Path***. To the right select the ***Libraries*** tab. Click the ***Add External JARs...*** button. You should start off in the Eclipse install directory, otherwise search through the file system until you find the Eclipse install directory.


Properties for CoffeeMaker_JUnit
type filter text
Resource
BeanInfo Path
Builders
FindBugs
Java Build Path
Java Code Style
Java Compiler
Java Coverage
Java Editor
Javadoc Location
Profile Compliance and Validatic
Project References
Refactoring History
Run/Debug Settings
Task Tags
Validation
Java Build Path
Source
Projects
Libraries
Order and Export
JARs and class folders on the build path:
JRE System Library [jdk]
JUnit 3
Add JARs...
Add External JARs...
Add Variable...
Add Library...
Add Class Folder...
Edit...
Remove
Migrate JAR File...
OK
Cancel

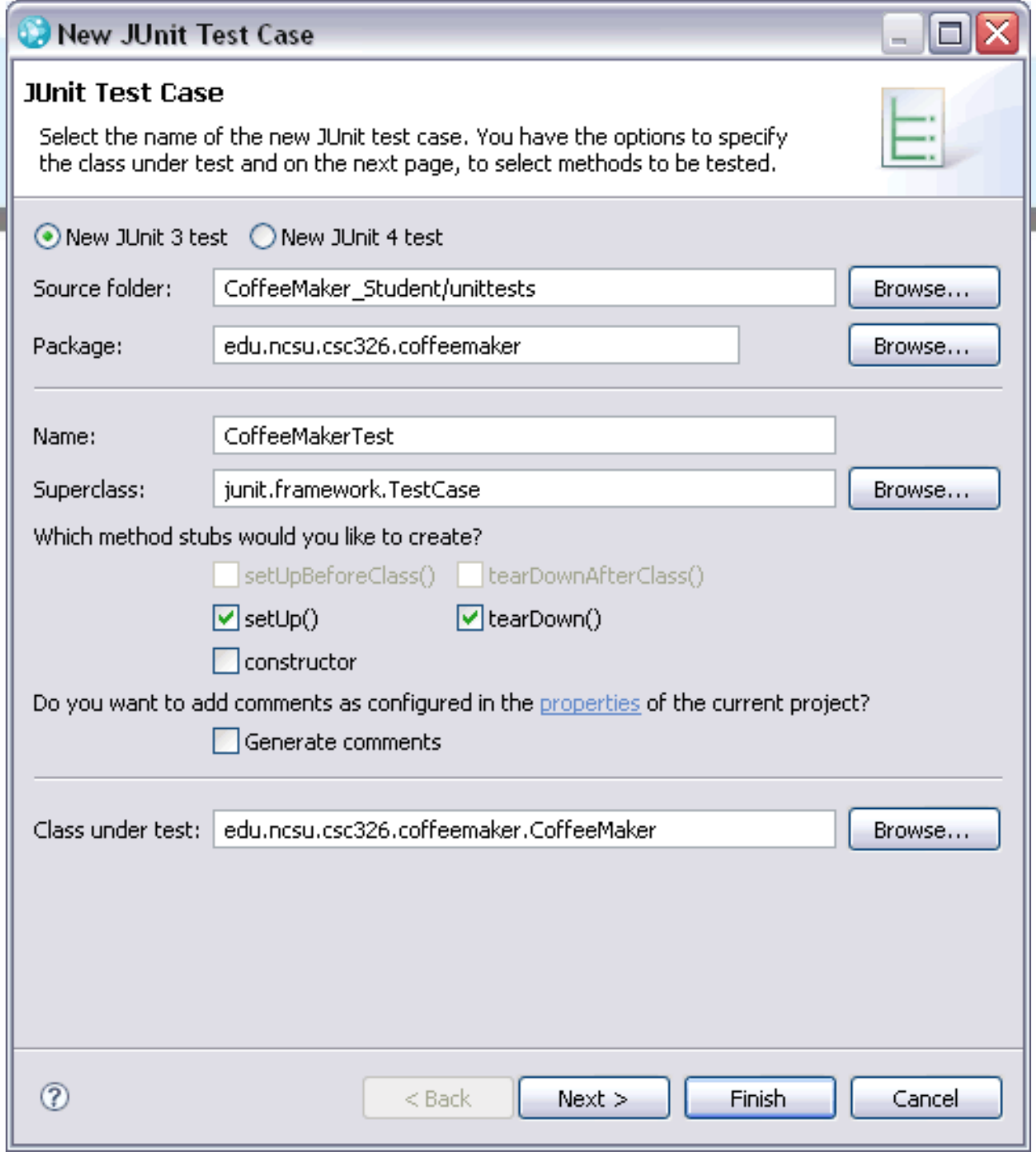
6.5 How to Use JUnit

- Steps for JUnit testing
