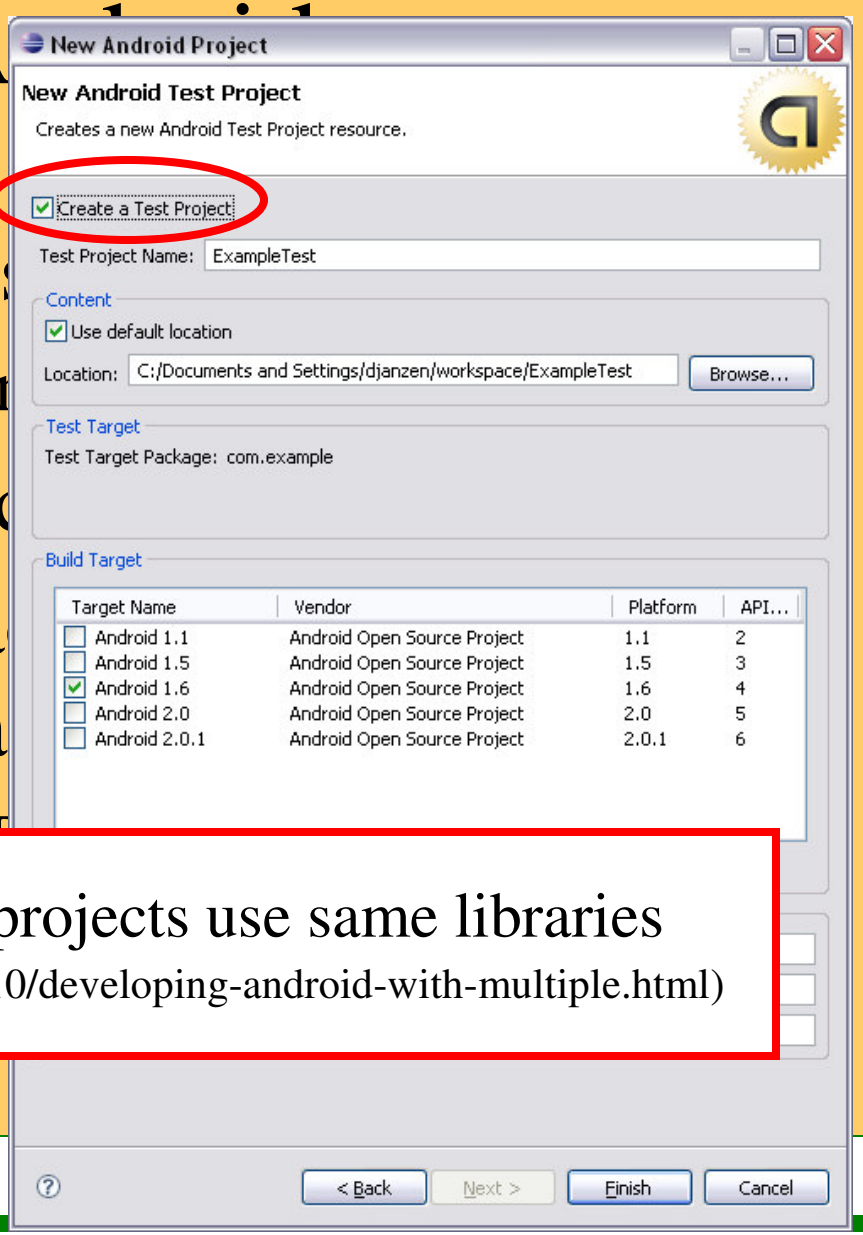
Upcoming Assignments

- Lab 4 due Wednesday, February 10
- Friday Quiz on TDD/JUnit 3
- Furlough Day Monday, February 8
  - Great time to work with team on Course Project
  - Each student should have a class from your course project ready for review by Wednesday, February 10
- How-to's and presentations
  - https://sites.google.com/site/androidhowto/presentations

TDD in A

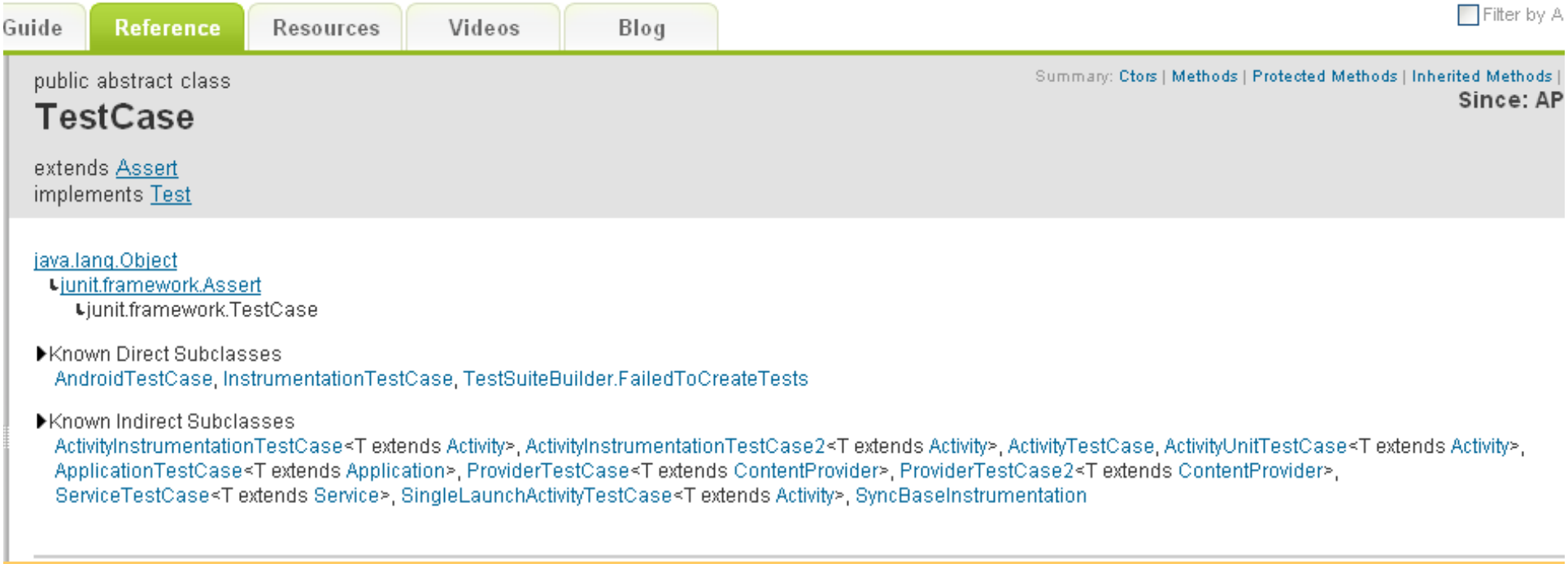
- Android SDK integrates
  - 1.6 does not support JUr
- Many helper TestCase c
- Recommended best pra
  separate project but sha



Beware if both src and test projects use same libraries

(see http://jimshowalter.blogspot.com/2009/10/developing-android-with-multiple.html)

Android TestCase Classes



Android TestCase Classes

- Basic JUnit tests
  - TestCase (run tests with assert methods)
- When you need an Activity Context
  - AndroidTestCase (see getContext())
- When you want to use a Mock Context
  - ApplicationTestCase (call setContext() before calling createApplication() which calls onCreate())

Android TestCase Classes

- When you want to test just one Activity
  - ActivityUnitTestCase (allows you to ask if the Activity has started another Activity or called finish() or requested a particular orientation)
- When you want to do a functional test on an Activity
  - ActivityInstrumentationTestCase2 (allows you to send key events to your Activity)

Android TestCase Classes

- When you want to test a Content Provider
  - ProviderTestCase2
- When you want to test a Service
  - ServiceTestCase
- When you want to stress test the UI
  - Monkey
    - http://d.android.com/guide/developing/tools/monkey.html

Android TestCase How-to

- Add instrumentation to ApplicationManifest.xml

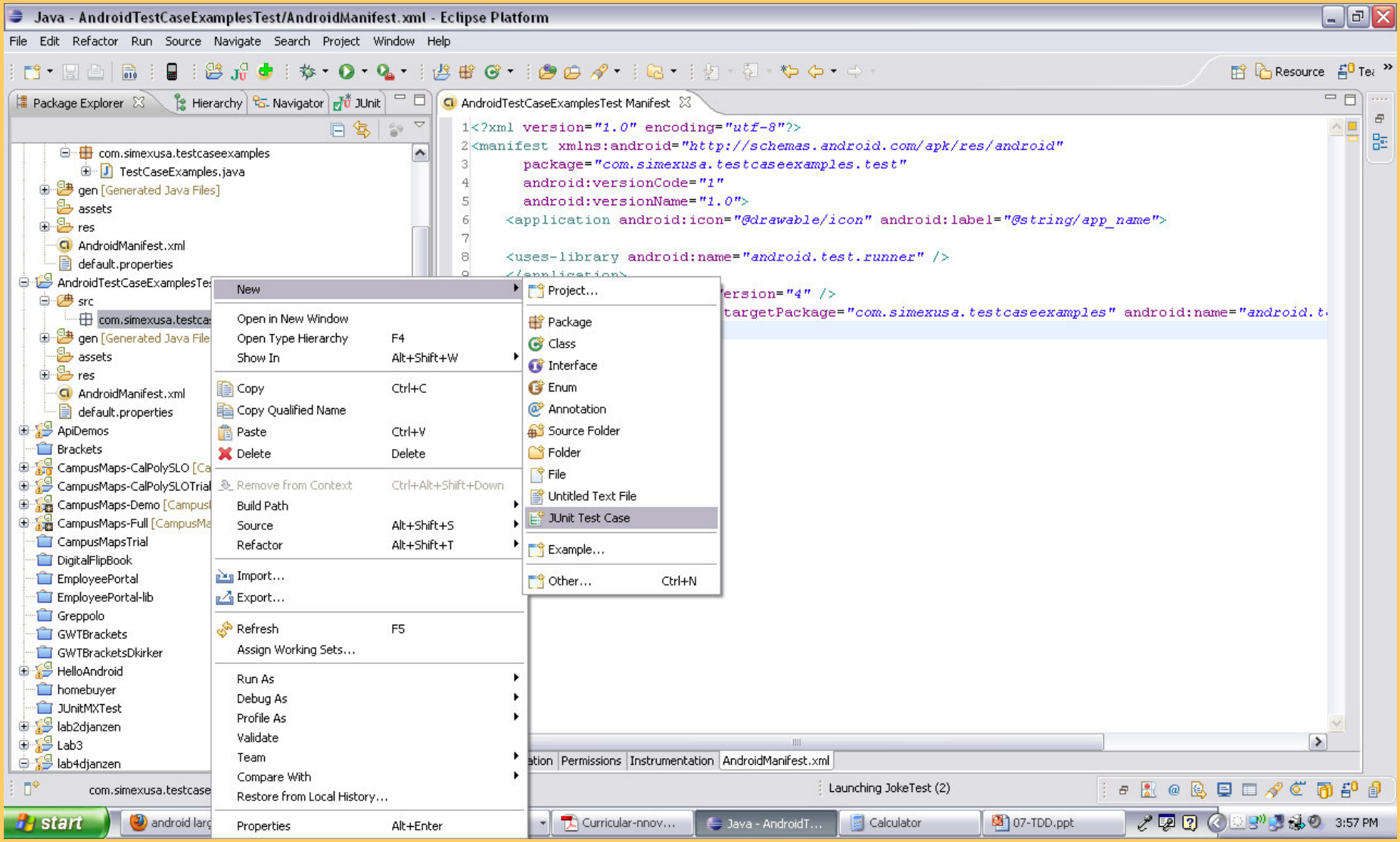
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.simexusa.testcaseexamples"  android:versionCode="1"
    android:versionName="1.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name"
          android:debuggable="true">
    <uses-library android:name="android.test.runner" />
    <activity android:name="SomeActivity android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>
  <uses-sdk android:minSdkVersion="3" />
  <instrumentation android:name="android.test.InstrumentationTestRunner"
                   android:targetPackage="com.simexusa.testcaseexamples"
                   android:label="Tests for my example."/>
</manifest>
```

Android TestCase How-to

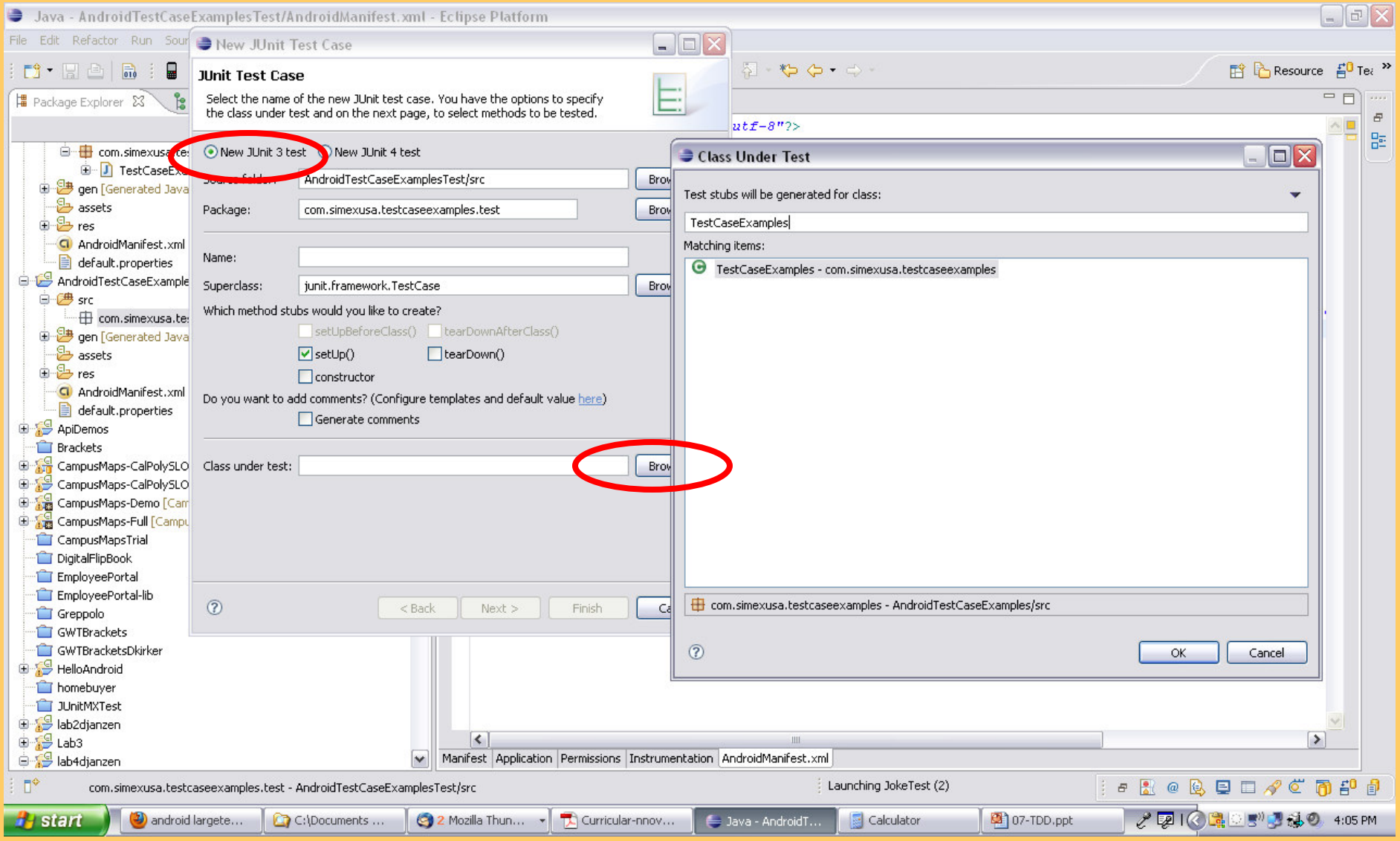
- Add instrumentation to ApplicationManifest.xml
  - When creating a second project

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.simexusa.testcaseexamples.test"
    android:versionCode="1"
    android:versionName="1.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name">

  <uses-library android:name="android.test.runner" />
  </application>
  <uses-sdk android:minSdkVersion="4" />
  <instrumentation android:targetPackage="com.simexusa.testcaseexamples"
          android:name="android.test.InstrumentationTestRunner" />
</manifest>
```

