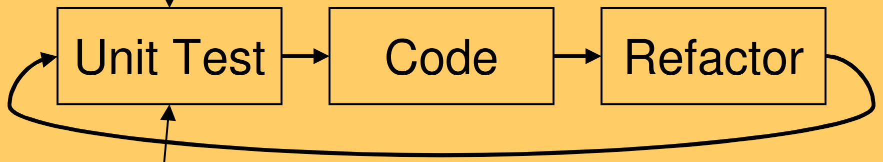
What is Test-Driven Development?

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

Unit tests are automated



Forces programmer to consider use of a method before implementation of the method

CAL POLY

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

This is the teeth of the test

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

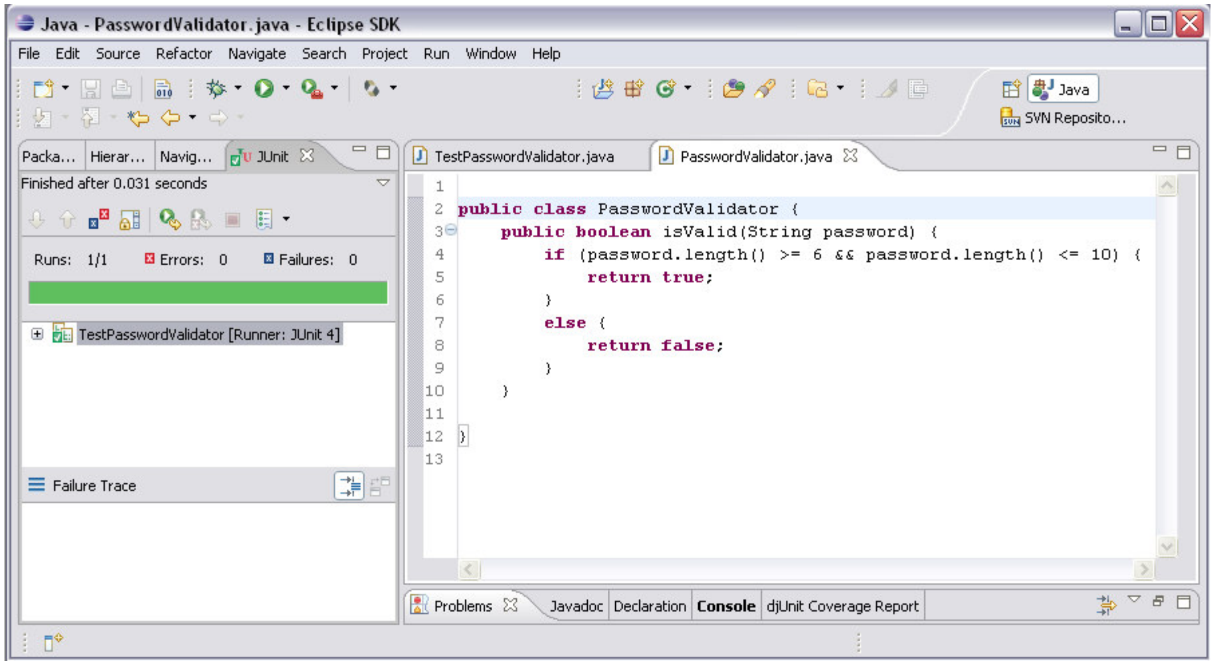
TDD Example: Write a test

```
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;
    }
  }
}
```


Java - PasswordValidator.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
SVN Reposito...
Packa... Hierar... Navig... JUnit
TestPasswordValidator.java
PasswordValidator.java
Problems Javadoc Declaration Console djUnit Coverage Report

```
Finished after 0.031 seconds
Runs: 1/1    Errors: 0    Failures: 0
TestPasswordValidator [Runner: JUnit 4]
Failure Trace
```

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

}
```

CAL POLY

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?

CAL POLY

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"));
  }
}
```

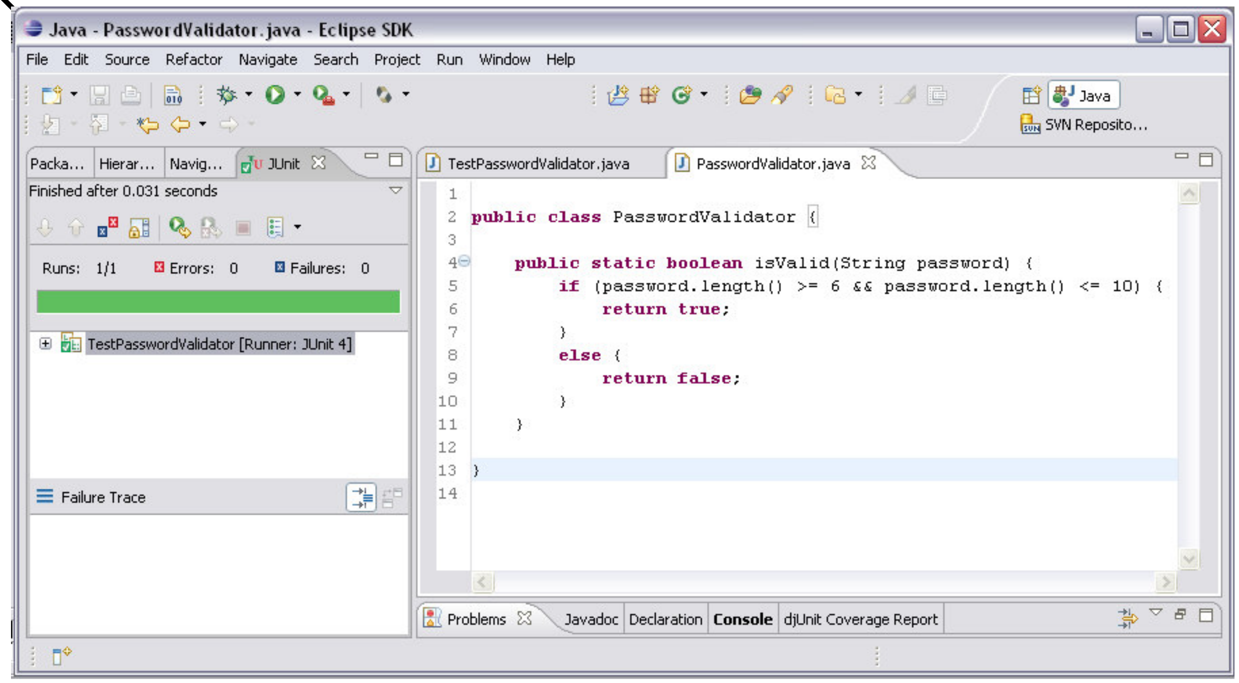
Design decision: static method

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;
    }
  }
}
```


Java - PasswordValidator.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
SVN Reposito...
Packa... Hierar... Navig... JUnit
TestPasswordValidator.java
PasswordValidator.java
Failure Trace
Problems Javadoc Declaration Console djUnit Coverage Report