  - Step 1: Create a Java Project
  - Step 2: Creating a Test Class
  - Step 3: Creating a Test Suite
  - Step 4: Running JUnit Test Cases



6.5 How to Use JUnit

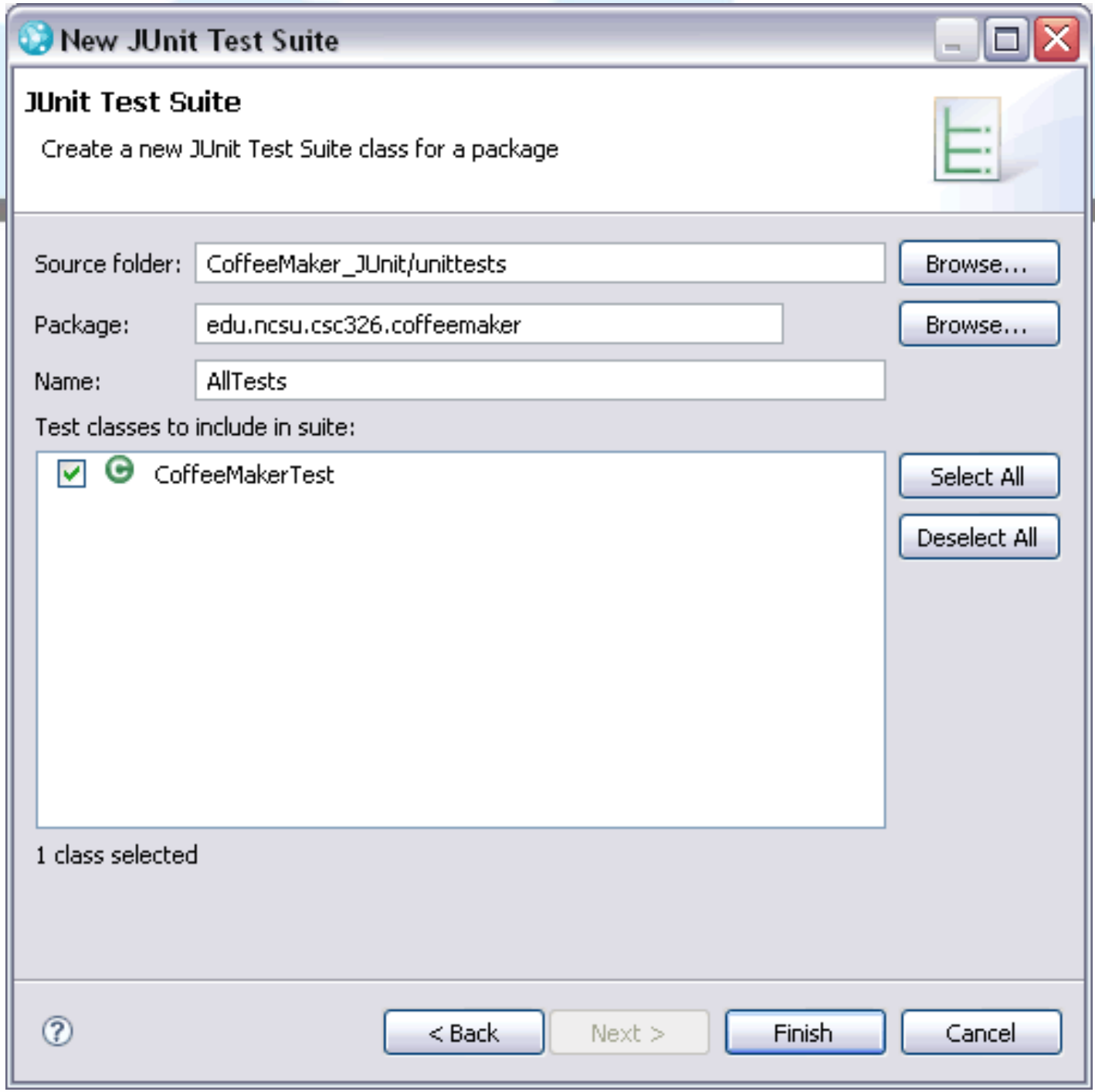
- Step 2: Creating a Test Class
  - Select **File > New > JUnit Test Case**
  - Select the arrow of the  button in the upper left of the toolbar. Select **JUnit Test Case** ,
  - Right click on a package in the Package Explorer view in the Java Perspective, and select **JUnit Test Case**, or
  - Click on the arrow of the  icon in the toolbar. Select **JUnit Test Case** .
  - You can create a normal Java class as shown in the Eclipse tutorial, but include *junit.framework.TestCase* as the super class of the test class you are creating.
  - Right click on an application class that you want to create a test class for, and select **New > JUnit Test Case**.