- Create a new JUnit Test Case



- Create a new JUnit Test Case

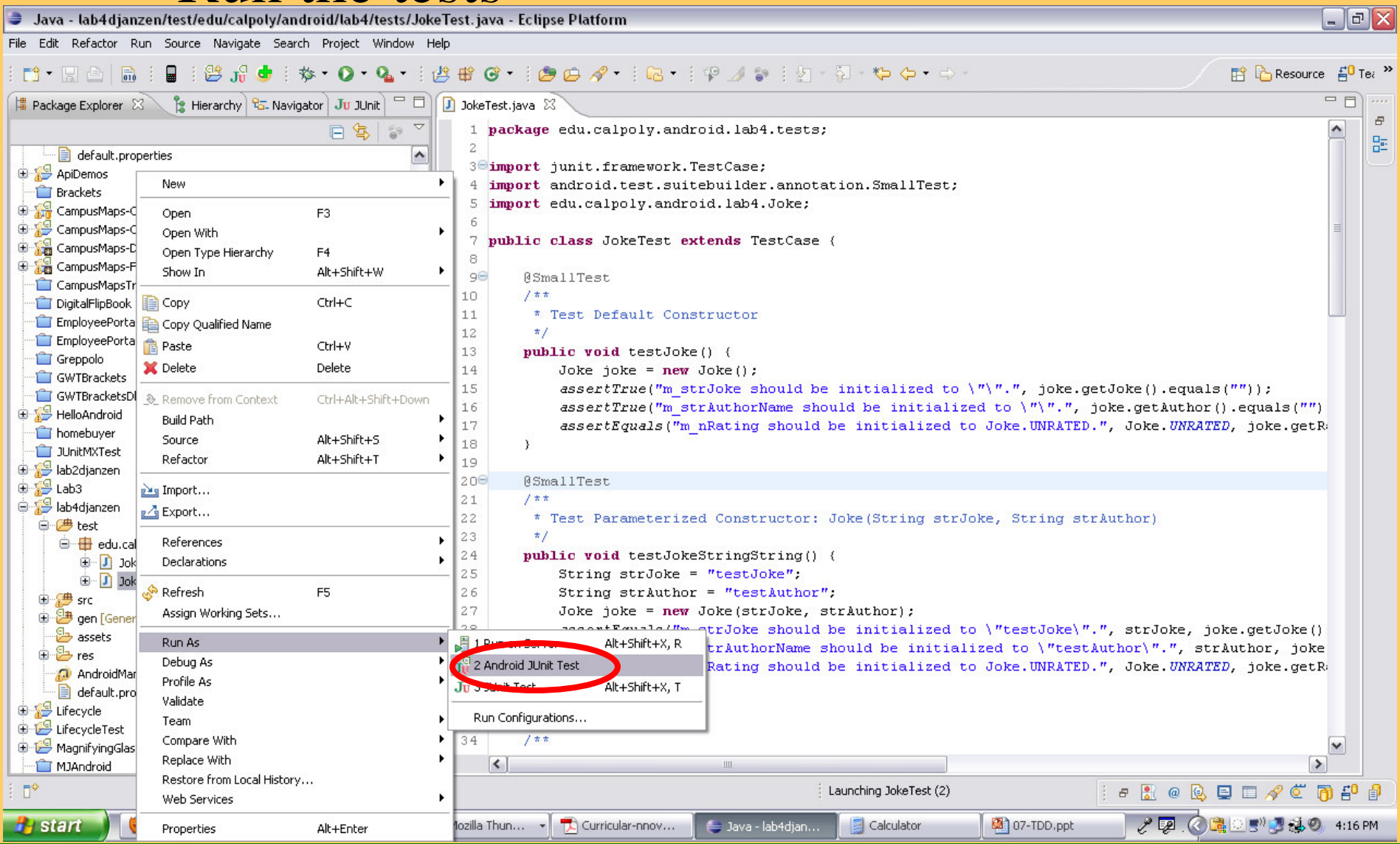


Testing POJO's

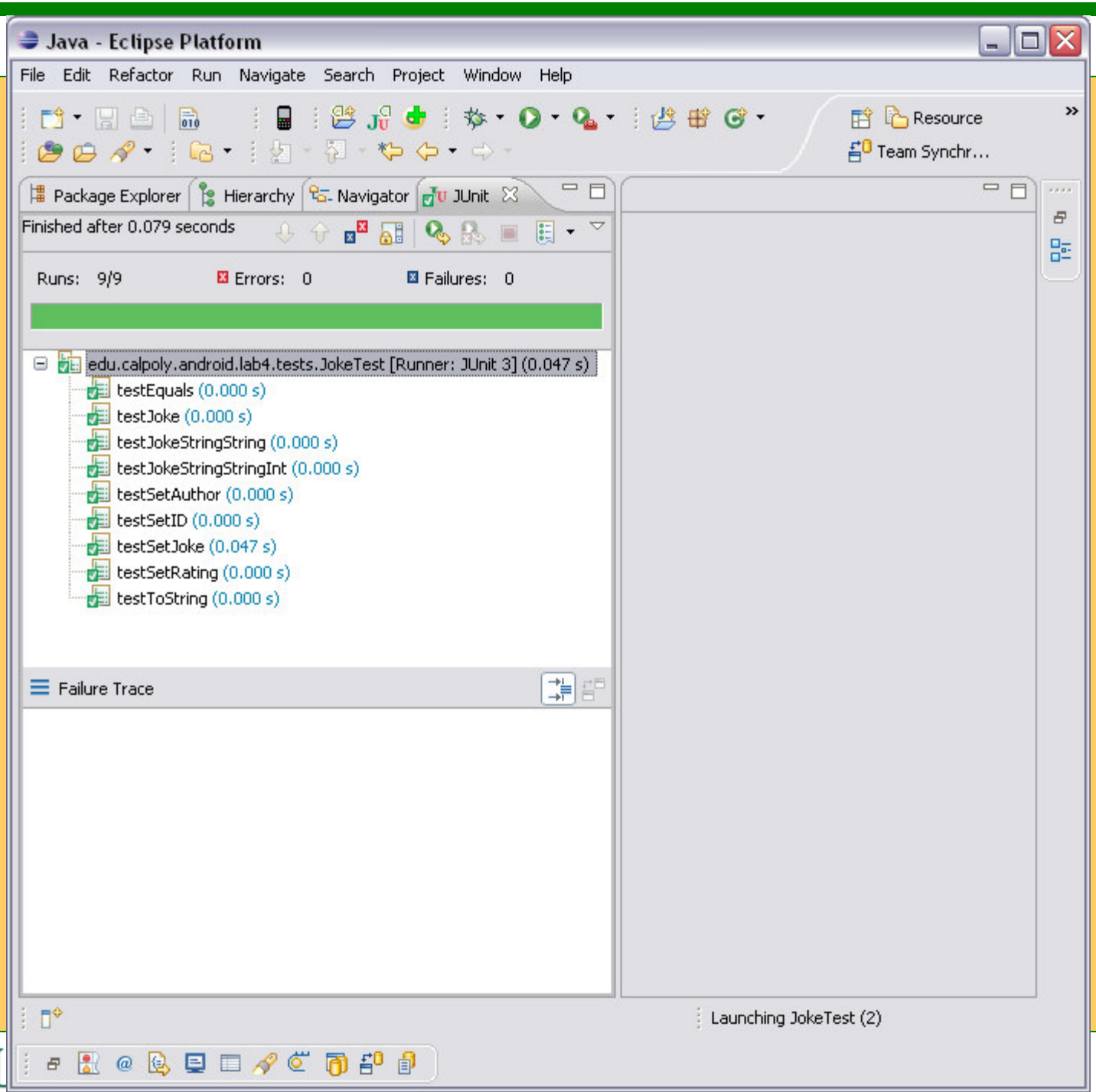
- Plain Old Java Objects
  - (i.e. independent of frameworks like Android or J2EE)

```
import junit.framework.TestCase;
import edu.calpoly.android.lab4.Joke;

public class JokeTest extends TestCase {

 public void testJoke() {
    Joke joke = new Joke();
    assertTrue("m_strJoke should be initialized to \"\".", joke.getJoke().equals(""));
    assertTrue("m_strAuthorName should be initialized to \"\".",
                        joke.getAuthor().equals(""));
    assertEquals("m_nRating should be initialized to Joke.UNRATED.",
                        Joke.UNRATED, joke.getRating());
 }
}
```

- Run the tests



CAL POLY


Java - Eclipse Platform
File Edit Refactor Run Navigate Search Project Window Help
Resource
Team Synchr...
Package Explorer
Hierarchy
Navigator
JUnit
Finished after 0.079 seconds
Runs: 9/9
Errors: 0
Failures: 0
edu.calpoly.android.lab4.tests.JokeTest [Runner: JUnit 3] (0.047 s)
testEquals (0.000 s)
testJoke (0.000 s)
testJokeStringString (0.000 s)
testJokeStringStringInt (0.000 s)
testSetAuthor (0.000 s)
testSetID (0.000 s)
testSetJoke (0.047 s)
testSetRating (0.000 s)
testToString (0.000 s)
Failure Trace
Launching JokeTest (2)

JUnit 3 How-to

- Import the JUnit framework

```
import junit.framework.*;
```

- Create a subclass of TestCase

```
public class TestBank extends TestCase {
```

- Write methods in the form testXXX()
- Use assertXXX() methods

```
public void testCreateBank() {
    Bank b = new Bank();
    assertNotNull(b);
}
```

- Compile test and functional code; Run a TestRunner to execute tests; Keep the bar green!

