Agile Software Development

Lecture 7: Software Testing

Mahmoud El-Gayyar

elgayyar@ci.suez.edu.eg

Slides are a modified version of the slides by Prof. Kenneth M. Anderson

Outline

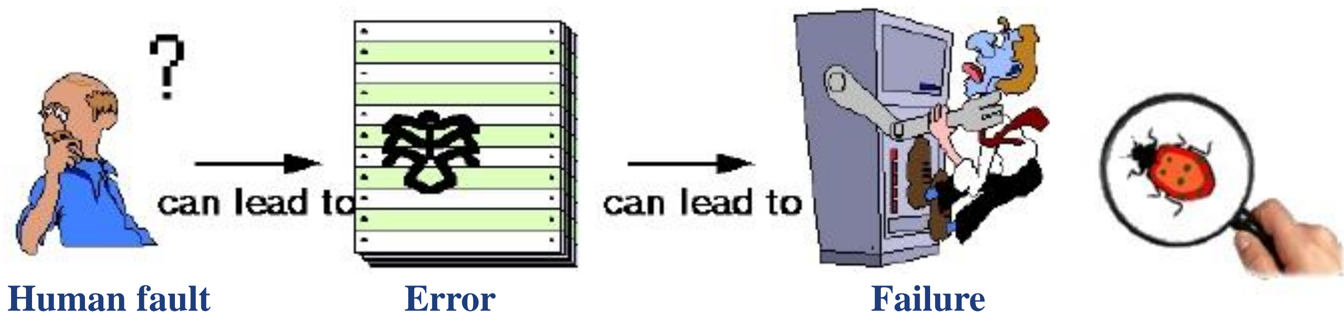
- *Testing Terminology*
- *Types of Testing*
- *Unit Testing*
  - Black-Box Testing
  - White-Box Testing
- *JUnit (Testing in Java)*

What is Testing?

- ***Testing:*** *Find the differences between the expected behavior and the observed behavior.*
- ***Goal:*** *Design tests that exercise defects in the system and to reveal problems*
- → *A* ***successful*** *test is a test that identifies faults*

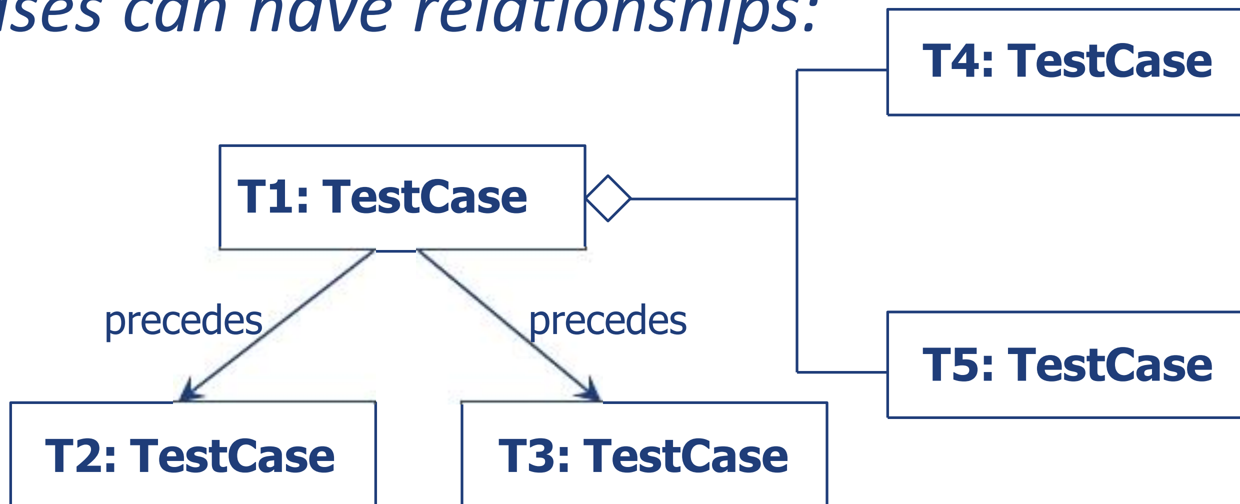
Type of Errors

- ***Fault (Bug):*** *A design or coding mistake that may cause abnormal component behavior.*
- ***Error:*** *The system is in a state such that further processing by the system will lead to a failure.*
- ***Failure:*** *Any perceivable deviation of the observed behavior from the specified behavior.*