New JUnit Test Case
JUnit Test Case
Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.
New JUnit 3 test
New JUnit 4 test
Source folder:
CoffeeMaker_Student/unittests
Browse...
Package:
edu.ncsu.csc326.coffeemaker
Browse...
Name:
CoffeeMakerTest
Superclass:
junit.framework.TestCase
Browse...
Which method stubs would you like to create?
setUpBeforeClass()
tearDownAfterClass()
setUp()
tearDown()
constructor
Do you want to add comments as configured in the properties of the current project?
Generate comments
Class under test:
edu.ncsu.csc326.coffeemaker.CoffeeMaker
Browse...
< Back
Next >
Finish
Cancel

```
import junit.framework.TestCase;
public class SampleTest extends TestCase {
     private java.util.List emptyList;
     /**
      * Sets up the test fixture.
      * (Called before every test case method.)
      */
     protected void setUp() {
          emptyList = new java.util.ArrayList();
     }
     /**
      * Tears down the test fixture.
      * (Called after every test case method.)
      */
     protected void tearDown() {
          emptyList = null;
     }
     public void testSomeBehavior() {
          assertEquals("Empty list should have 0 elements", 0, emptyList.size());
     }

     public void testForException() {
          try {
               Object o = emptyList.get(0);
               fail("Should raise an IndexOutOfBoundsException");
          }
          catch (IndexOutOfBoundsException success) {
          }
     }

}
```

6.5 How to Use JUnit

- Step 3:Creating a Test Suite
  - First, select the directory (usually *unittests/*) that you wish to create the test suite class in.
  - Select **File > New > Other... > Java > JUnit > JUnit Test Suite.**
  - Select the arrow of the  button in the upper left of the toolbar. Select **Other... > Java > JUnit > JUnit Test Suite**,
  - Right click on a package in the Package Explorer view in the Java Perspective, and select **New > Other... > Java > JUnit > JUnit Test Suite**, or
  - You can create a normal Java class as shown in the Eclipse tutorial, but include *junit.framework.TestSuite* as the super class of the test class you are creating.