Fixtures

- Notice redundancy in test methods

```
import junit.framework.TestCase;
public class TestBank extends TestCase {
    public void testCreateBank() {
        Bank b = new Bank();
        assertNotNull(b);
    }
    public void testCreateBankEmpty() {
        Bank b = new Bank();
        assertEquals(b.getNumAccounts(),0);
    }
}
```

- Common test setup can be placed in a method named setUp() which is *run before each test*

setUp()

```
import junit.framework.*;
public class TestBank extends TestCase {
    private Bank b;
    public void setUp() {
        b = new Bank();
    }
    public void testCreateBank() {
        assertNotNull(b);
    }
    public void testCreateBankEmpty() {
        assertEquals(b.getNumAccounts(),0);
    }
    public void testAddAccount() {
        Account a = new Account("John Doe",123456,0.0);
        b.addAccount(a);
        assertEquals(b.getNumAccounts(),1);
    }
}
```

setUp() is run before *each* test

tearDown()

- tearDown() is run after each test
  - Used for cleaning up resources such as files, network, or database connections

```
import junit.framework.TestCase;
public class TestBank extends TestCase {
    private Bank b;
    public void setUp() {
        b = new Bank();
    }
    public void tearDown() {
        b = null;
    }
    …
}
```

tearDown() is run after *each* test

Grouping Tests with @xTest

- Some tests run fast, others don't
  - You can separate them with @SmallTest, @MediumTest, @LargeTest

```
public class JokeTest extends TestCase {

@SmallTest
/**
 * Test Default Constructor
 */
  public void testJoke() {
     Joke joke = new Joke();
     assertTrue("m_strJoke should be initialized to \"\".", joke.getJoke().equals(""));
     assertTrue("m_strAuthorName should be initialized to \"\".",
                    joke.getAuthor().equals(""));
     assertEquals("m_nRating should be initialized to Joke.UNRATED.",
                    Joke.UNRATED, joke.getRating());
}
```

Running Tests with @xTest

- Run the tests with adb from the command line
  - http://developer.android.com/reference/android/test/InstrumentationTestRunner.html

```
C:\adb shell am instrument -w -e size
small edu.calpoly.android.lab4/android.test.InstrumentationTestRunner


edu.calpoly.android.lab4.tests.dflt.JokeCursorAdapterTest:....
edu.calpoly.android.lab4.tests.dflt.JokeTest:.........
Test results for InstrumentationTestRunner=.............
Time: 1.975


OK (13 tests)
```

Testing Campus Maps

```
package com.simexusa.campusmaps_full;

import com.simexusa.campusmaps_full.CampusMap;
import com.simexusa.campusmaps_full.TranslatorUtility;
import junit.framework.TestCase;

public class TestTranslatorUtility extends TestCase {

  protected void setUp() throws Exception {
     super.setUp();
  }
  public void testTranslateLatToY() {
     double b1lat = 35.302518;
     double b2lat = 35.299365;
     int b1py = 445;
     int b2py = 840;
     double latitude = 35.299812;
     assertEquals(784,TranslatorUtility.latToCoordinate(latitude,b1lat,b2lat,b1py,b2py));
}
```

Testing Campus Maps

```
package com.simexusa.campusmaps_full;

import com.simexusa.campusmaps_full.CampusMap;
import com.simexusa.campusmaps_full.TranslatorUtility;
import junit.framework.TestCase;

public class TestTranslatorUtility extends TestCase {

  protected void setUp() throws Exception {
     super.setUp();
  }
  public void testTranslateLatToY() {
     double b1lat = 35.302518;
     double b2lat = 35.299365;
     int b1py = 445;
     int b2py = 840;
     double latitude = 35.299812;
     assertEquals(784,TranslatorUtility.latToCoordinate(latitude,b1lat,b2lat,b1py,b2py));
}
```

Test complicated methods

Testing Campus Maps

```
public void testSplit2() {
   String s = "go+180";
   String [] results = s.split("\\+");
   assertEquals(results[0],"go");
   assertEquals(results[1],"180");
}

public void testParser() {
   CampusMap [] maps = TranslatorUtility.parseMapData(
            "Bethel College|http://www.bethelks.edu/map/bcmap.png|" +
            "39.298664|39.296903|-76.593761|-76.590527|383|614|171|352\n");
   assertEquals(maps[0].title,"Bethel College");
}

public void testGetMaps() {
   CampusMap[] myCampusMaps = new CampusMap[5];
   TranslatorUtility.retrieveMapData("http://simexusa.com/cm/fav5defaultmapdata.txt",
                                     myCampusMaps);
   assertEquals(myCampusMaps[0].title,"Cal Poly - SLO");
}
```

Explore library API's

Verify it works like I expect

Functional tests
This one gets data from the web

TDD in Software Development Lifecycle

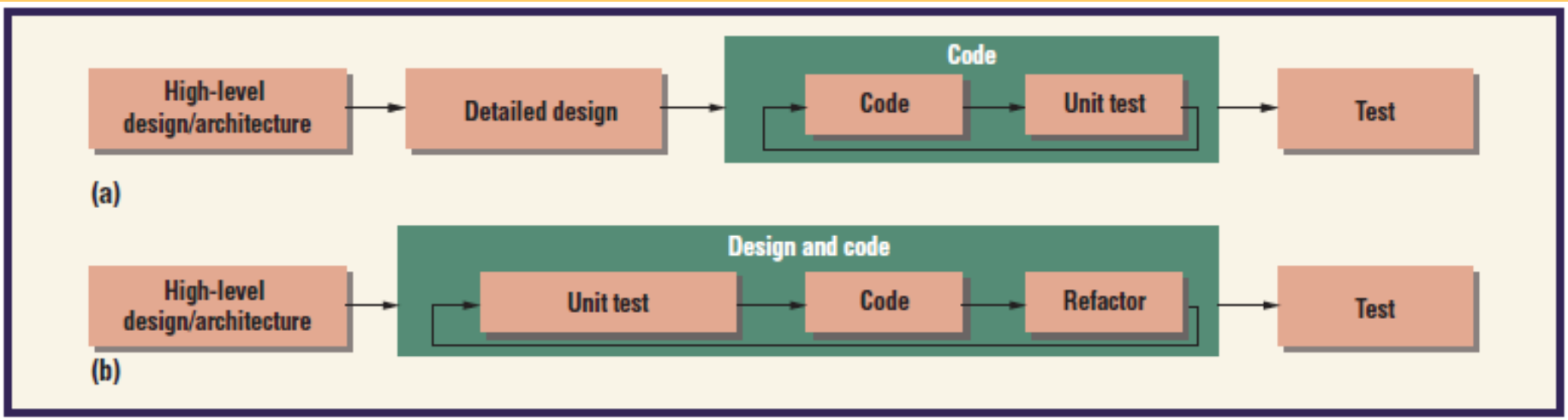
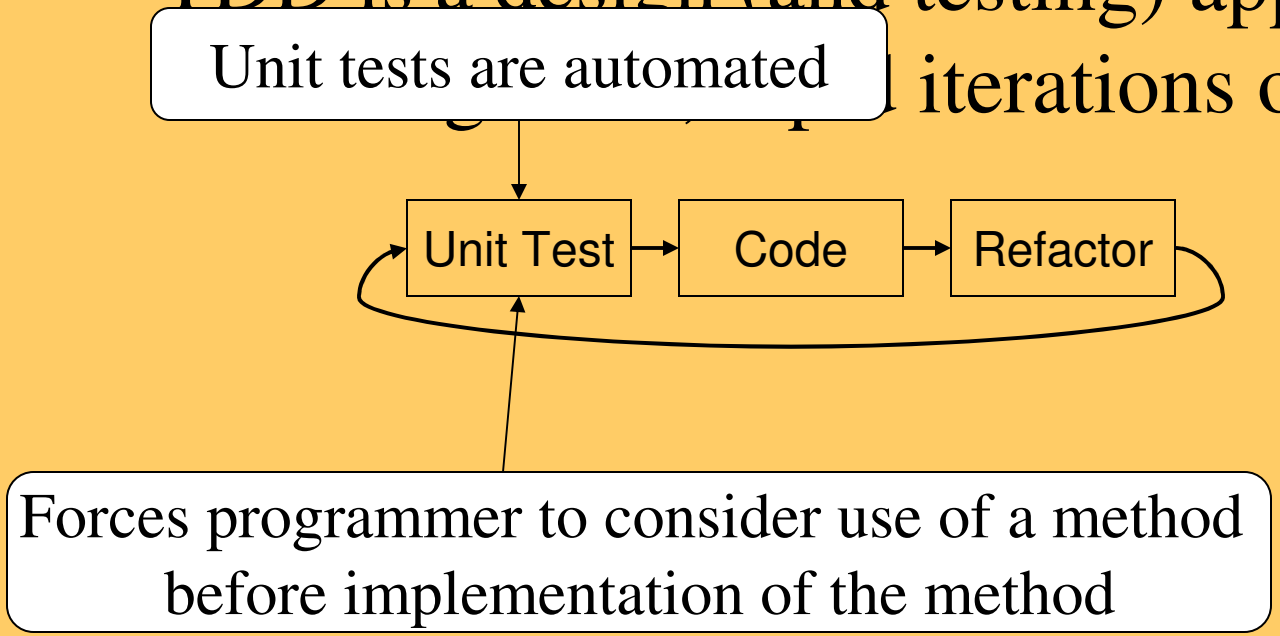


Figure 1. Development flow: (a) traditional test-last and (b) test-driven development/test-first flow.

What is Test-Driven Development?

- TDD is a design (and testing) approach
  iterations of



TDD Example: Requirements

- Ensure that passwords meet the following criteria:
  - Between 6 and 10 characters long
  - Contain at least one digit
  - Contain at least one upper case letter

TDD Example: Write a test

```
import static org.junit.Assert.*;
import org.junit.Test;

public class TestPasswordValidator {
  @Test
  public void testValidLength() {
     PasswordValidator pv = new PasswordValidator();
     assertEquals(true, pv.isValid("Abc123"));
  }
}
```

Needed for JUnit 4

This is the teeth of the test

Cannot even run test yet because PasswordValidator doesn't exist!

