TESTS

Jeremie Dequidt

OVERVIEW

- Introduction
  - What and how
  - Test and development process
  - Different types of test
- Test driven development

TEST TO PREVENT…

- … an error introduced by the developer
  - An error is an inappropriate or erroneous decision done by a developer that leads to a default introduction
- … a default in the system
  - A default is an imperfection in one of the system aspects that contributes or may potentially contribute to one or several failure occurrence…
  - Sometimes several defaults are required to provoke a failure.
- … a failure during the execution
  - A failure is an unacceptable behaviour of a system.
  - The failure frequency reflects the reliability.

© Dr Benoît Combemale

DEFINITION

Testing is a manual or automated process that aims to check that a system satisfies properties requested by its specifications, or to detect differences between results produced by the system and those expected by the specifications

IEEE-STD729, 1983

© Dr Benoît Combemale

TEST PRINCIPLE

- Trying to discover bugs
- Trying to see if it works

© Dr Benoît Combemale

TEST PRINCIPLE

- ➤ Trying to discover bugs
- ➤ Trying to see if it works

Learning
- Why it is done
- What it should do
- How it is done

Designing
Having an overview
Executing
Analysing

What is to see?
What should we look at?
What is visible?
What are we looking for?
How to look at it?

What should be working?
Identifying an error
Diagnosing an error
Categorizing these errors

© Dr Benoît Combemale

WHAT ARE WE TESTING?

- Which properties?
  - Functionality
  - Security / integrity
  - Usability
  - Coherence
  - Maintenability
  - Efficiency
  - Robustness
  - Etc.

© Dr Benoît Combemale

HOW DO WE TEST?

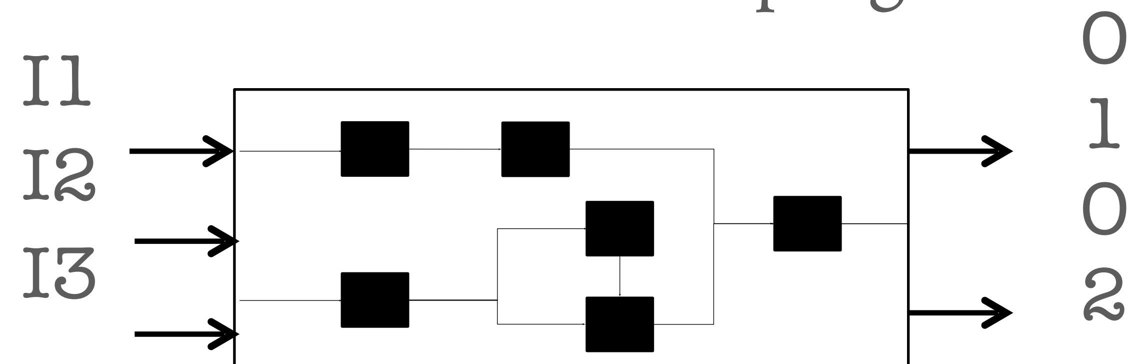
- Static test
  - Reading / reviewing code
  - Automatic analysis (checking properties, coding rules)
- Dynamic test
  - Executing the program with input data and observing the behaviour

© Prof David Janzen

HOW DO WE TEST?

- Functional test (black box testing)
  - Use the program functionalities description

- Structural test (white box testing)
  - Use the internal structure of the program