New JUnit Test Suite
JUnit Test Suite
Create a new JUnit Test Suite class for a package
Source folder:
CoffeeMaker_JUnit/unittests
Browse...
Package:
edu.ncsu.csc326.coffeemaker
Browse...
Name:
AllTests
Test classes to include in suite:
CoffeeMakerTest
Select All
Deselect All
1 class selected
< Back
Next >
Finish
Cancel

6.5 How to Use JUnit

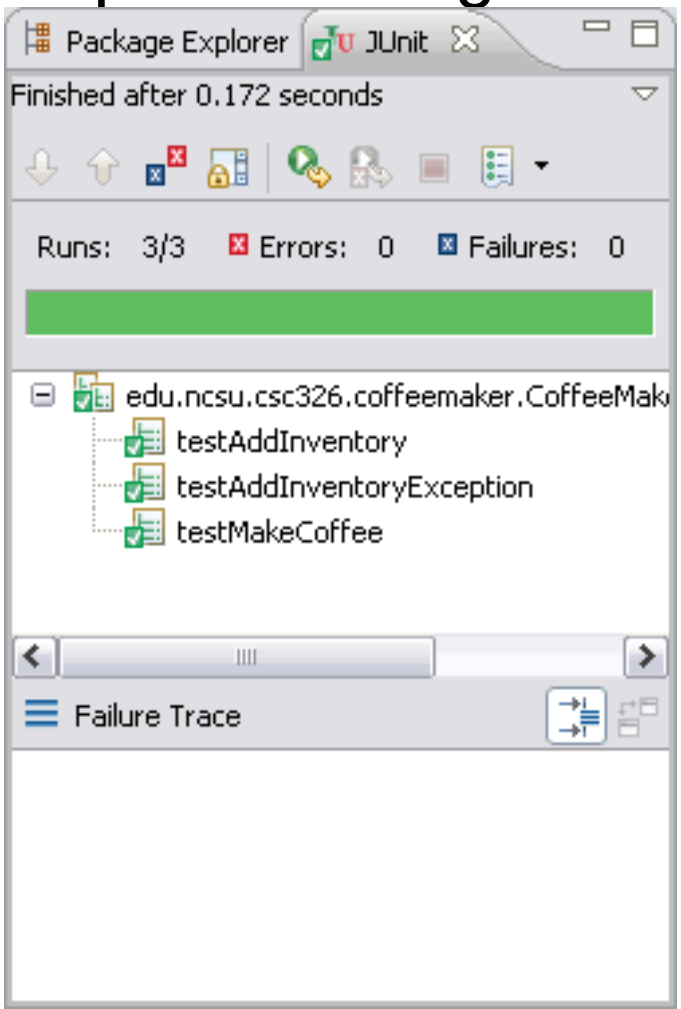
```
import junit.framework.Test;
import junit.framework.TestSuite;


public class SampleTestSuite {


     public static Test suite() {
          TestSuite suite = new TestSuite("Sample Tests");

          // Add one entry for each test class
          // or test suite.
          suite.addTestSuite(SampleTest.class);

          // For a master test suite, use this pattern.
          // (Note that here, it's recursive!)
          suite.addTest(AnotherTestSuite.suite());

          return suite;
     }

}
```