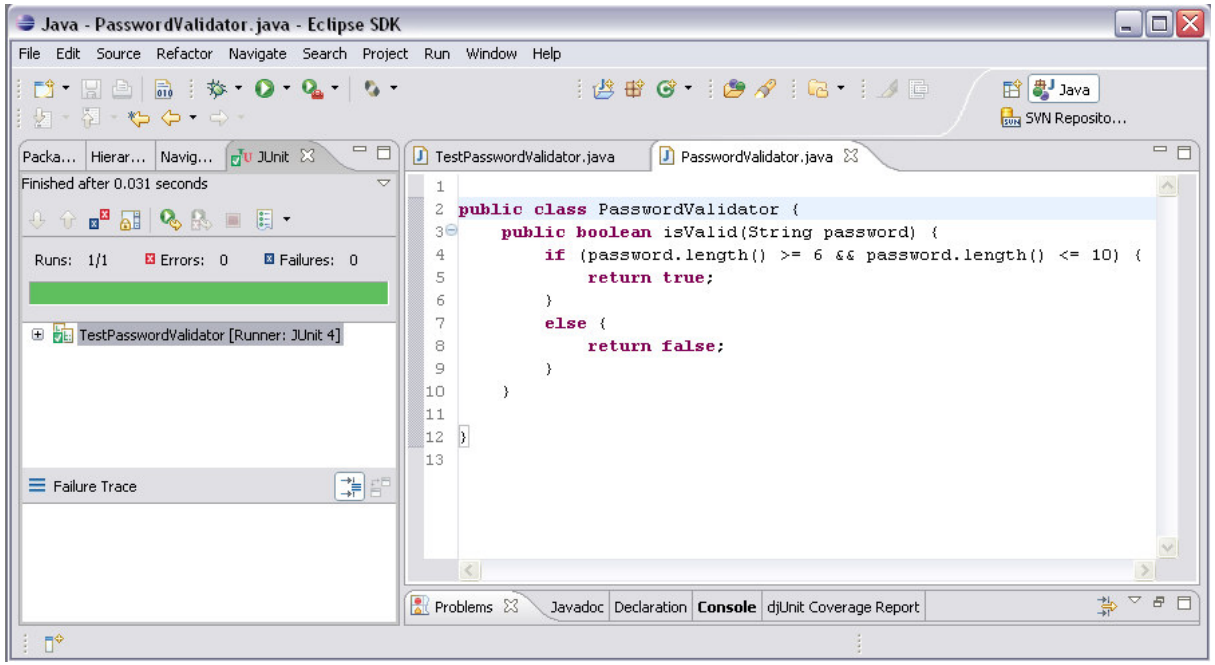
TDD Example: Write a test

```
import static org.junit.Assert.*;
import org.junit.Test;

public class TestPasswordValidator {
  @Test
  public void testValidLength() {
     PasswordValidator pv = new PasswordValidator();
     assertEquals(true, pv.isValid("Abc123"));
  }
}
```

Design decisions:
- class name, constructor,
- method name, parameters and return type

TDD Example: Write the code

```
public class PasswordValidator {
   public boolean isValid(String password) {
      if (password.length() >= 6 && password.length() <= 10) {
         return true;
      }
      else {
         return false;
      }
   }
}
```



TDD Example: Refactor

```
import static org.junit.Assert.*;
import org.junit.Test;

public class TestPasswordValidator {
  @Test
  public void testValidLength() {
     PasswordValidator pv = new PasswordValidator();
     assertEquals(true, pv.isValid("Abc123"));
  }
}
```

Do we really need an instance of PasswordValidator?

TDD Example: Refactor the test

```
import static org.junit.Assert.*;
import org.junit.Test;

public class TestPasswordValidator {
  @Test
  public void testValidLength() {
     assertEquals(true, PasswordValidator.isValid("Abc123"));
  }
}
```

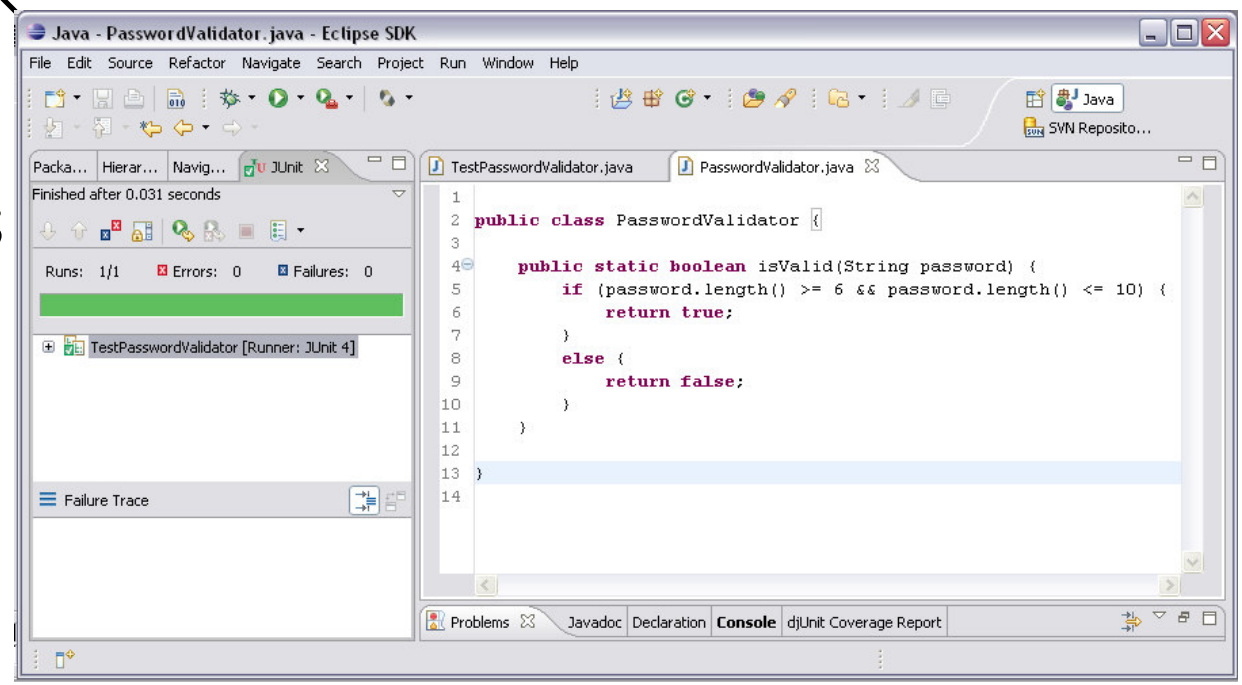

Design decision:
static method

What is Refactoring?

- Changing the *structure* of the code without changing its *behavior*
  - Example refactorings:
    - Rename
    - Extract method/extract interface
    - Inline
    - Pull up/Push down
- Some IDE's (e.g. Eclipse) include automated refactorings

TDD Example: Refactor the code

```
public class PasswordValidator {
   public static boolean isValid(String password) {
      if (password.length() >= 6 && password.length() <= 10) {
         return true;
      }
      else {
         return false;
      }
   }
}
```



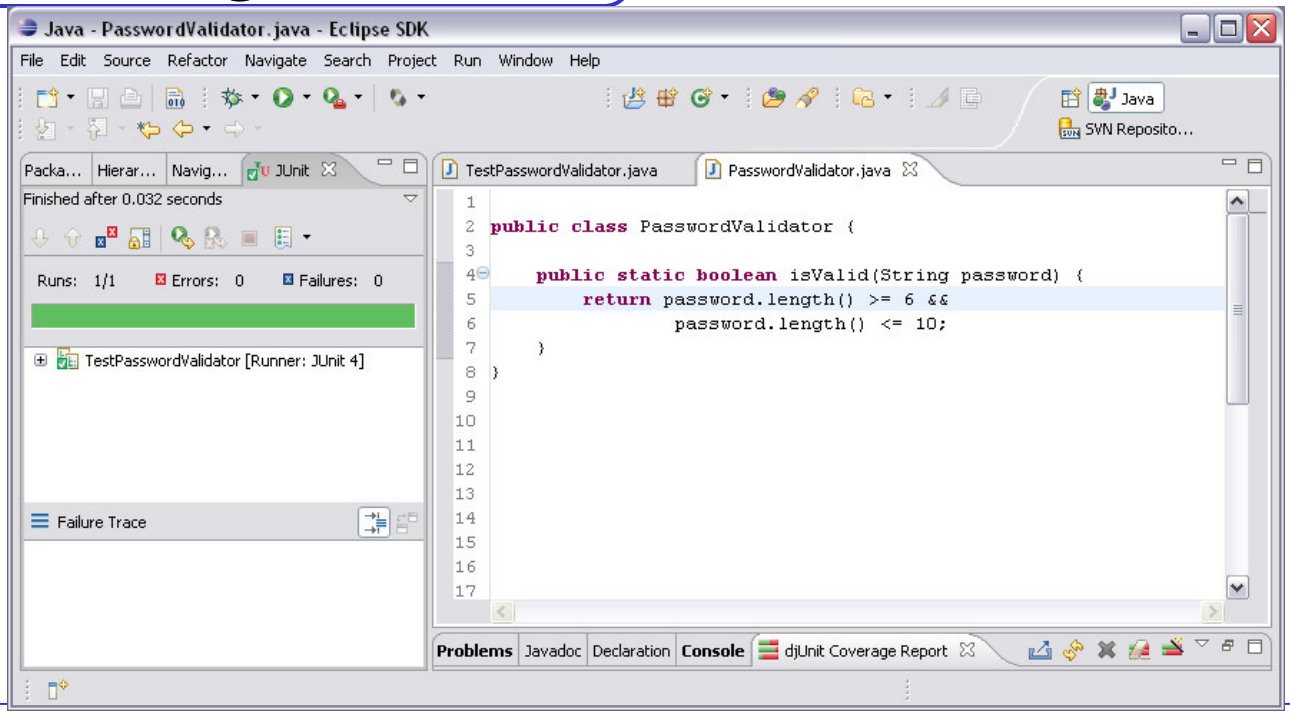
TDD Example: Refactor the code

```
public class PasswordValidator {
   public static boolean isValid(String password) {
      if (password.length() >= 6 && password.length() <= 10) {
         return true;
      }
      else {
         return false;
      }
   }
}
```

Can we simplify this?

TDD Example: Refactoring #1

```
public class PasswordValidator {
   public static boolean isValid(String password) {
      return password.length() >= 6 &&
             password.length() <= 10;
   }
}
```



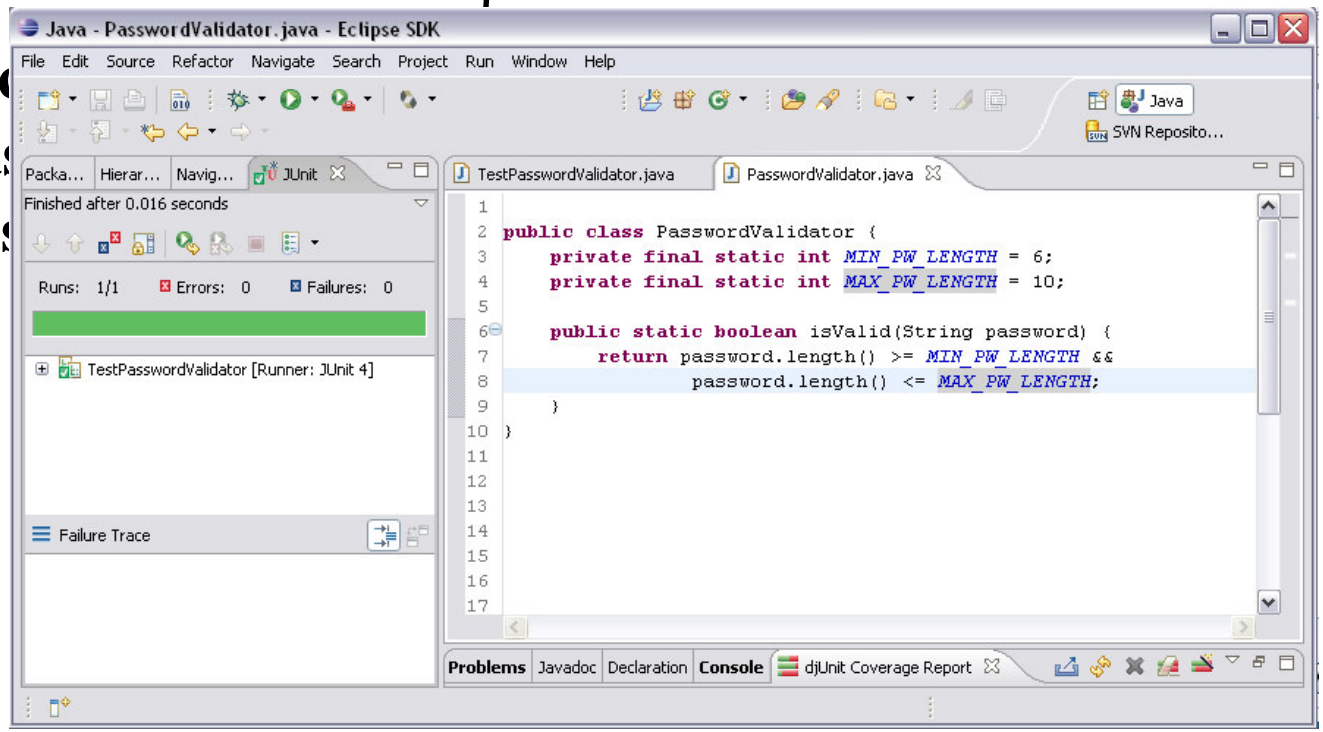
TDD Example: Refactoring #1

```
public class PasswordValidator {
   public static boolean isValid(String password) {
      return password.length() >= 6 &&
             password.length() <= 10;
   }
}
```


"Magic numbers" (i.e. literal constants that are buried in code) can be dangerous