```
Finished after 0.031 seconds
Runs: 1/1    Errors: 0    Failures: 0
TestPasswordValidator [Runner: JUnit 4]
```

```

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

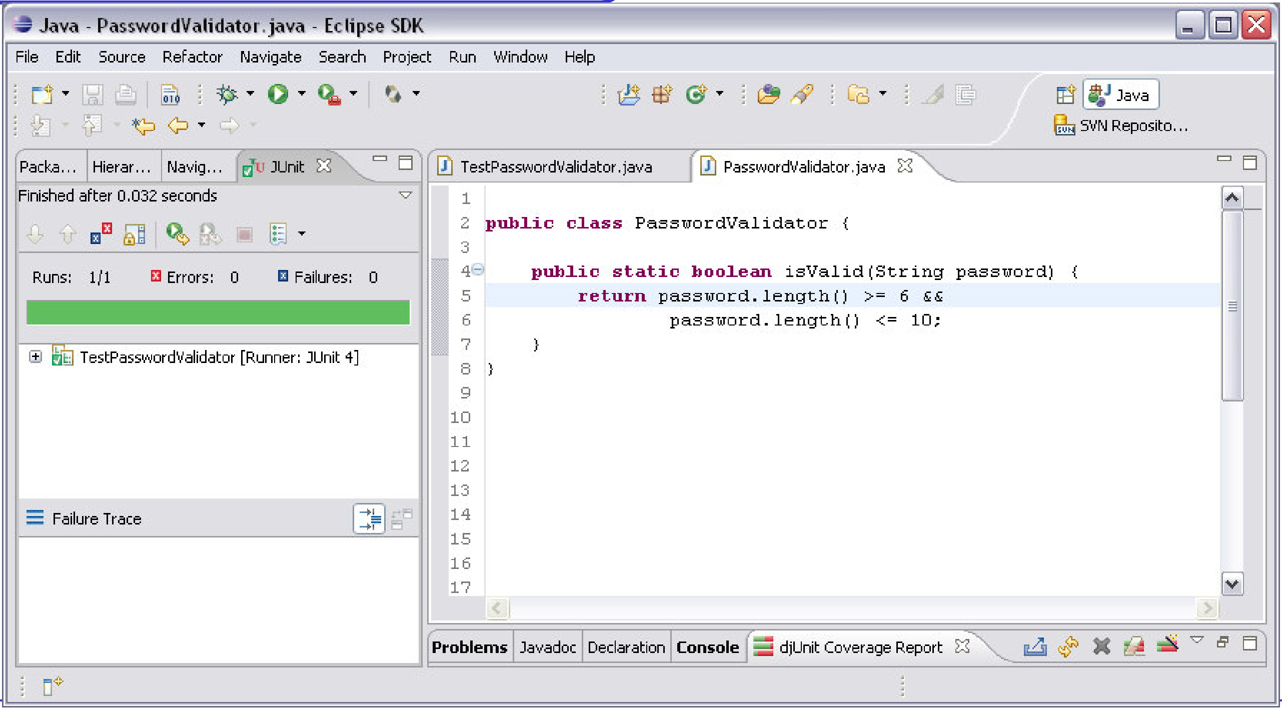
```
public class PasswordValidator {
  public static boolean isValid(String password) {
    if (password.length() >= 6 && password.length() <= 10) {
      return true;
    }
    else {
      return false;
    }
  }
}
```

Can we simplify this?

CAL POLY

TDD Example: Refactoring #1

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


Java - PasswordValidator.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
SVN Reposito...
Packa... Hierar... Navig... JUnit
TestPasswordValidator.java
PasswordValidator.java
Problems Javadoc Declaration Console djUnit Coverage Report
Failure Trace