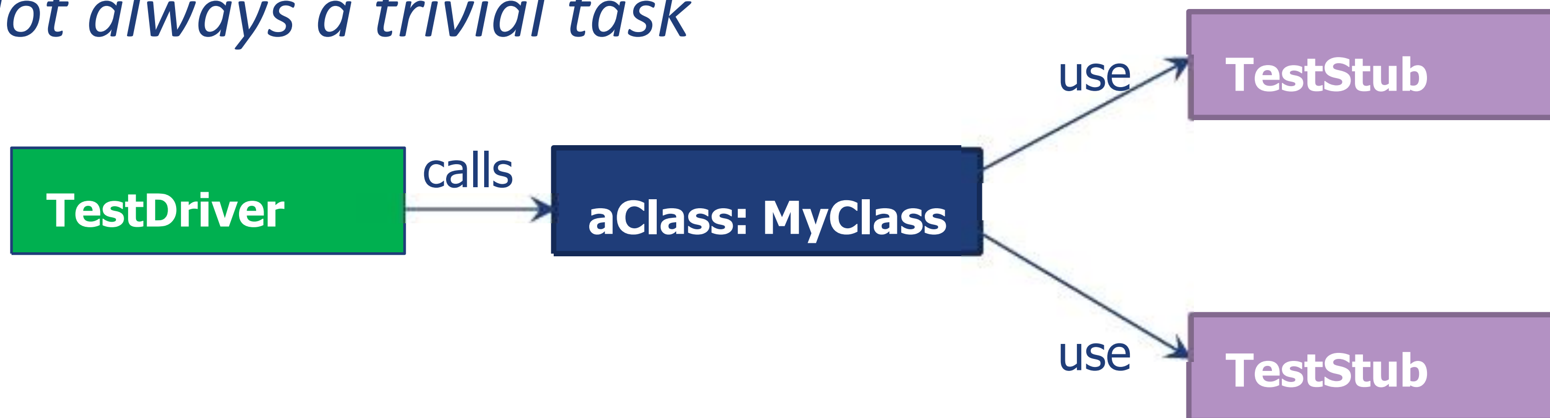
Test Cases

- ***Test Case:*** *set of* ***input data*** *and* ***expected results*** *that exercise a component with the purpose of causing failures and detecting faults*
- *Test cases can have relationships:*



Test Stub and Drivers

- ***Test Driver:*** *simulates the part of the system that calls the component under test (CUT)*
- ***Test Stub (or Mock):*** *simulates component that are called by the tested component*
- *Not always a trivial task*



Type of Testing

- ***Unit Testing:***
  - Individual subsystem (modules)
  - Carried out by developers
  - ***Goal:*** Confirm that subsystems is correctly coded and carries out the intended functionality
- ***Integration Testing:***
  - Groups of subsystems (collection of classes) and eventually the entire System to ensure that modules work together correctly
  - Carried out by developers or test team
  - ***Goal:*** Test the interface among the subsystem