TDD Example: Refactoring #2

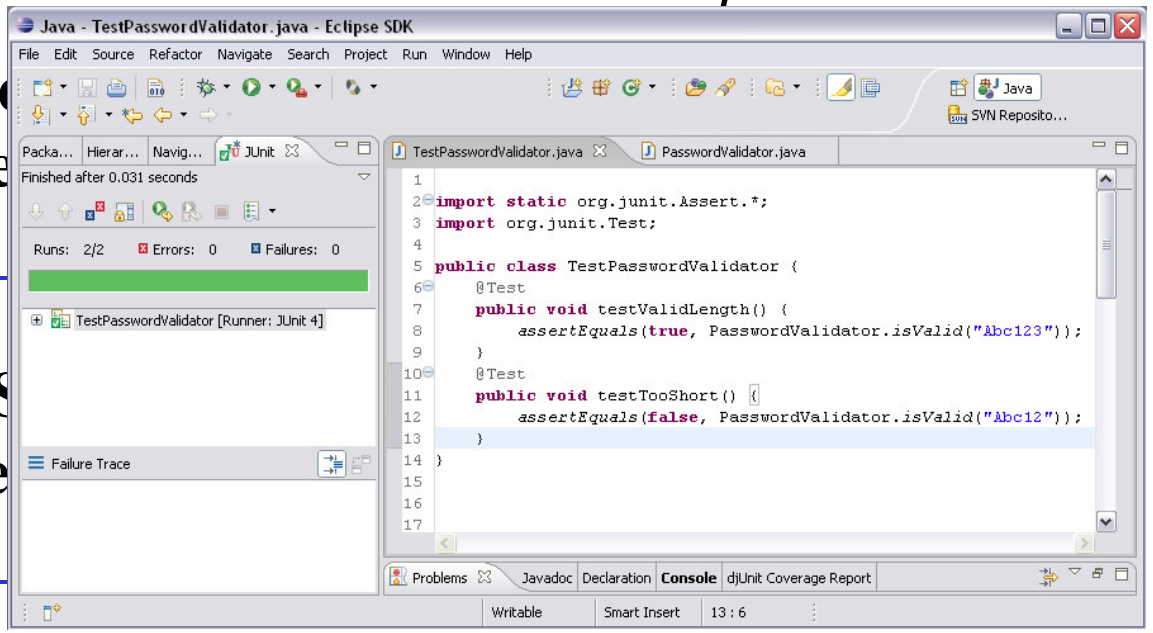
```
public class PasswordValidator {
    private final static int MIN_PW_LENGTH = 6;
    private final static int MAX_PW_LENGTH = 10;

    public stati
      return pas
                pas
    }
}
```



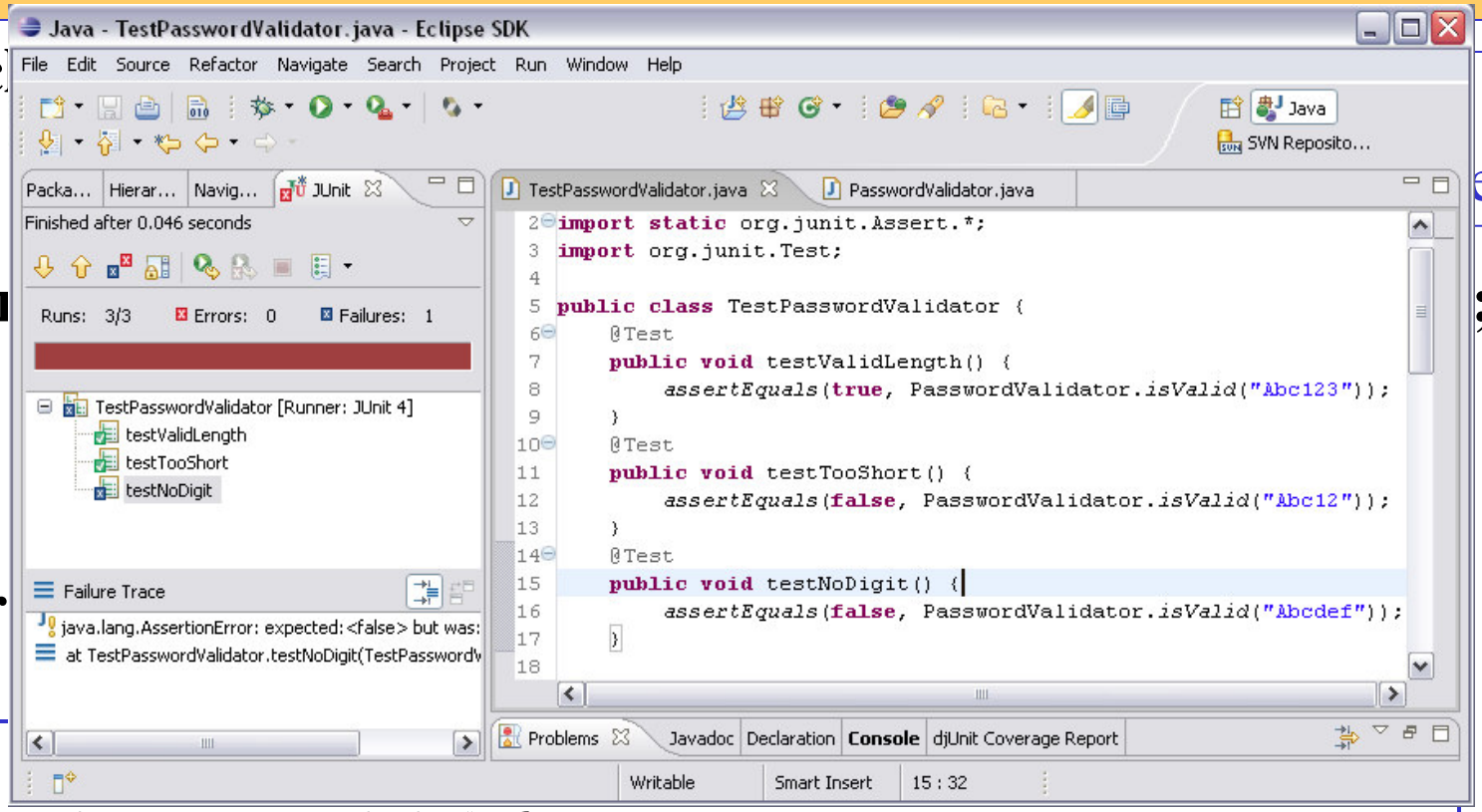
TDD Example: Write another test

```
import static org.junit.Assert.*;
import org.junit.Test;

public class TestPasswordValidator {
  @Test
  public void testVali
     assertEquals(true
  }
  @Test
  public void testToo
     assertEquals(false
  }
}
```

No design decisions; just unit testing



TDD Example: Write another test




```
public c
  @Test
  public
      asse
  }
  @Test
  public
    asser
  }
  @Test
  public void testNoDigit() {
      assertEquals(false, PasswordValidator.isValid("Abcdef"));
  }
}
```

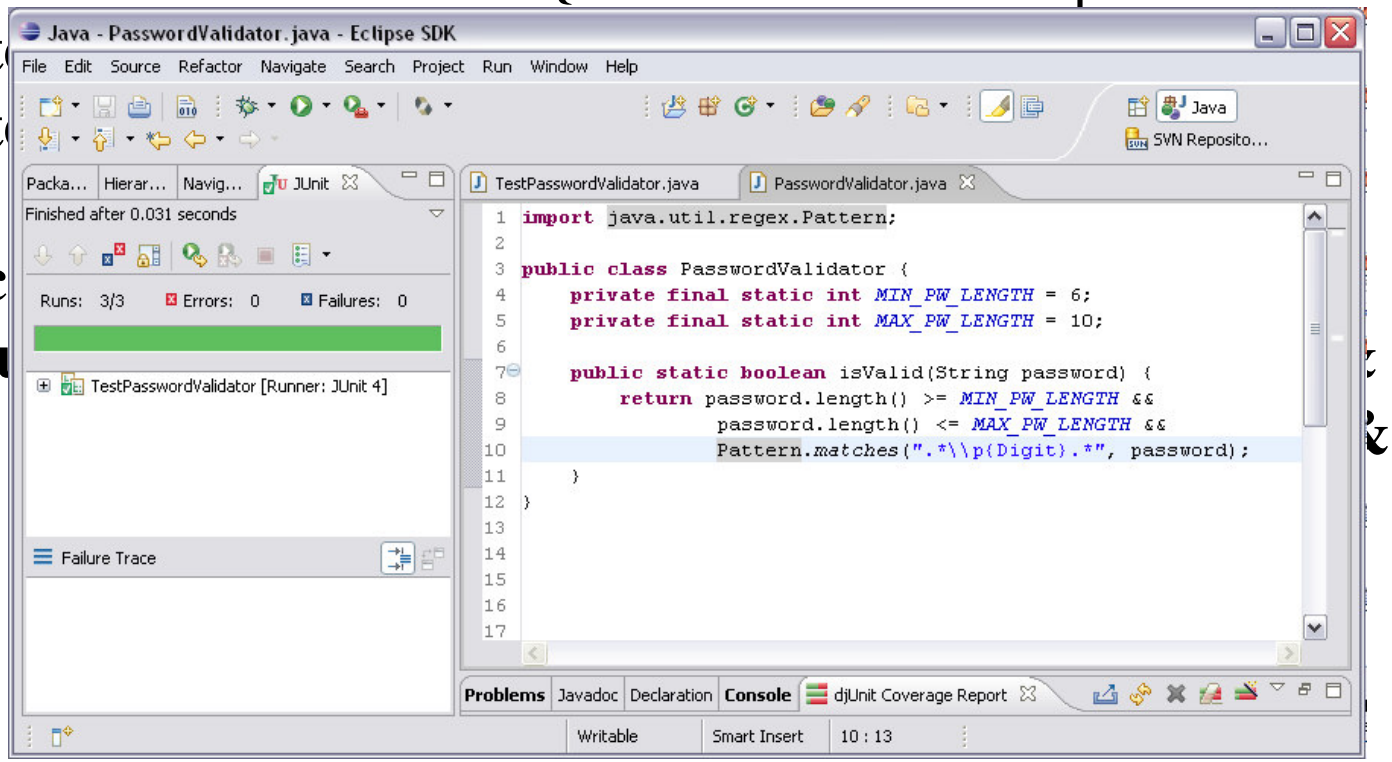
TDD Example: Make the test pass

```
public class PasswordValidator {
        private final static int MIN_PW_LENGTH = 6;
        private final static int MAX_PW_LENGTH = 10;

        public static boolean isValid(String password) {
            return password.length() >= MIN_PW_LENGTH &&
                        password.length() <= MAX_PW_LENGTH;
        }
    }
```

TDD Example: Make the test pass

```
import java.util.regex.Pattern;

public class PasswordValidator {
        privat
        privat

        public
            retu

        }
    }
```

Check for a digit



TDD Example: Refactor

```
import java.util.regex.Pattern;

public class PasswordValidator {
        private final static int MIN_PW_LENGTH = 6;
        private final static int MAX_PW_LENGTH = 10;

        public static boolean isValid(String password) {
            return password.length() >= MIN_PW_LENGTH &&
                   password.length() <= MAX_PW_LENGTH &&
                   Pattern.matches(".*\\p{Digit}.*", password);
        }
    }
```

Extract methods for readability

TDD Example: Done for now

```
import java.util.regex.Pattern;
public class PasswordValidator {
  private final static int MIN_PW_LENGTH = 6;
  private final static int MAX_PW_LENGTH = 10;
  private static boolean isValidLength(String password) {
     return password.length() >= MIN_PW_LENGTH &&
               password.length() <= MAX_PW_LENGTH;
  }
  private static boolean containsDigit(String password) {
     return Pattern.matches(".*\\p{Digit}.*", password);
  }
  public static boolean isValid(String password) {
     return isValidLength(password) &&
               containsDigit(password);
  }
}
```

Test-Driven Development

- **Short introduction**[1]
  - Test-driven development (TDD) is the craft of producing automated tests for production code, and using that process to *drive design* and *programming*. For every tiny bit of functionality in the production code, you first develop a test that specifies and validates what the code will do. You then produce exactly as much code as will enable that test to pass. Then you refactor (simplify and clarify) both the production code and the test code.