Finished after 0.032 seconds

Runs: 1/1 Errors: 0 Failures: 0

TestPasswordValidator [Runner: JUnit 4]

```

public class PasswordValidator {

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

CAL POLY

TDD Example: Refactoring #1

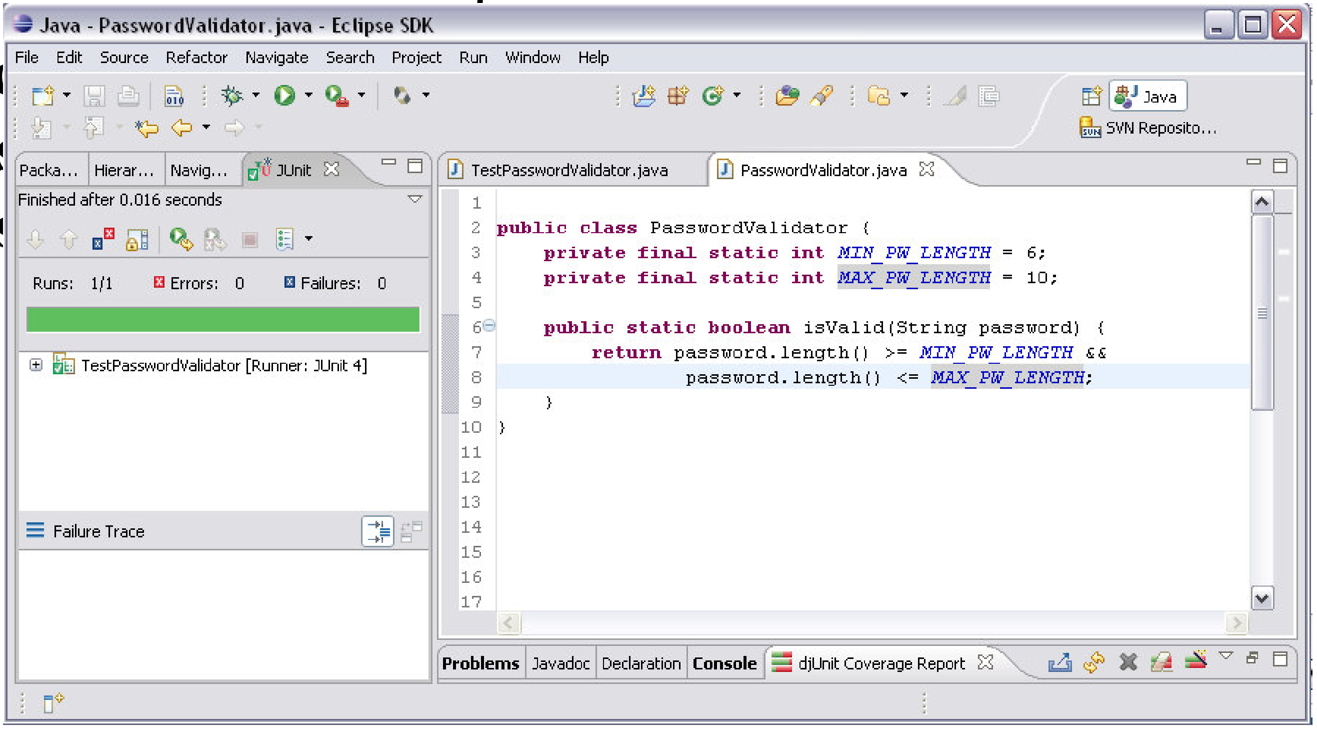
```
public class PasswordValidator {
  public static boolean isValid(String password) {
    return password.length() >= 6 &&
            password.length() <= 10;
  }
}
```

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

CAL POLY

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 pa
              pa
    }
 }
```



Java - PasswordValidator.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
SVN Reposito...
Packa... Hierar... Navig... JUnit
TestPasswordValidator.java
PasswordValidator.java
Failure Trace
Problems Javadoc Declaration Console djUnit Coverage Report

Finished after 0.016 seconds

Runs: 1/1 Errors: 0 Failures: 0

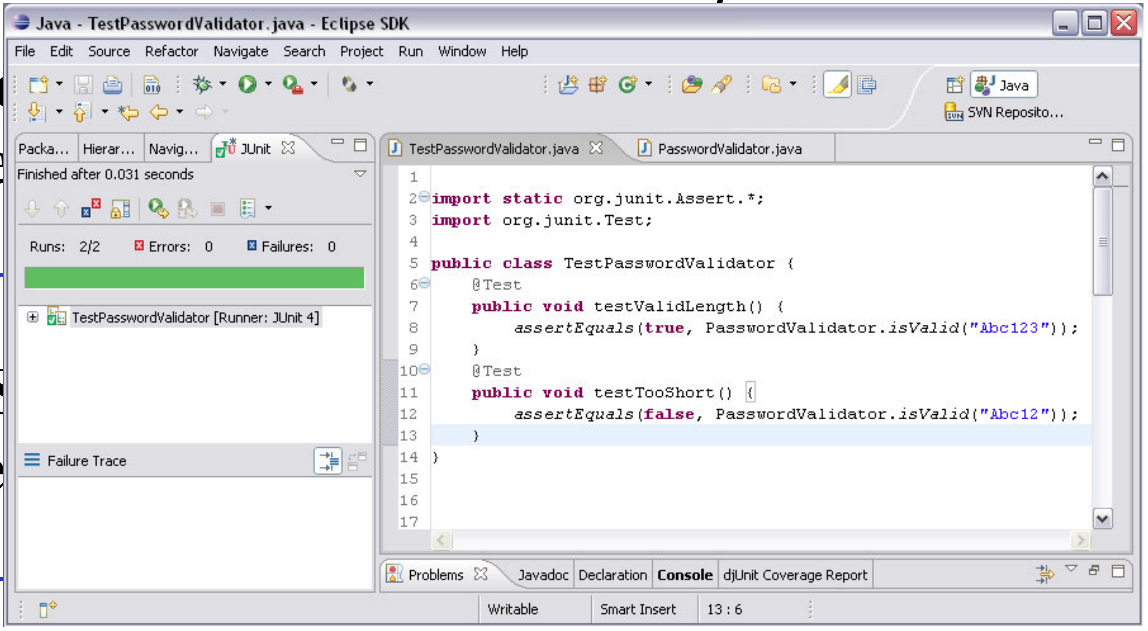
TestPasswordValidator [Runner: JUnit 4]

```
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: Write another test

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

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

No design decisions; just unit testing


Java - TestPasswordValidator.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
SVN Reposito...
Packa... Hierar... Navig... JUnit
TestPasswordValidator.java
PasswordValidator.java
Problems Javadoc Declaration Console djUnit Coverage Report

Finished after 0.031 seconds

Runs: 2/2 Errors: 0 Failures: 0

TestPasswordValidator [Runner: JUnit 4]

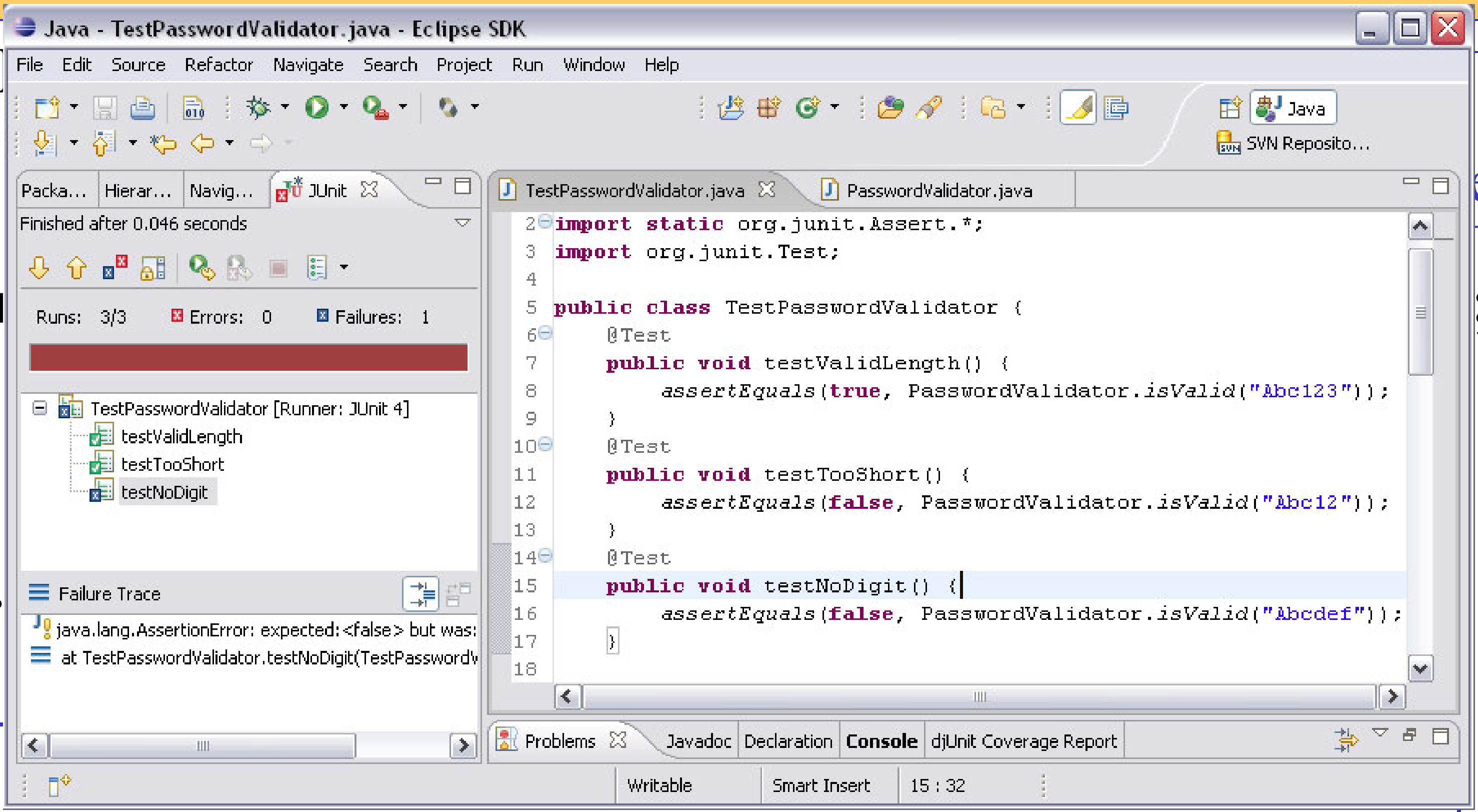
Failure Trace