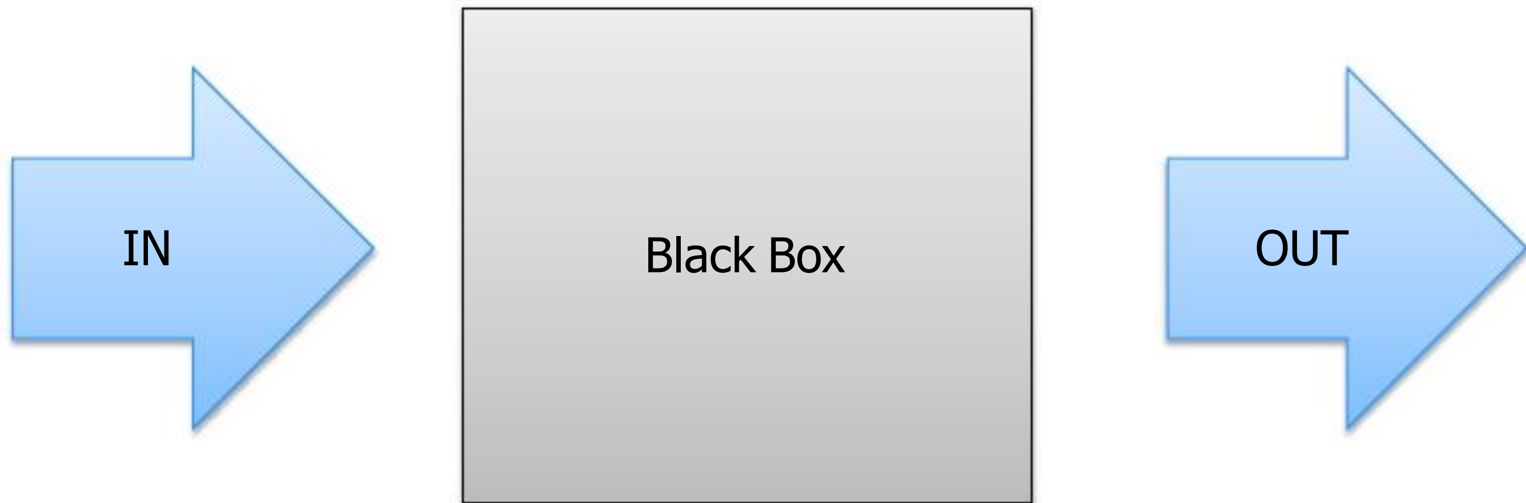
Type of Testing

- ***System Testing:***
  - The entire system
  - Carried out by test team (developers shouldn't be involved)
  - ***Goal:*** Determine if the system meets the requirements (functional and global)
  - ***Functional Testing:*** Test of functional requirements
  - ***Performance Testing:*** Test of non-functional requirements
- ***Acceptance Tests***
  - performed by ***users*** to check that the delivered system meets their needs
  - In large, custom projects, developers will be on-site to install system and then respond to problems as they arise

Multi-Level Testing

- *Black Box Testing*
  - Does the system behave as predicted by its specification
- *White Box Testing*
  - Since, we have access to most of the code, lets make sure we are covering all aspects of the code: statements, branches, …
- *Gray Box Testing*
  - Having a bit of insight into the architecture of the system, does it behave as predicted by its specification

Black-Box Testing



Expected Data

Test Data

Actual Data

Ideally: All possible value
Unmanagable! To expensive!
=> Equivalence classes

Comparison of expected and actual data

Black-Box Testing

- ***Focus:*** *I/O behavior. If for any given input we can predict the output, then the module passes the test.*
  - Do not deal with the internal aspects of the tested component
  - Almost always impossible to generate all possible inputs
- ***Goal:*** *Reduce number of test cases*
- ***Method:*** *Equivalence Testing*
  - Divide input conditions into equivalence classes
  - Choose test cases for each equivalence class. (Example: If an object is supposed to accept a negative number, testing one negative number is enough)

Equivalence Classes

- *Square Root Function*
  - Negative, Zero, Positive
  - Test data = {-16, 0, 25}, Expected Result = {4, 0, 5}
- *In a computer store, the computer item can have a quantity between -500 to +500. What are the equivalence classes?*
  - Valid class: -500 <= QTY <= +500
  - Invalid class: QTY > +500
  - Invalid class: QTY < -500

Gray Box Testing

- *Use **knowledge of system's architecture** to create a more complete set of black box tests*
- *Verifying logging information*
  - for each function is the system really updating all ***internal state*** correctly
- *Data destined for other systems*
- *"Looking for Scraps"*
  - Is the system correctly cleaning up after itself temporary files, memory leaks, data duplication/deletion

Your time !!

- *Exercise : Black and Gray box testing*

White-Box Testing

- *Focus on the internal structureof the component.*
- ***Goal:*** *Each state in dynamic model of an object and each interaction among the objects should be tested.*
- *Four quality metrics for white-box testing:*
  - **Statement Coverage**
    - *Is each statement exercised(covered) by a test?*
  - **Loop Coverage**
    - *Is each loop body executed zero times, exactly once, and more than once (consecutively)?*
  - **Branch Coverage**
    - *Is each possible outcome of an decision covered?*
  - **Path Coverage**
    - *Is each possible path covered?*