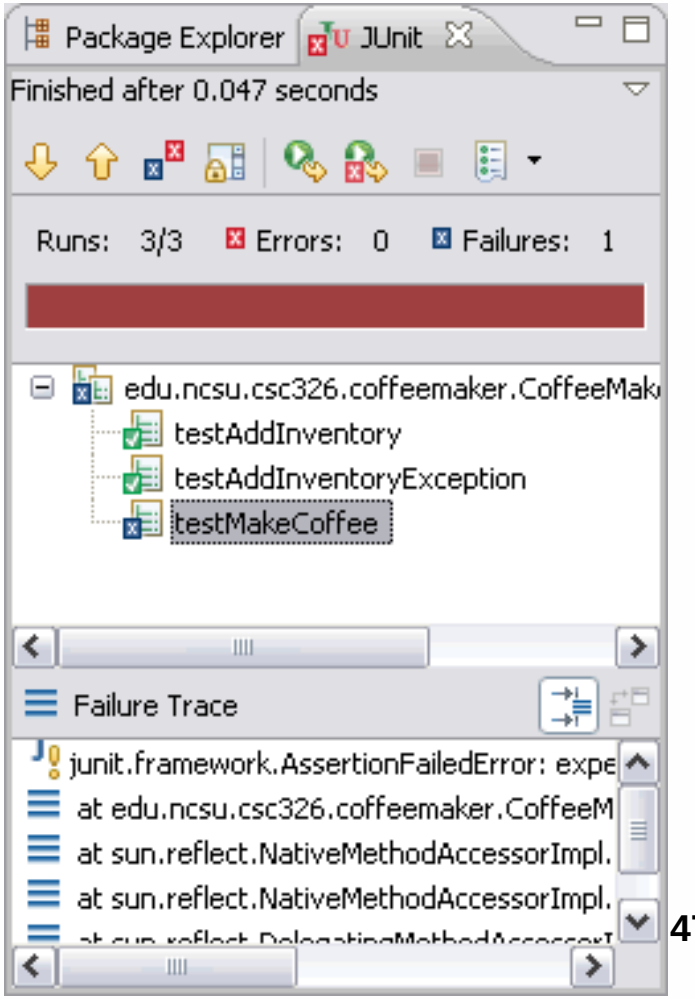
6.5 How to Use JUnit

- There are three ways to run JUnit Test Cases or Test Suites
  - You can right click on the test case class or test suite class and select **Run As > JUnit Test.**
  - You can select a test case or suite and click the arrow on the  icon or select Run from the toolbar, and select **Run As > JUnit Test.**
  - You can select a test case or suite and click the arrow on the  icon or select Run from the toolbar, and select **Open Run Dialog...** From here you will create a new JUnit test configuration, and name it. You can choose to run a single test case, a test suite, or run all test cases in a project or folder.

6.5 How to Use JUnit

- Step 4:Running JUnit Test Cases





6.5 How to Use JUnit

- Demo
- www.junit.org

6.5 How to Use JUnit

- Example：the difference between JUnit3.x and Junit 4.x

```
package test
public class BaseClass{
    public String method(){
        return this.getClass().getName();
    }
}
```

6.5 How to Use JUnit-Junit 3.x (Example)

```
package test
import junit.framwork.TestCase;   //引入TestCase类
public class BaseClassTest extends TestCase { //类必须继承TestCase
    BaseClass baseClass;

    protected void setUp() throws Exception{
        super.setUp();
        baseClass = new BaseClass();
    }
    public void testMethod() { //测试方法必须以test开头
        //通过assert*来检验
        assertTrue(baseClass.method().equals("test.BaseClass"));
    }
}
```

6.5 How to Use JUnit-Junit 4.x(Example)

```
package test
import org.junit.Test; //引入Test类
import static org.junit.Assert.*; //引入Assert.*包
public class BaseClassTestNew{ //这里不用再继承TestCase类
    BaseClass baseClass;
    @Before
    protected void runBeforeTest(){
        baseClass = new BaseClass();
    }
    @Test    // 以标签@Test标记，名字可以任意取
    public void methodOne(){
        //仍然是通过assert*来检验
        assertTure(baseClass.method().equals("test.BaseClass.class")));
    }
}
```

6.5 How to Use JUnit

- **JUnit 3与 JUnit 4的主要区别**
  - **1.**在JUnit3中需要继承TestCase类，但在JUnit4中已经不需要继承TestCase
  - **2.**在JUnit3中需要覆盖TestCase中的setUp和tearDown方法，其中setUp方法会在测试执行前被调用以完成初始化工作，而tearDown方法则在结束测试结果时被调用，用于释放测试使用中的资源，而在JUnit4中，只需要在方法前加上@Before，@After
  - **3.**在JUnit3中对某个方法进行测试时，测试方法的命令是固定的，例如对addBook这个方法进行测试，需要编写名字为testAddBook的测试方法，而在JUnit4中没有方法命令的约束，在方法的前面加上@Test,这就代表这个方法是测试用例中的测试方法

6.5 How to Use JUnit

- **JUnit 3与 JUnit 4的主要区别**
  - **4.**新的断言assertThat
  - **5.** @BeforeClass 和 @AfterClass 。在JUnit3，如果所有的test case仅调用一次setUp()和tearDown()需要使用TestSetup类
  - **6.**测试异常处理@Test(expected = DataFormatException.class)
  - **7.**设置超时@Test(timeout = 1000)
  - **8.**忽略测试@Ignore

Thank you !