1. http://www.agilealliance.org/programs/roadmaps/Roadmap/tdd/tdd_index.htm

Some Types of Testing

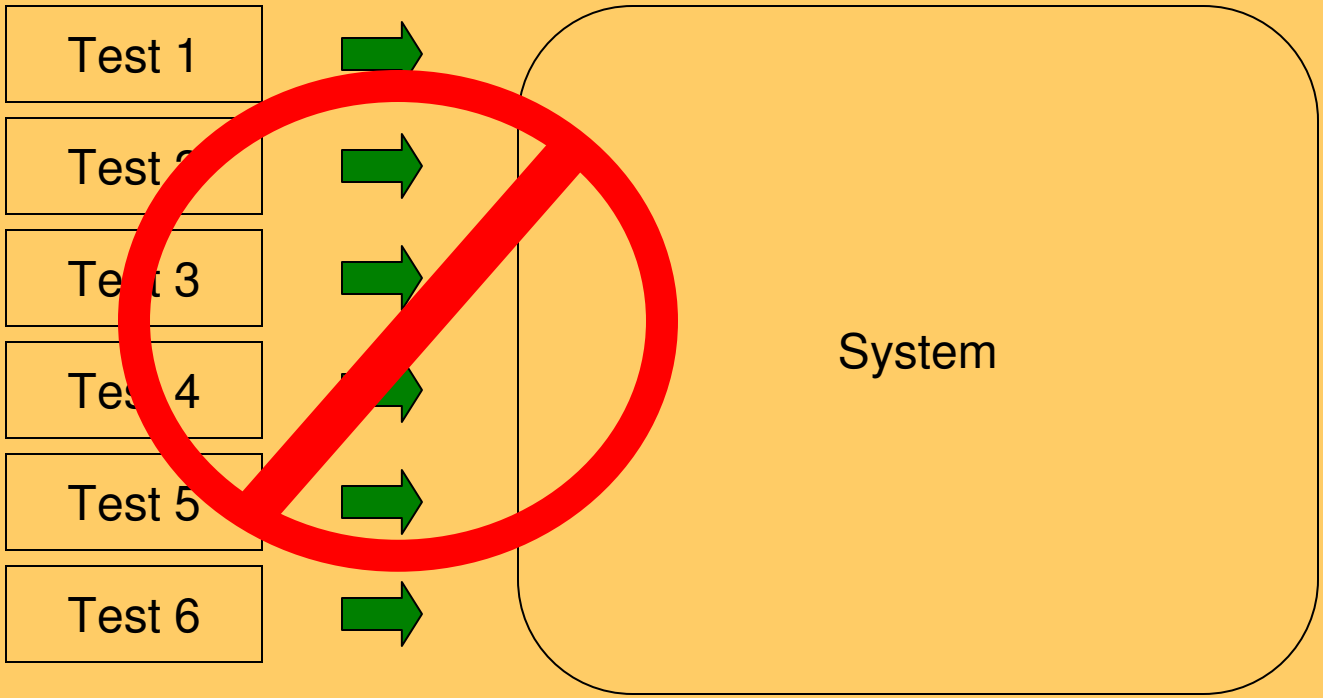
- Unit Testing ← TDD focuses here
  - Testing individual units (typically methods)
  - White/Clear-box testing performed by original programmer
- Integration and Functional Testing ← and may help here
  - Testing interactions of units and testing use cases
- Regression Testing ← and here
  - Testing previously tested components after changes
- Stress/Load/Performance Testing
  - How many transactions/users/events/… can the system handle?
- Acceptance Testing
  - Does the system do what the customer wants?

TDD Misconceptions

- There are many misconceptions about TDD
- They probably stem from the fact that the first word in TDD is “Test”
- TDD is **not about testing**,
  TDD is about **design**
  - Automated tests are just a nice side effect

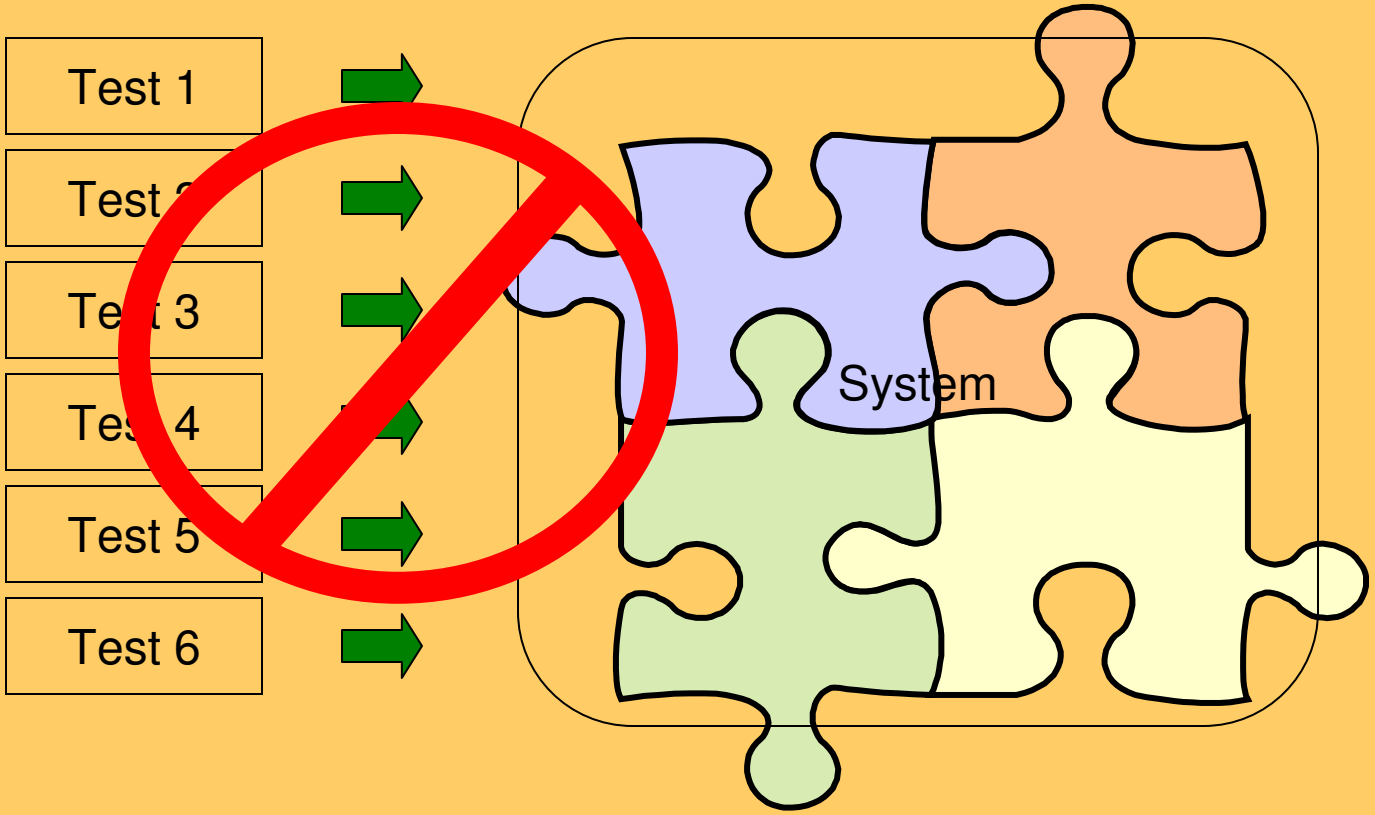
TDD Misconception #1

- TDD does not mean "write all the tests, then build a system that passes the tests"



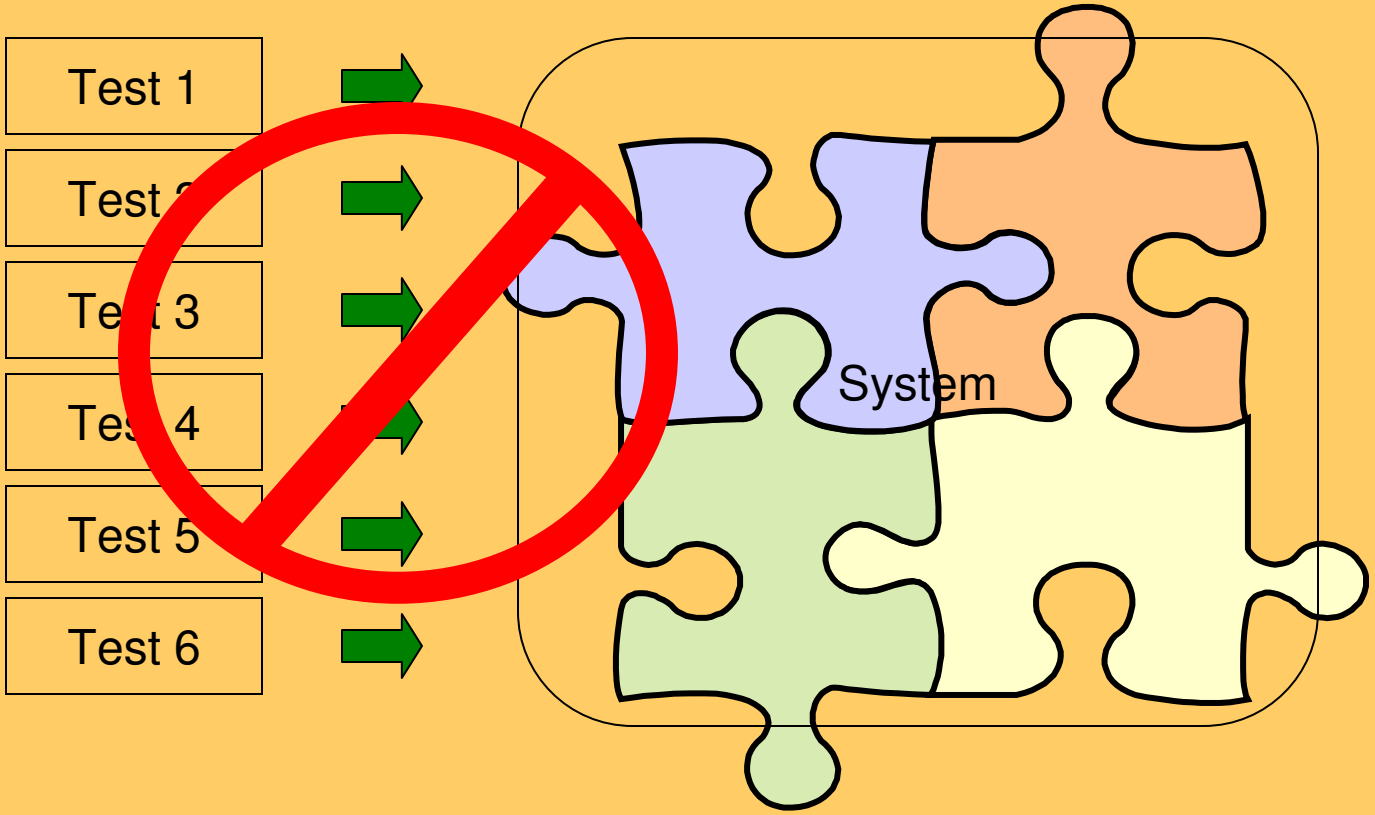
TDD Misconception #2

- TDD does not mean "write some of the tests, then build a system that passes the tests"