Example: White-Box Testing

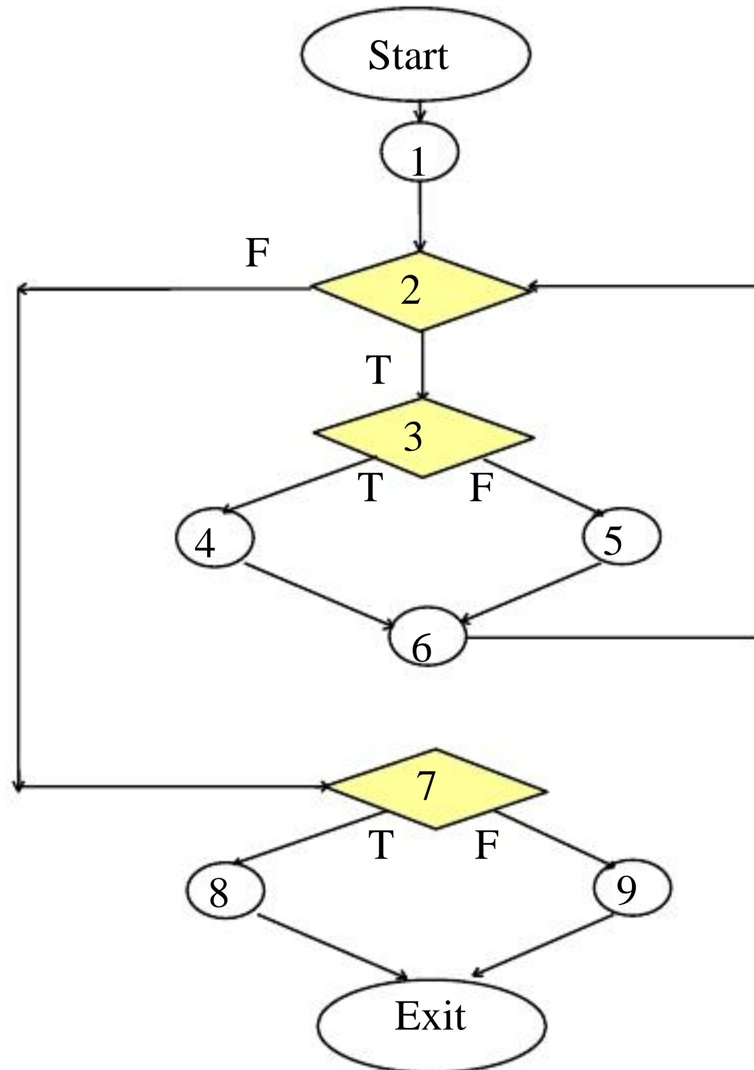
```
findMean(File ScoreFile){
    float SumOfScores = 0.0;
    int NumberOfScores = 0;
    float Mean=0.0; float Score;
    Read(ScoreFile, Score);
    while !EOF(ScoreFile) {
        if (Score > 0.0 ) {
                SumOfScores = SumOfScores + Score;
                NumberOfScores++;
        }

        Read(ScoreFile, Score);
    }
    /* Compute the mean and print the result */
    if (NumberOfScores > 0) {
            Mean = SumOfScores / NumberOfScores;
            printf(" The mean score is %f\n", Mean);
    } else
            printf ("No scores found in file\n");
}
```