```

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

public class TestPasswordValidator {
    @Test
    public void testValidLength() {
        assertEquals(true, PasswordValidator.isValid("Abc123"));
    }
    @Test
    public void testTooShort() {
        assertEquals(false, PasswordValidator.isValid("Abc12"));
    }
}
```

Writable Smart Insert 13 : 6

TDD Example: Write another test

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



Java - TestPasswordValidator.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help
Java
SVN Reposito...
Packa... Hierar... Navig... JUnit
TestPasswordValidator.java
PasswordValidator.java
Failure Trace
Problems Javadoc Declaration Console djUnit Coverage Report

Finished after 0.046 seconds

Runs: 3/3 Errors: 0 Failures: 1

TestPasswordValidator [Runner: JUnit 4]
- testValidLength
- testTooShort
- testNoDigit

```
java.lang.AssertionError: expected:<false> but was:
at TestPasswordValidator.testNoDigit(TestPasswordV
```

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

public class TestPasswordValidator {
    @Test
    public void testValidLength() {
        assertEquals(true, PasswordValidator.isValid("Abc123"));
    }
    @Test
    public void testTooShort() {
        assertEquals(false, PasswordValidator.isValid("Abc12"));
    }
    @Test
    public void testNoDigit() {
        assertEquals(false, PasswordValidator.isValid("Abcdef"));
    }

```

Writable Smart Insert 15 : 32

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;