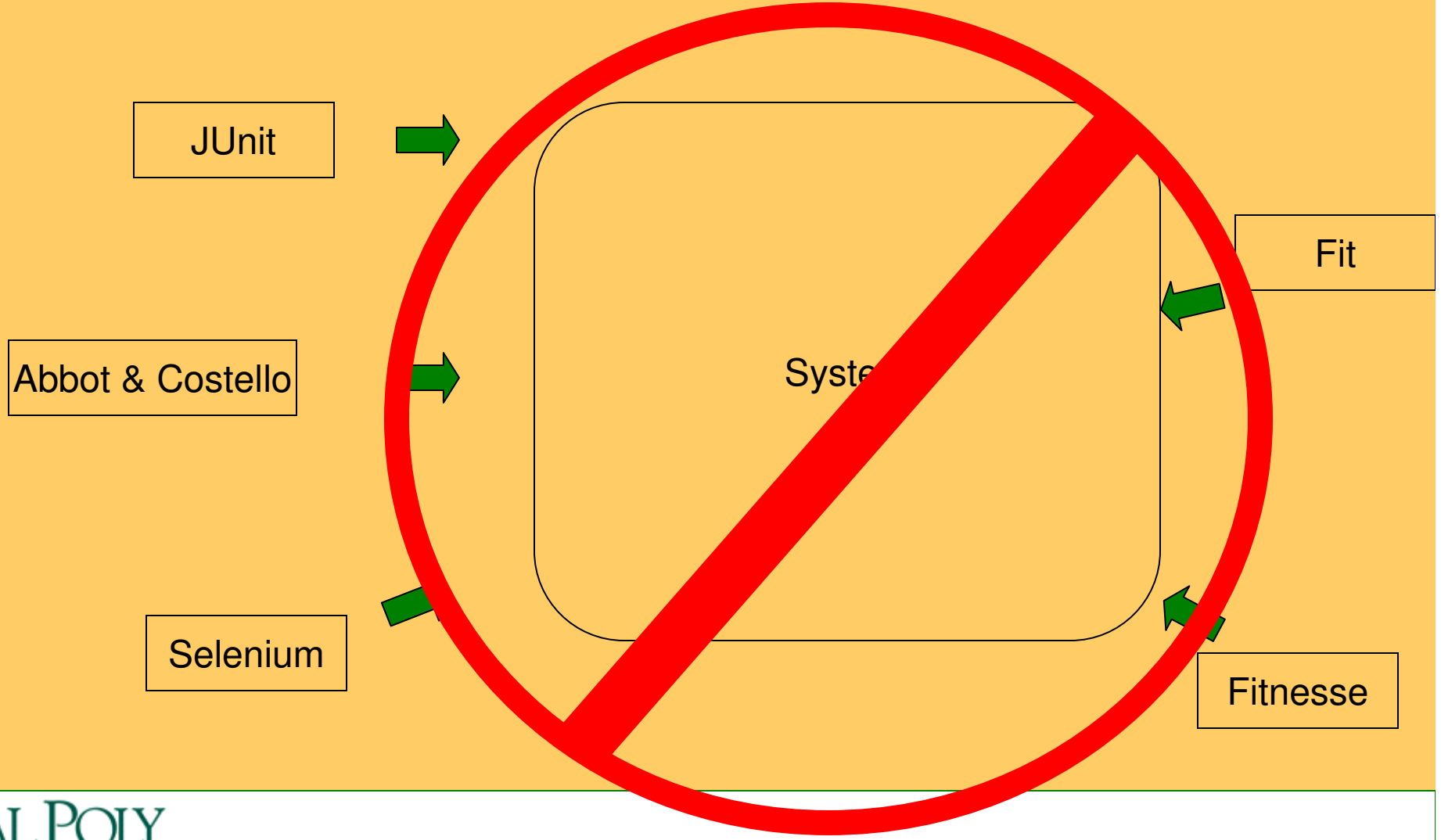
TDD Misconception #3

- TDD does not mean "write some of the code, then test it before going on"



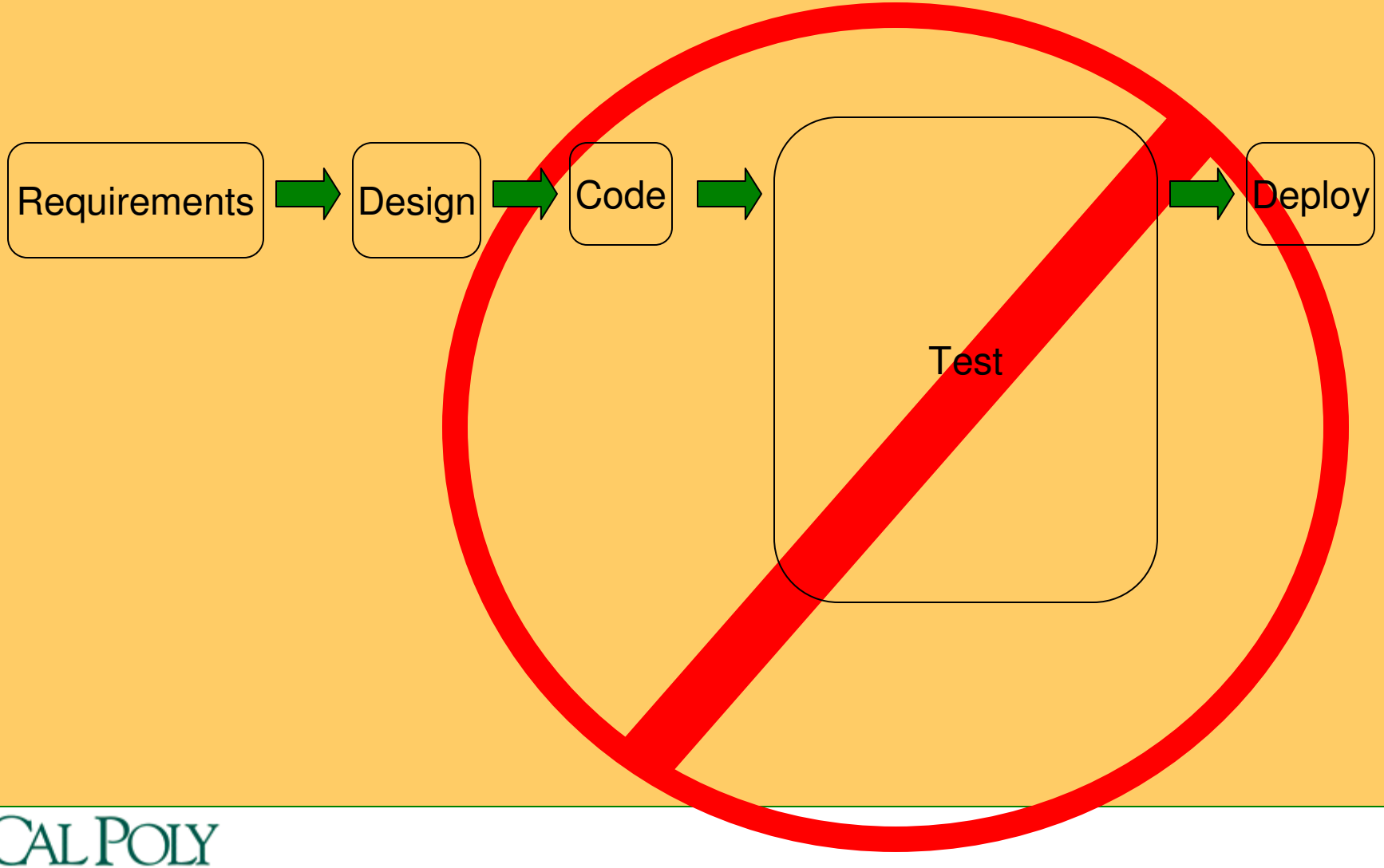
TDD Misconception #4

- TDD does not mean “do automated testing”



TDD Misconception #5

- TDD does not mean "do lots of testing"



TDD Misconception #6

- TDD does not mean “the TDD process”
- TDD is a *practice*

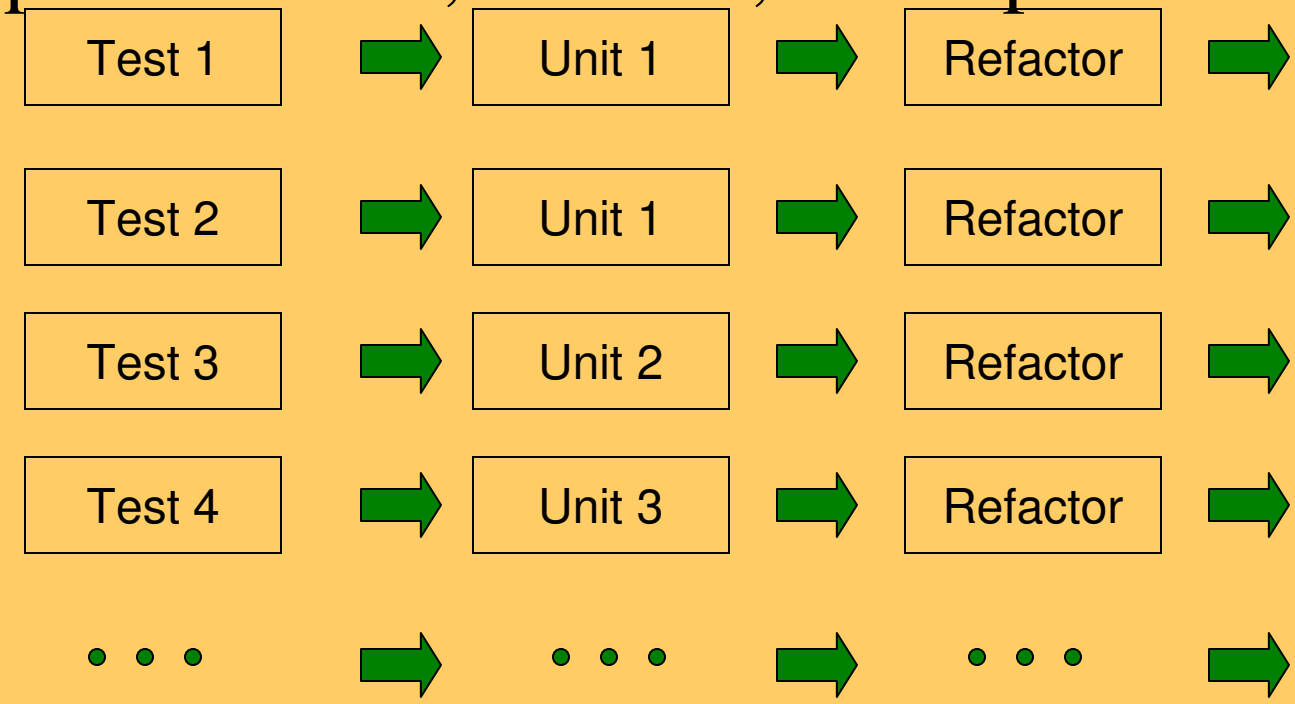
  (like pair programming, code reviews, and stand-up meetings)

  not a *process*

  (like waterfall, Scrum, XP, TSP)

TDD Clarified

- TDD means “write one test, write code to pass that test, refactor, and repeat”



Why Test-Driven Development?

- Everybody else is using TDD
  - OK, not a great reason, but can't ignore it
  - Examples:
    - MS Silverlight 2 Beta 1 included >2000 tests, boasting >80% coverage for Controls.Test[1]
    - IEEE Software dedicated a 2007 edition to TDD
    - Wikipedia lists xUnit frameworks for 55 languages
    - Testimonials from companies such as Google, Intuit, and Salesforce.com (see Agile 200x for more)
    - Steve McConnell included TDD among his top ten software advances of the last decade

1. http://blogs.msdn.com/sburke/archive/2008/03/05/silverlight-2-beta-1-controls-available-including-source-and-unit-tests.aspx

Why Test-Driven Development?

- Promising Claims:
  - Courage
    - Automated tests allow immediate feedback to the implications of refactorings and defect fixes
  - Better Designs
    - Focuses on the use of code, not the implementation
    - Encourages simple designs
      - Less coupling, more cohesion
  - Increased Test Coverage
  - Teamwork
    - Tests are a form of code documentation
    - Unit tests can be completed in parallel, unlike integration tests which require complete units
  - Fun and addictive
    - Become Test-Infected and keep your code clean