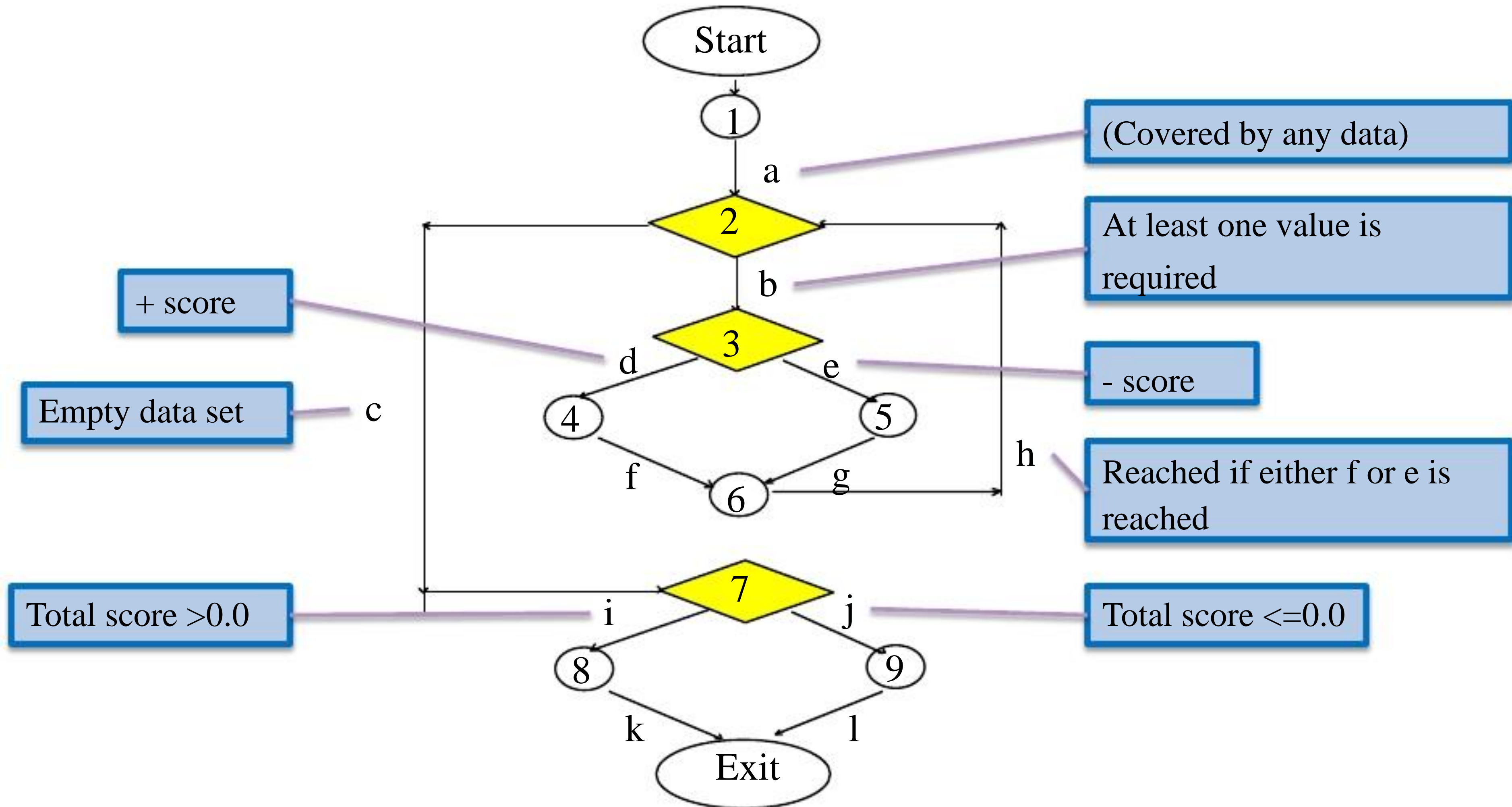
Example: White-Box Testing: Determine the paths

```
findMean(File ScoreFile)
{ float SumOfScores = 0.0;
  int NumberOfScores = 0;
  float Mean=0.0; float Score;
  Read(ScoreFile, Score);                                    ← 1
2 while !EOF(ScoreFile) {
  3 if (Score > 0.0 ) {
        SumOfScores = SumOfScores + Score;
        NumberOfScores++;                                    ← 4
      }
    5
    Read(ScoreFile, Score);                                  ← 6
  }
  /* Compute the mean and print the result */
7 if (NumberOfScores > 0) {
      Mean = SumOfScores / NumberOfScores;
      printf(" The mean score is %f\n", Mean);               ← 8
  } else
      printf ("No scores found in file\n");                  ← 9
}
```