Check for a digit

public class PasswordValidator {

privat

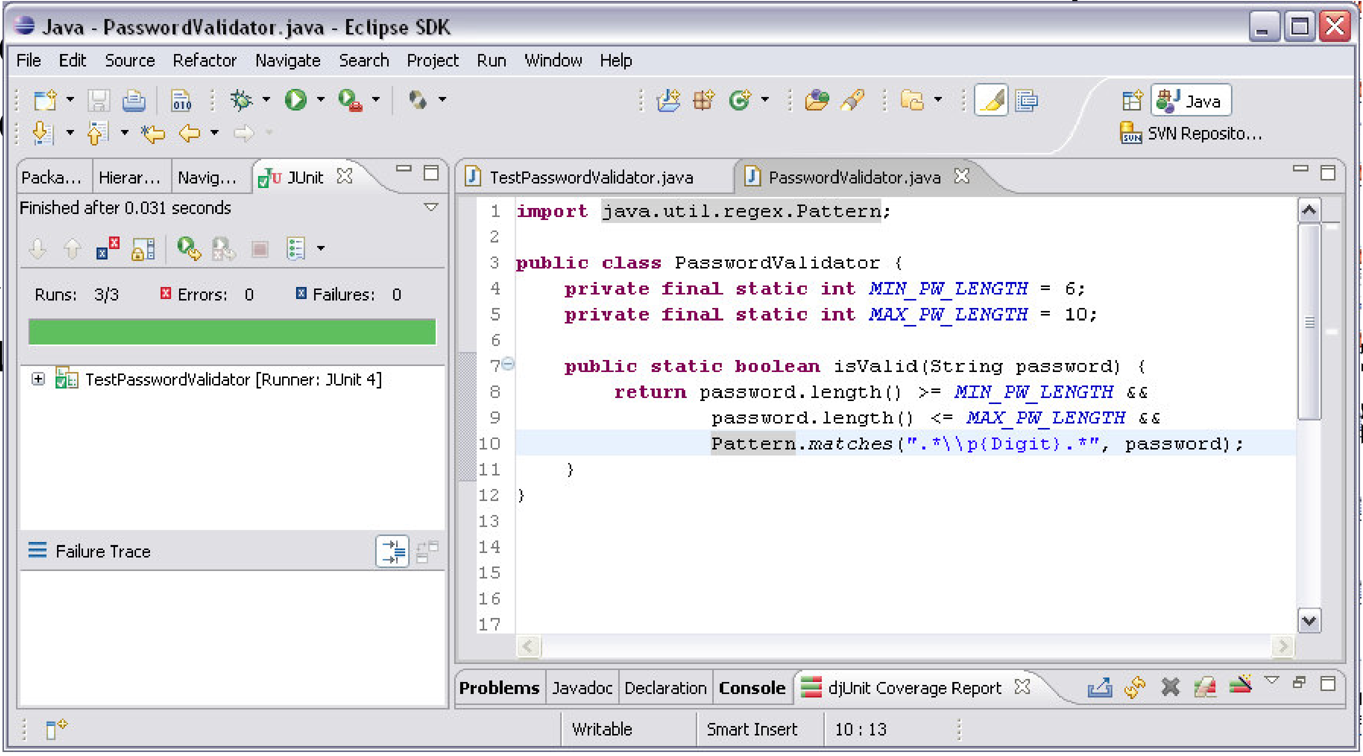
privat

public

retu

}

}


Java - PasswordValidator.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
Java
SVN Reposito...
Packa... Hierar... Navig... JUnit
TestPasswordValidator.java
PasswordValidator.java
Failure Trace
Problems Javadoc Declaration Console djUnit Coverage Report
Writable Smart Insert 10 : 13

Finished after 0.031 seconds

Runs: 3/3 Errors: 0 Failures: 0

TestPasswordValidator [Runner: JUnit 4]

```
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);
    }
}
```

TDD Example: Refactor

Extract methods for readability

```
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);
        }
    }
```

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

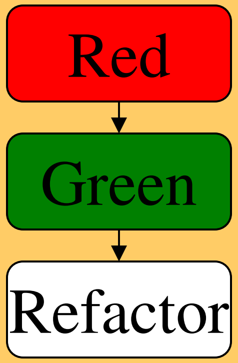
Test-Driven Development

- **Definition**[1]
  - Test-driven Development (TDD) is a programming practice that instructs developers to write new code only if an automated test has failed, and to eliminate duplication. The goal of TDD is "clean code that works."

1. "JUnit in Action" Massol and Husted.

- **The TDD Two-Step**[2]
  - Write a failing automatic test before writing new code
  - Eliminate duplication

- **The TDD Cycle**[2]
  - Write a test
  - Make it run
  - Make it right



2. "Test-Driven Development By Example" Beck.

Some Types of Testing

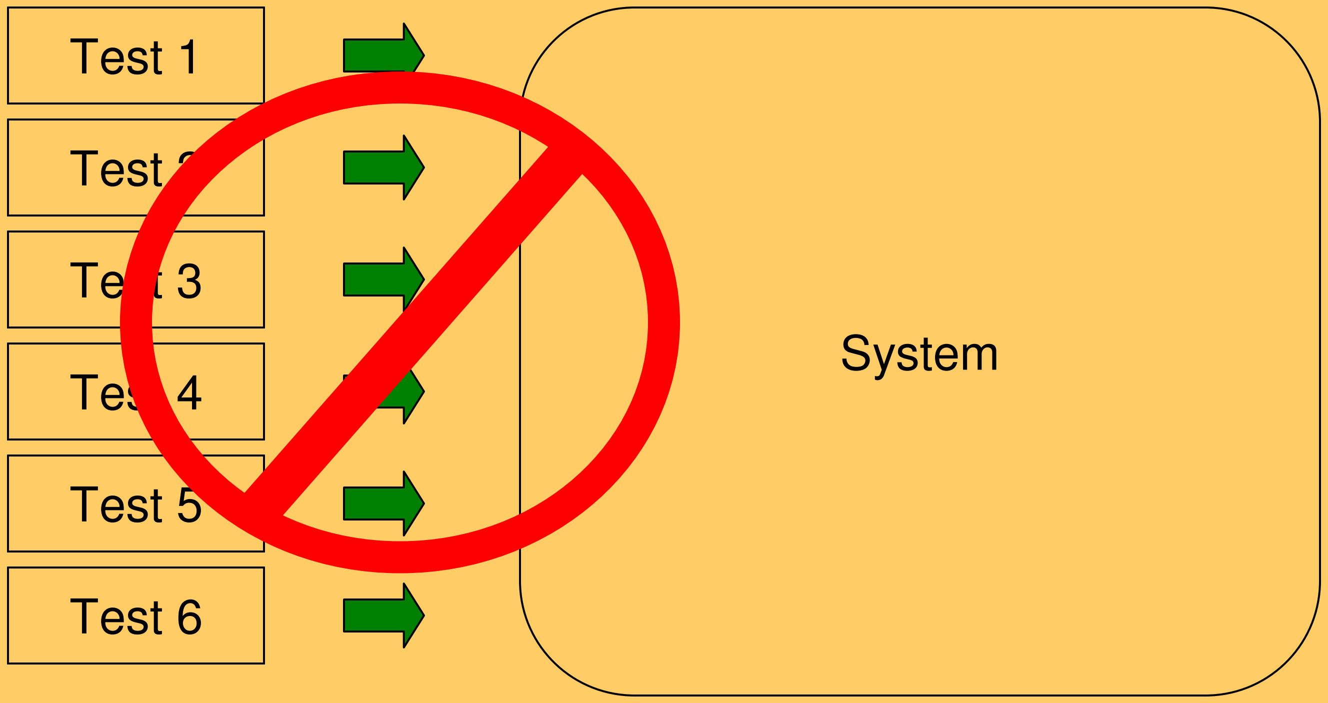
- Unit Testing ← TDD focuses here
  - Testing individual units (typically methods)
  - White/Clear-box testing performed by original programmer
- Integration and Functional Testing ← and may help here
  - Testing interactions of units and testing use cases
- Regression Testing ← and here
  - Testing 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