TDD Evidence: Productivity and External Quality

- May 2007 IEEE Software article summarized 9 industry studies and

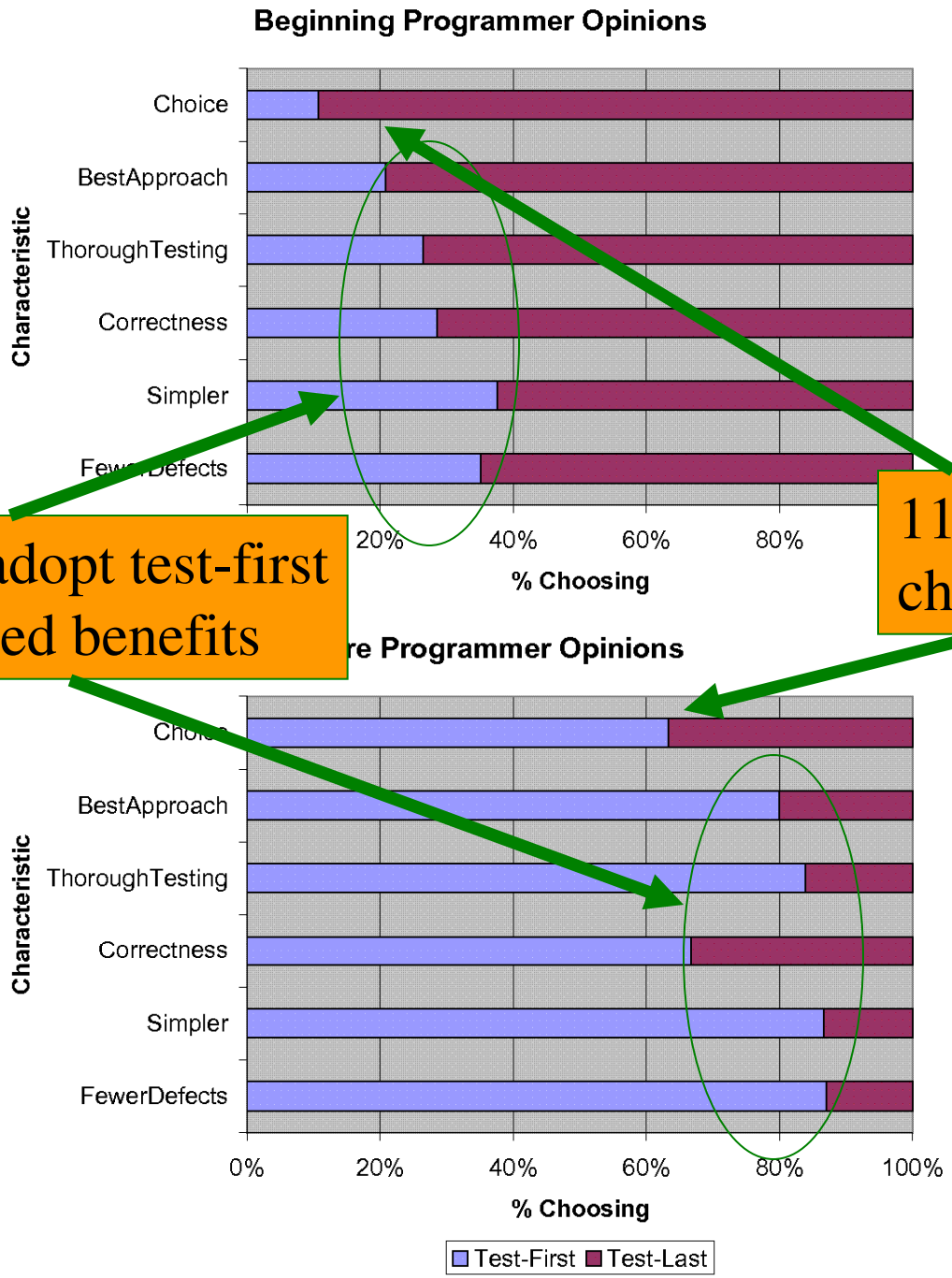
  9 academic studies
  - Industry study results
    - Most reported **increased effort**
      - Up to 100% on small projects, 5-19% on larger projects
    - Consistently reported **increased quality**
      - Up to 267% reduction in internal defect density
  - Academic studies a bit more mixed

TDD Evidence: Internal Quality

- March 2008 IEEE Software study (mine)
  - 15 projects over 5 years, 30+ KLOC
  - TDD produced **higher test coverage**
    - 30% line coverage, 78% branch coverage
  - TDD produced **smaller methods/classes**
    - 33% fewer LOC/method
    - 35% fewer methods/class
  - TDD produced **less complex methods/classes**
    - 54% lower average cyclomatic complexity (V(G))
    - 46% lower weighted methods per class (WMC)
  - Coupling and Cohesion results mixed

TDD Evidence: Opinions

- Study with ~150 students and industry professionals
  - Differences between early programmers and more mature programmers
  - TDD acceptance increased 47% with TDD experience
    - i.e. try it and you're more likely to like it



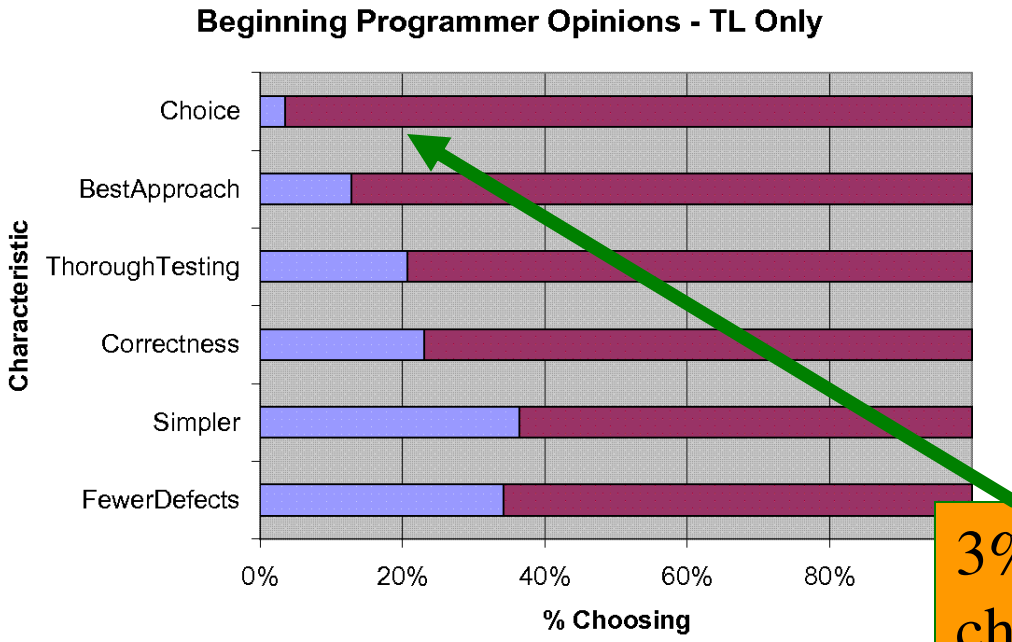
Reluctance to adopt test-first despite perceived benefits

11% vs 63% would choose test-first

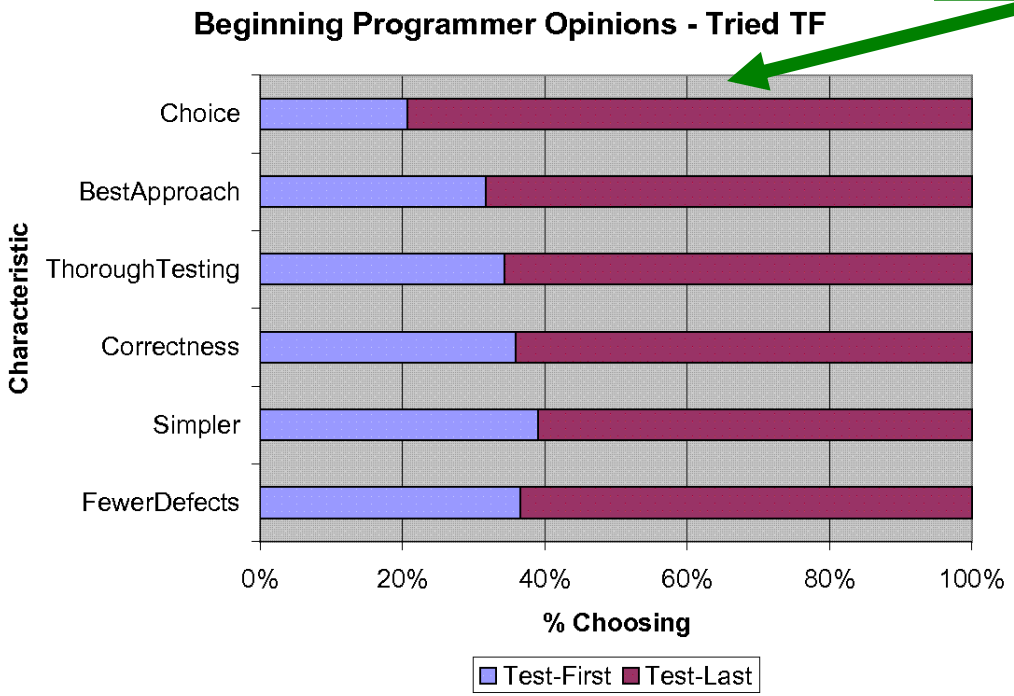
CAL POLY

Influence of TDD Experience

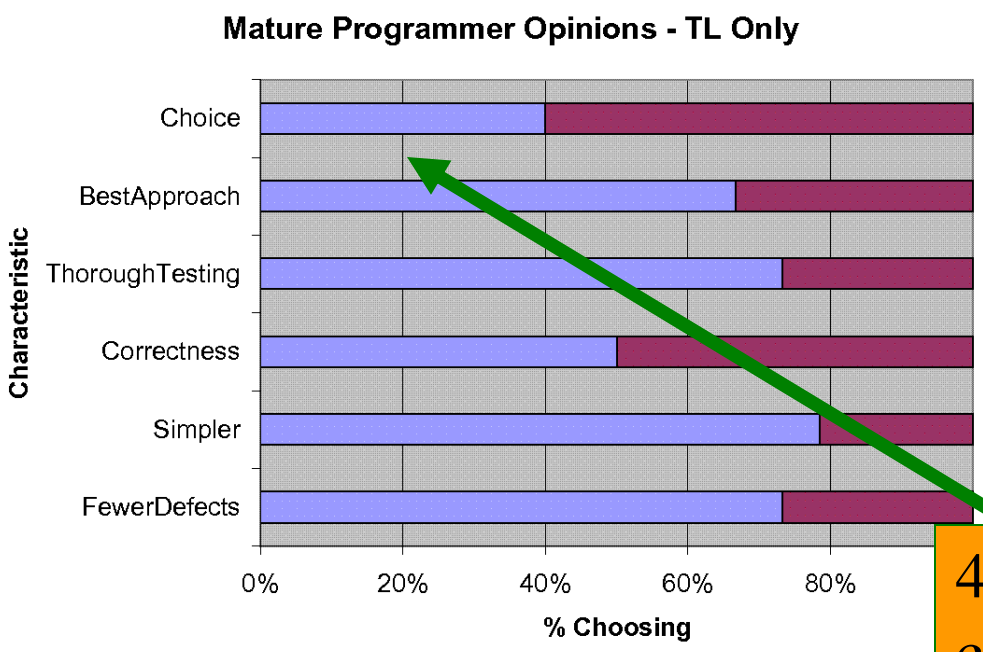
- Did using TDD influence programmer opinions regarding TDD perceptions?



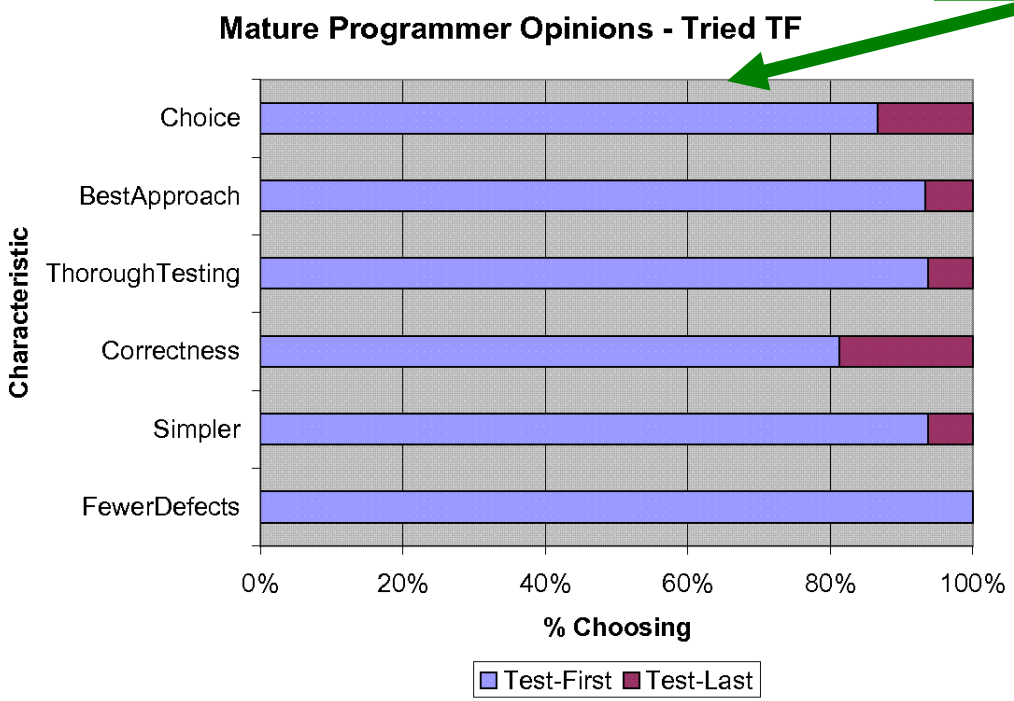
3% vs 21% would choose test-first



CAL POLY



40% vs 87% would choose test-first



CAL POLY