Constructing the Logic Flow Diagram

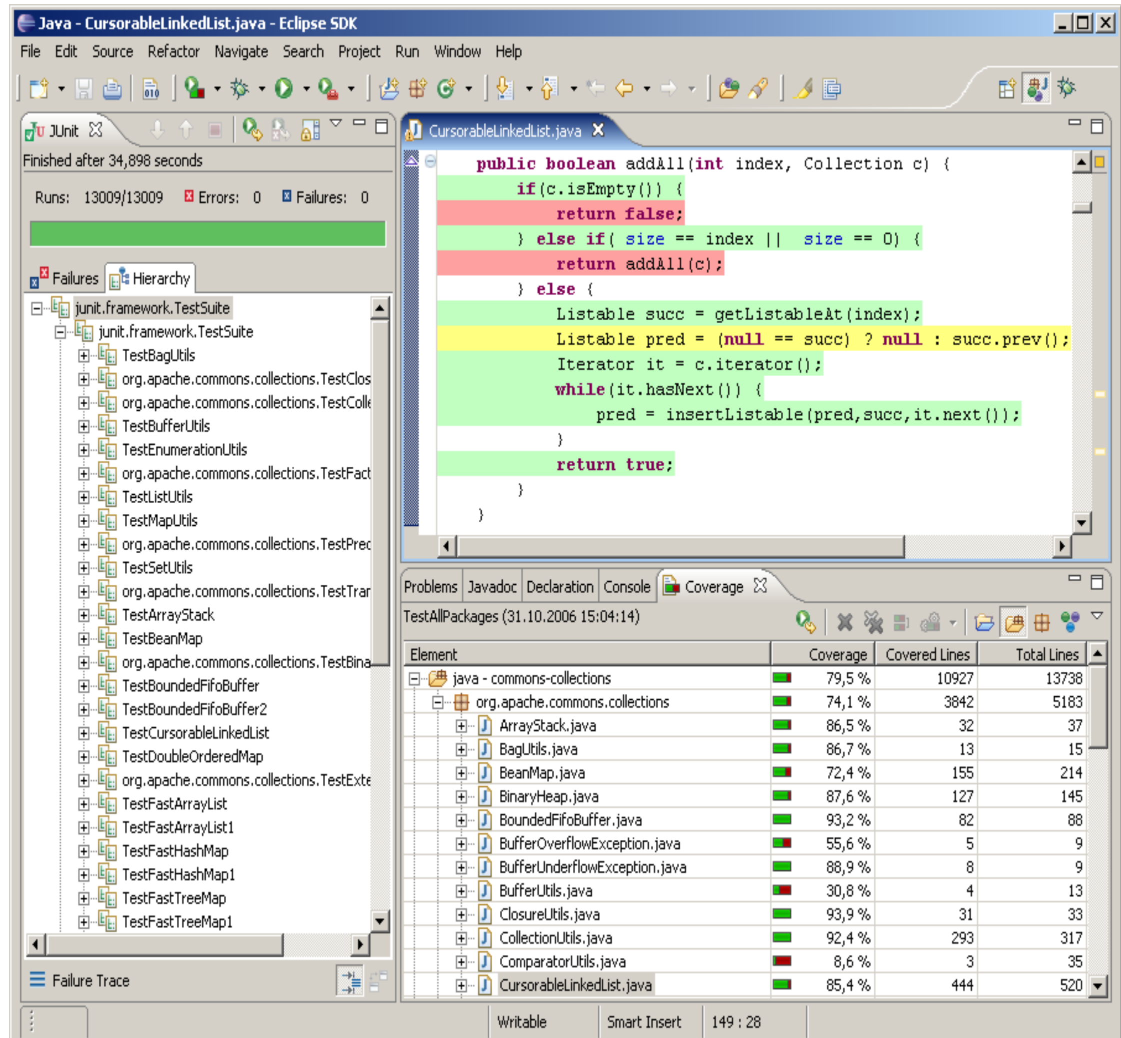


Finding Test Cases



Code Coverage Tools

- *Tools that can track code coverage metrics for you (mostly just statement and branch coverage)*
- *Ex: EclEmma for Java*



Test Automation

- *It is important that your tests be automated*
  - More likely to be run
  - More likely to catch problems as changes are made
- *testing frameworks allow tests to be run with a single command*
  - e.g. JUnit for JAVA (but there are lots of testing frameworks out there)
  - Test presentation !!!!!