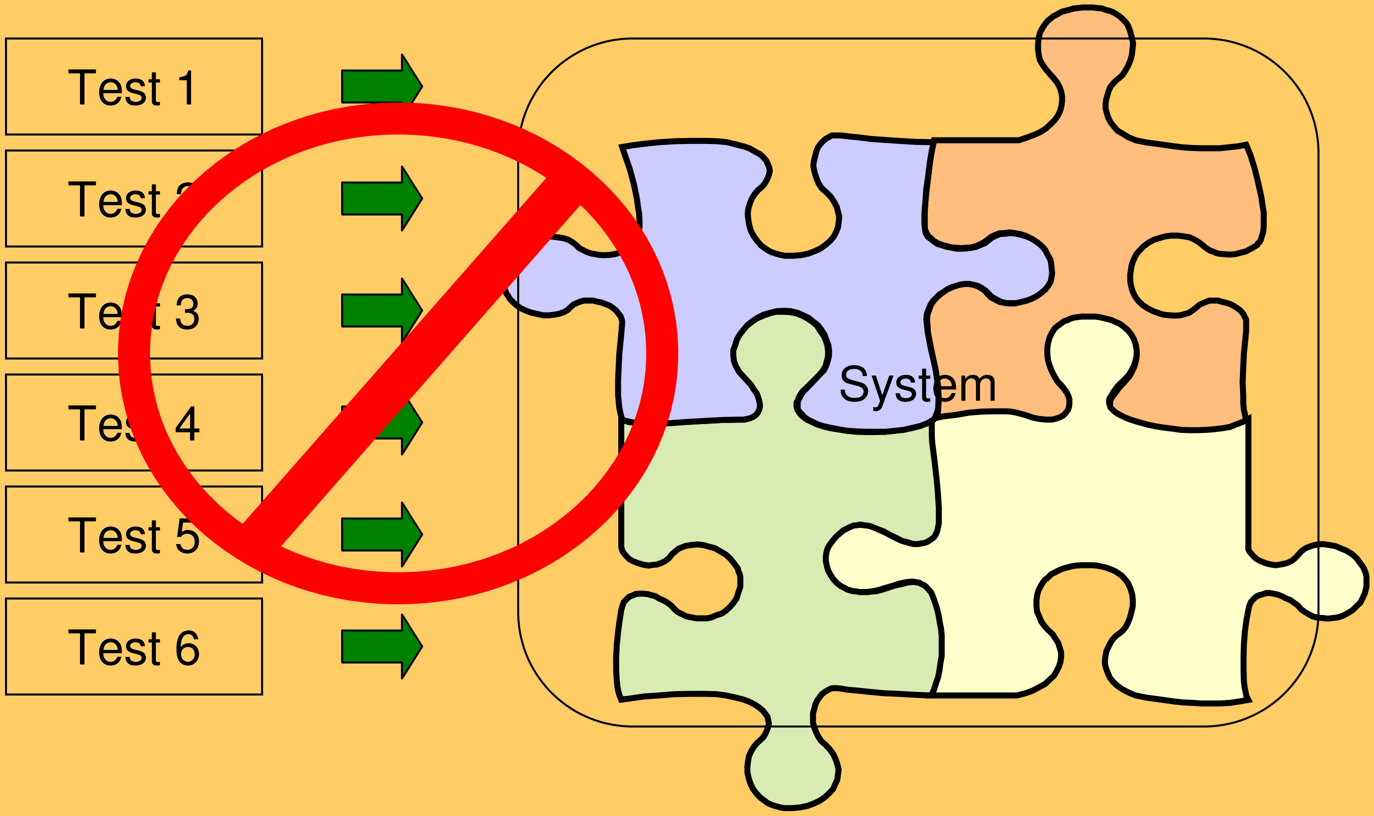
TDD Misconception #1

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



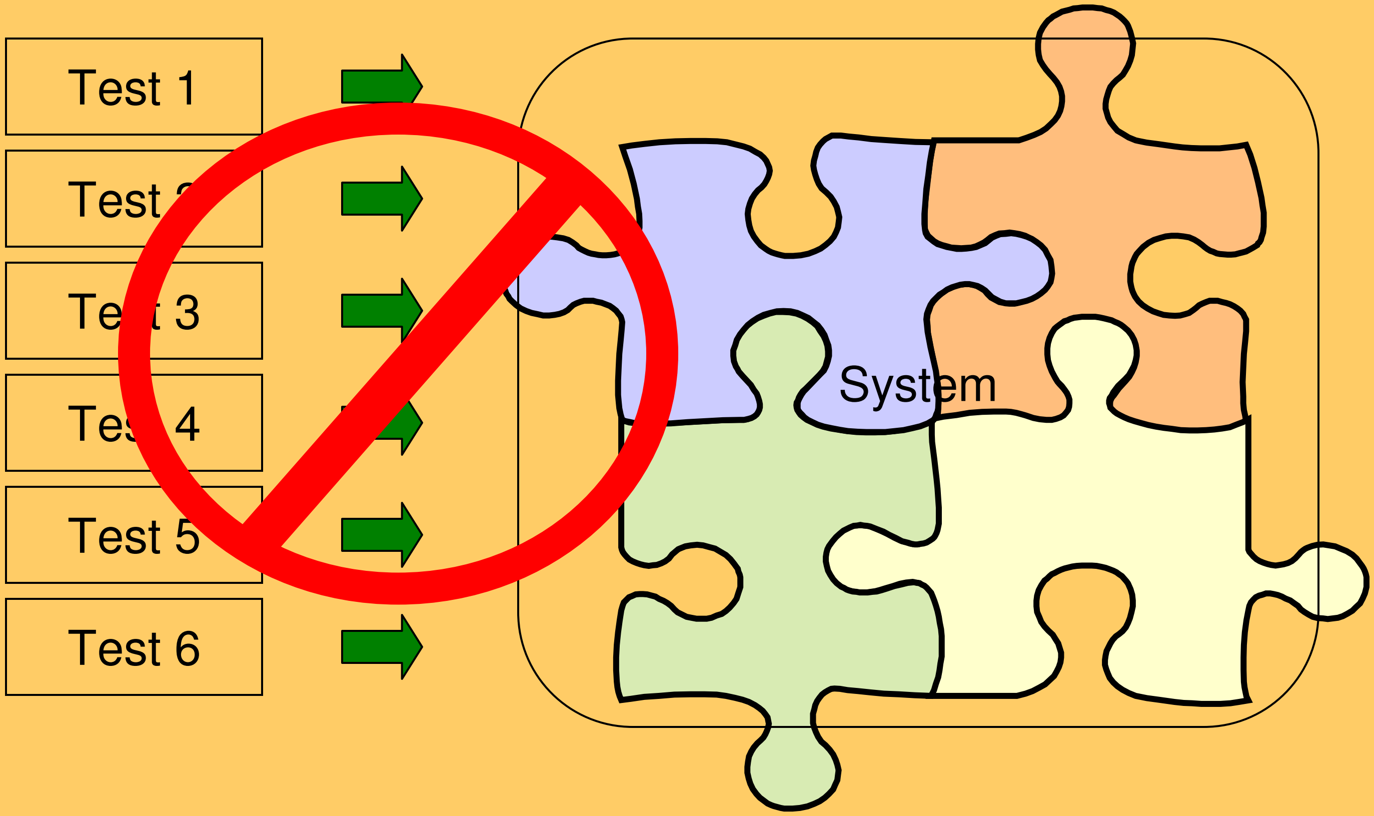
TDD Misconception #2

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



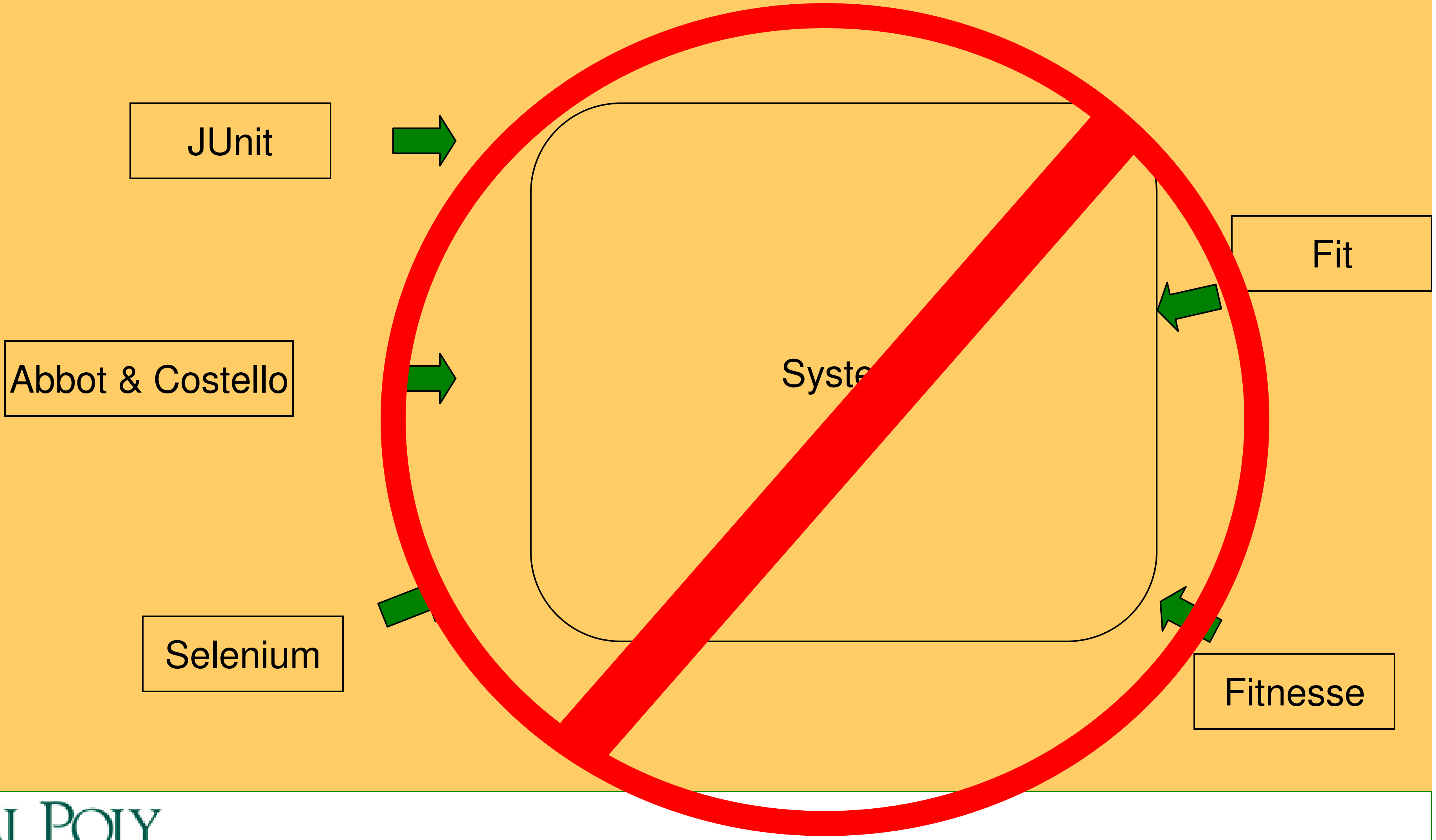
TDD Misconception #3

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



TDD Misconception #4

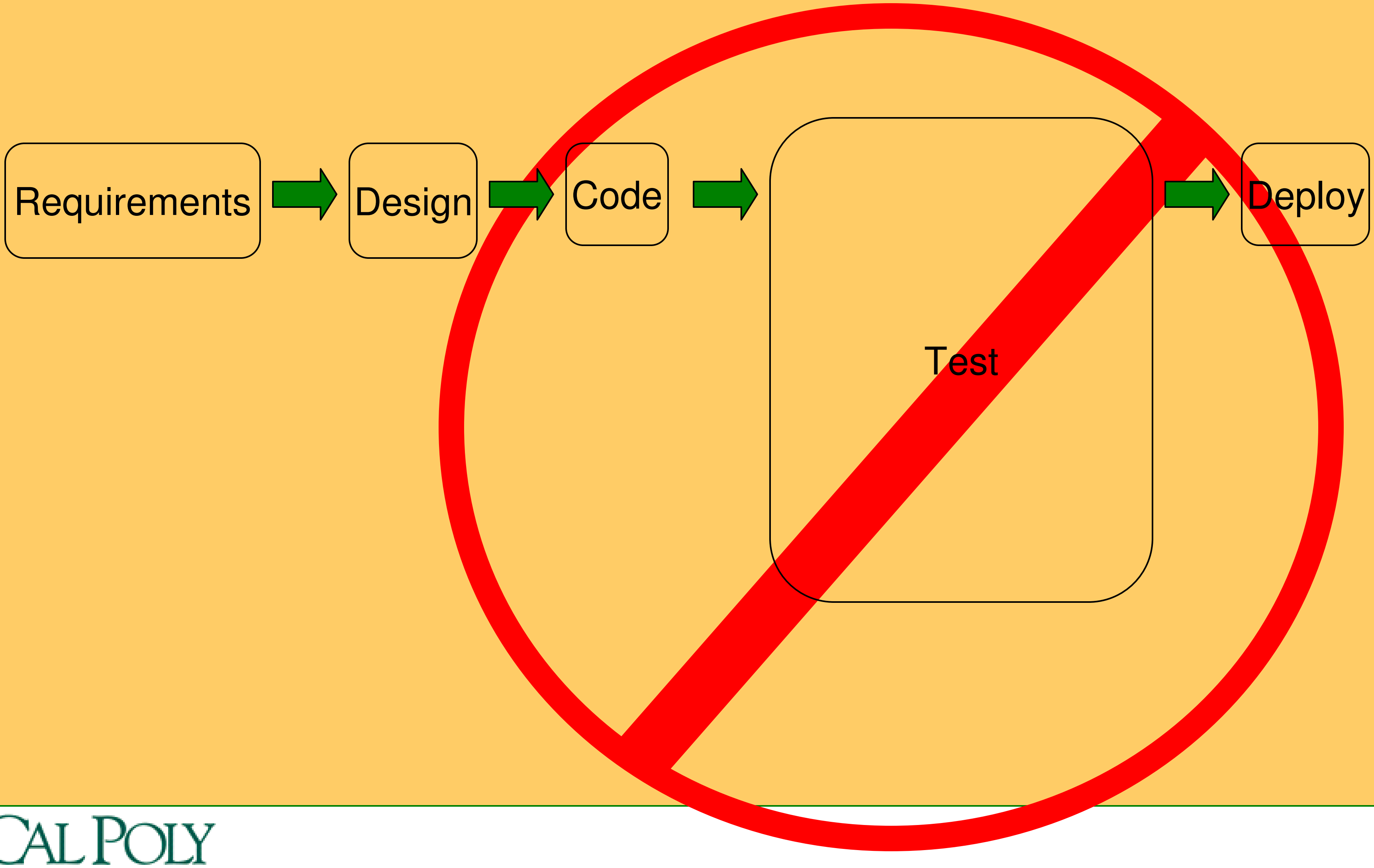
- TDD does not mean "do automated testing"



CAL POLY

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*

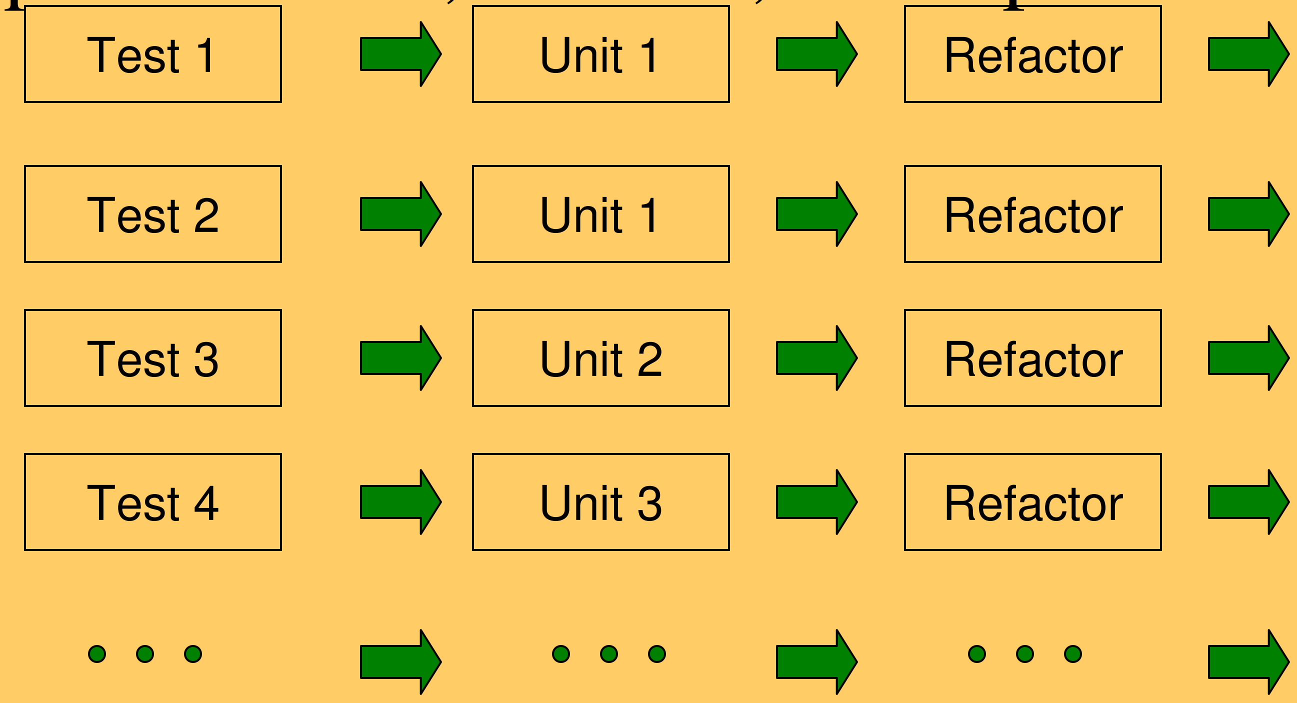
  (like pair programming, code reviews, and stand-up meetings)

  not a *process*

  (like waterfall, Scrum, XP, TSP)

TDD Clarified

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


Test 1
Unit 1
Refactor
Test 2
Unit 1
Refactor
Test 3
Unit 2
Refactor
Test 4
Unit 3
Refactor
• • •
• • •
• • •