Testing in Java: JUnit

- *De facto standard Java framework for unit (object) testing*
- *JUnit helps the programmer:*
  - define and execute tests and test suites
  - write and debug code
  - integrate code and always be ready to release a working version
- *JUnit is not included in Sun’s SDK, but almost all IDEs include it*

Junit: Terminology

- *A **test fixture** sets up the data (both objects and primitives) that are needed to run tests*
  - Example: If you are testing code that updates an employee record, you need an employee record to test it on
- *A **test case** tests the response of a single method to a particular set of inputs*
- *A **test suite** is a collection of test cases*

A Simple Example

- *Suppose you have a class* ***Arithmetic*** *with static methods*
  - int multiply(int x,int y)
  - boolean isPositive(int x)

```
import org.junit.*;
Import static org.junit.Assert.*;
public Class ArithmaticTest{
  @Test
  public void testMultiply(){
    assertEquals(4, Arithmetic.multiply(2,2));
    assertEquals(4,Arithmetic.multiply(3,-5));
  }
  @Test
  public void testIsPositive(){
    assertTrue( Arithmetic.isPositive(2));
    assertFalse( Arithmetic.isPositive(-2));
    assertFalse( Arithmetic.isPositive(0));
  }
}
```

Test Suites

- *You can define a suite of tests*

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)

@Suite.SuiteClasses({
    FirstTest.class,
    SecondTest.class,
    ThirdTest.class
})

public Class AllTests(){ }
```

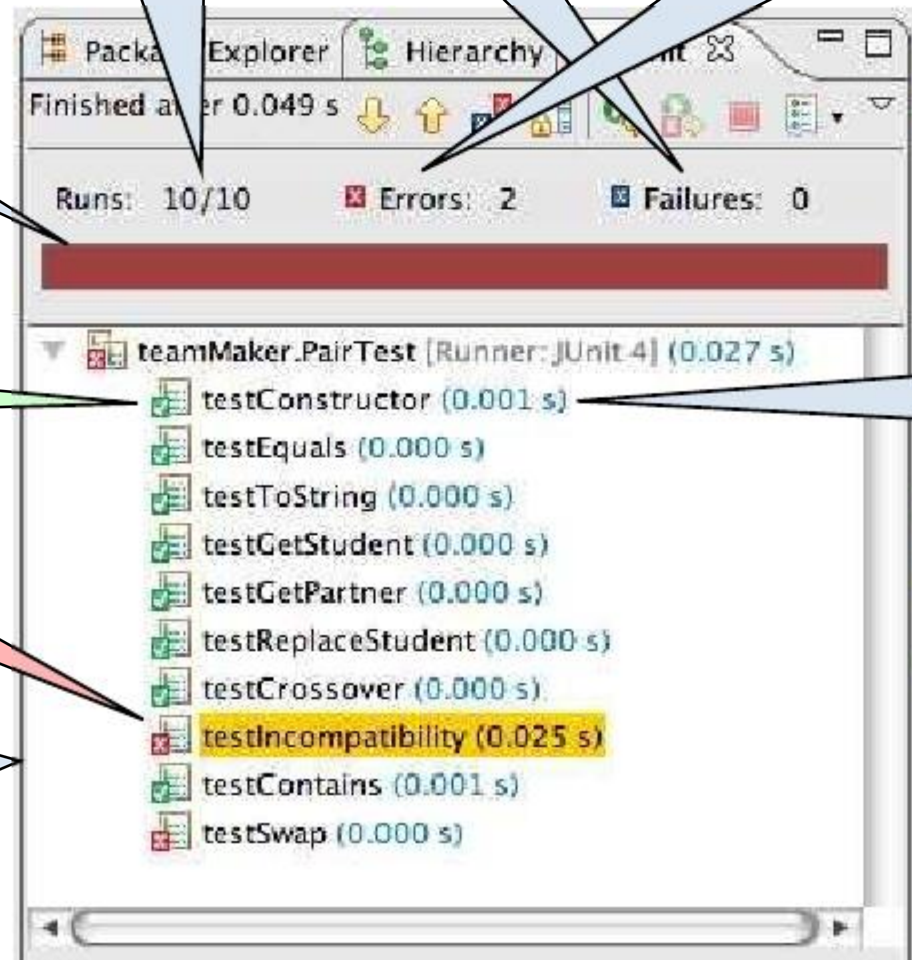
JUnit in Eclipse I

Bar is green if *all* tests pass, red otherwise

Ran 10 of the 10 tests

No tests failed, but…

Something unexpected happened in two tests



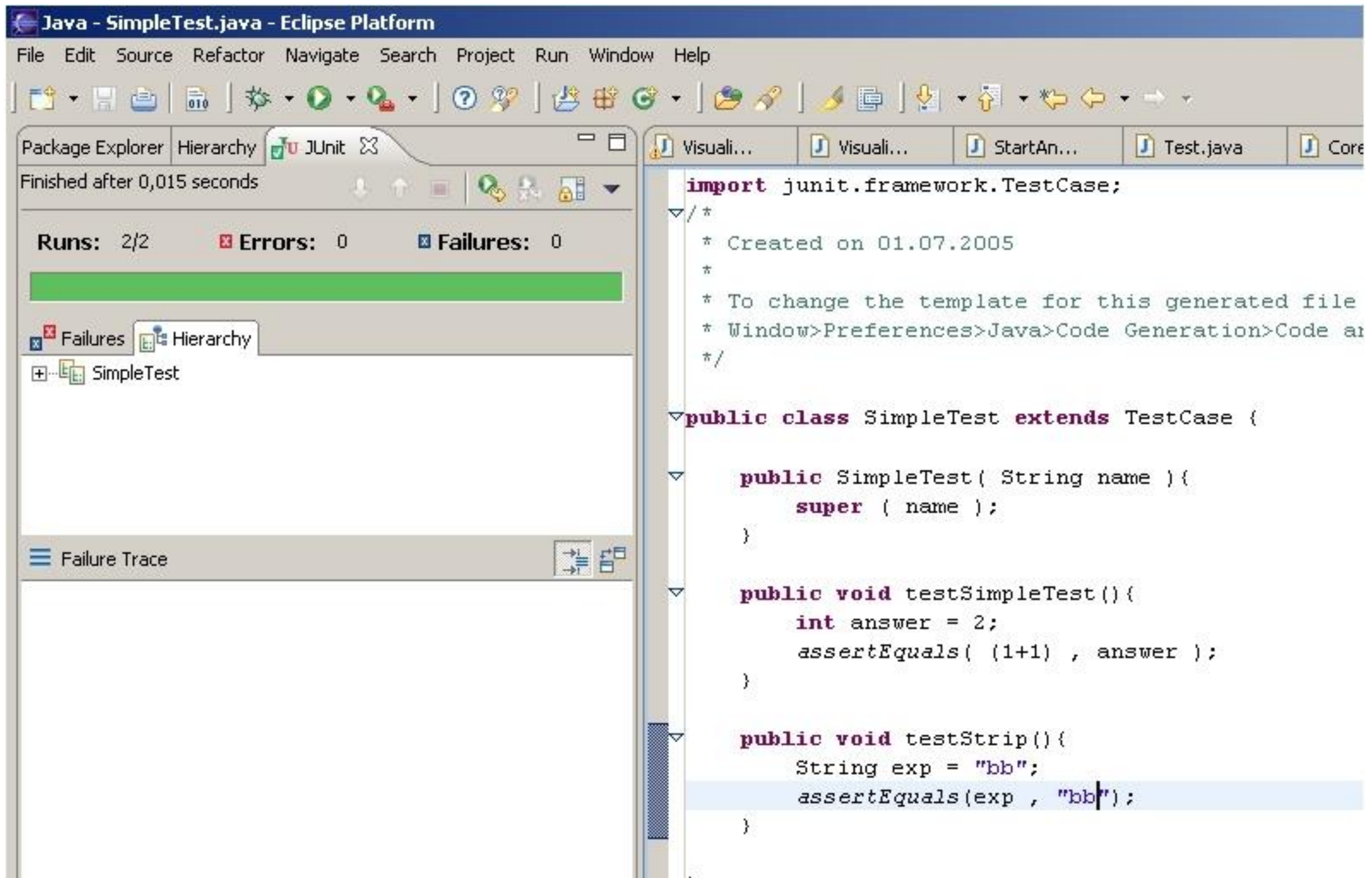
This test passed

This is how long the test took

Something is wrong

Depending on your preferences, this window might show *only* failed tests

JUnit in Eclipse II



Continuous Integration

- *Since test automation is so critical, systems known as **continuous integration frameworks (CI)** have emerged*
- *CI systems wrap version control, compilation, and testing into a single repeatable process*

Summary

- *Testing Terminology*
- *Types of Testing*
- *Multi-Level Testing*
  - Black-Box Testing
  - Gray-Box Testing
  - White-Box Testing
- *Test Automation and Continuous Integration*