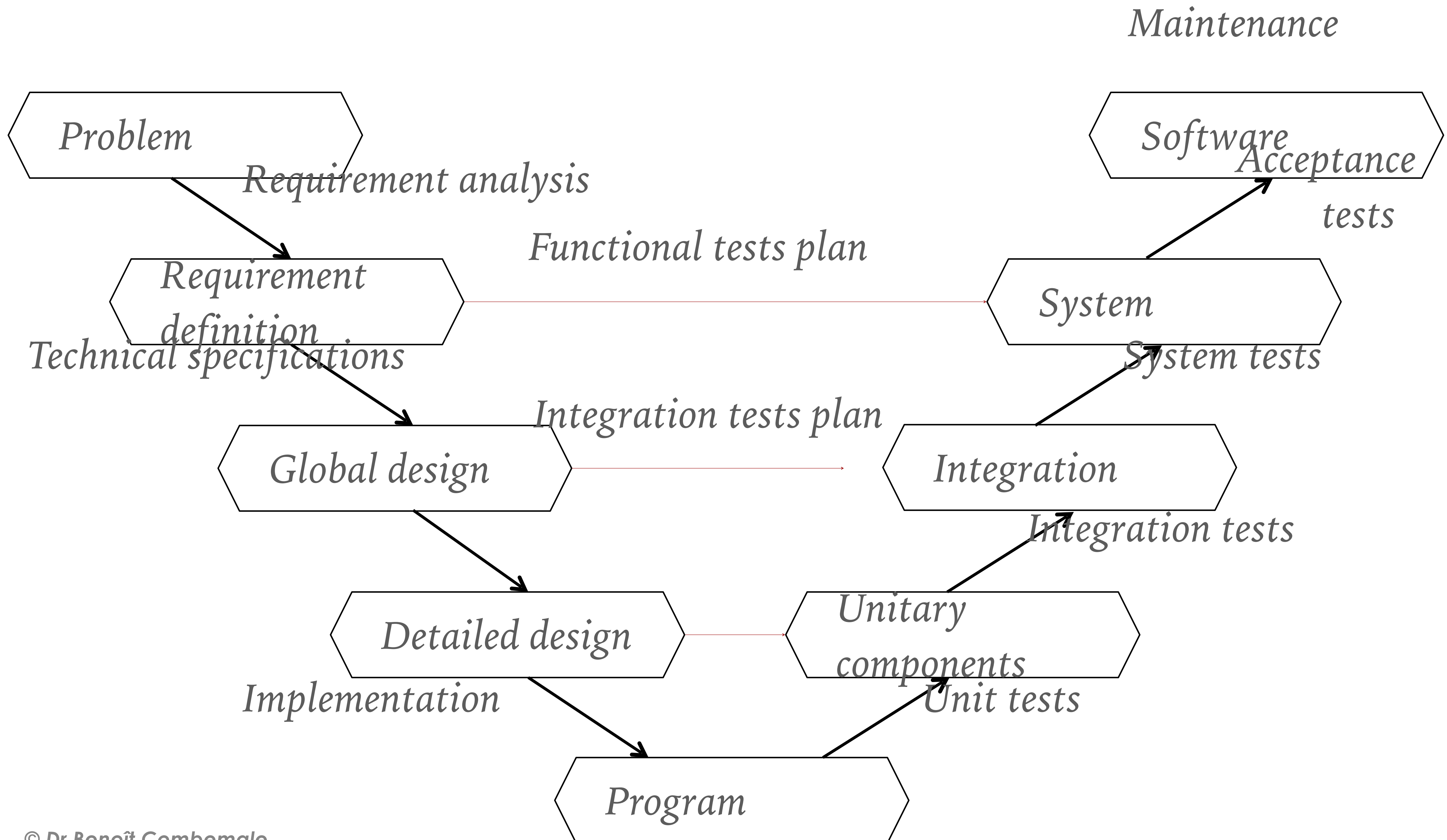
© Dr Benoît Combemale

WITH WHAT WE TEST?

- A specification: expressing what is expected from the system
  - Coding rules
  - Technical specifications (natural language)
  - Comments in code
  - Contracts on operations (as in Eiffel)
  - A UML model
  - A formal specification (automata, B model, …)

© Dr Benoît Combemale

TEST HIERARCHY



© Dr Benoît Combemale

SOME TYPES OF TESTING

- Unit Testing
  - Testing individual units (typically methods)
  - White/Clear-box testing performed by original programmer
- Integration and Functional Testing
  - Testing interactions of units and testing use cases
- Regression Testing
  - 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?

© Prof David Janzen

UNIT TEST

- Validate a module independently from the others
- Intensively validate the unitary functions
- Are the unit enough specified?
- Is the code readable, maintainable?

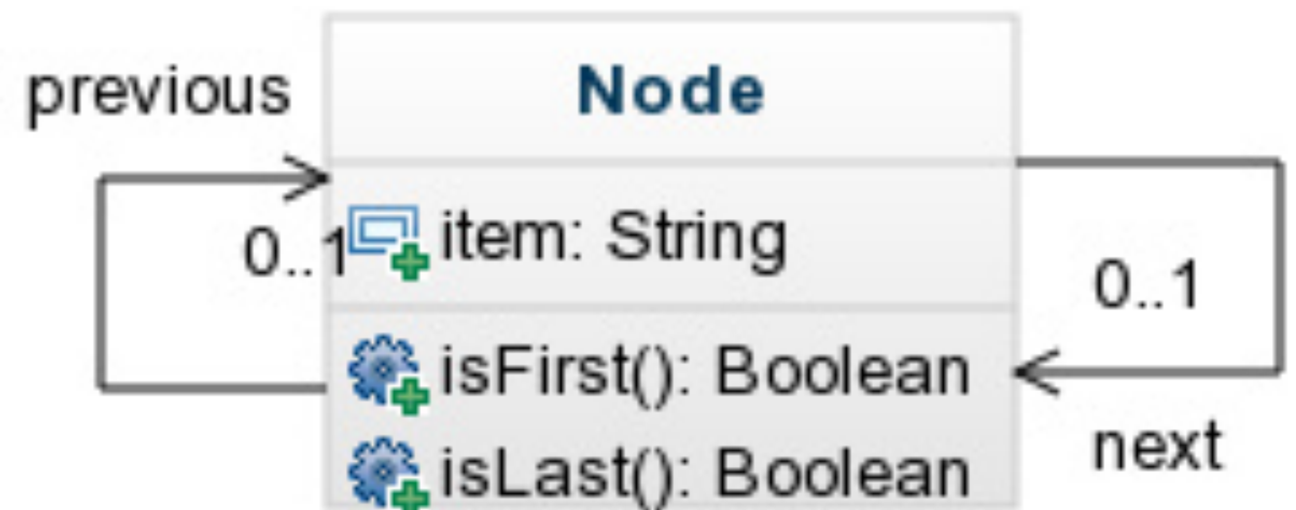
© Dr Benoît Combemale

UNIT TEST

- For a procedural language
  - Unit of test = procedure

```
void Ouvrir (char *nom, Compte *C, float S, float D)
{
  C->titulaire = AlloueEtCopieNomTitulaire(nom);
  (*C).montant = S ;
  (*C).seuil = D ;
  (*C).etat = DEJA_OUVERT ;
  (*C).histoire.nbop = 0;
  EnregistrerOperation(C);
  EcrireTexte("Ouverture du compte numero ");
  EcrireEntier(NumeroCourant+1);
}
```

- For a OO language
  - Unit of test = class



© Dr Benoît Combemale

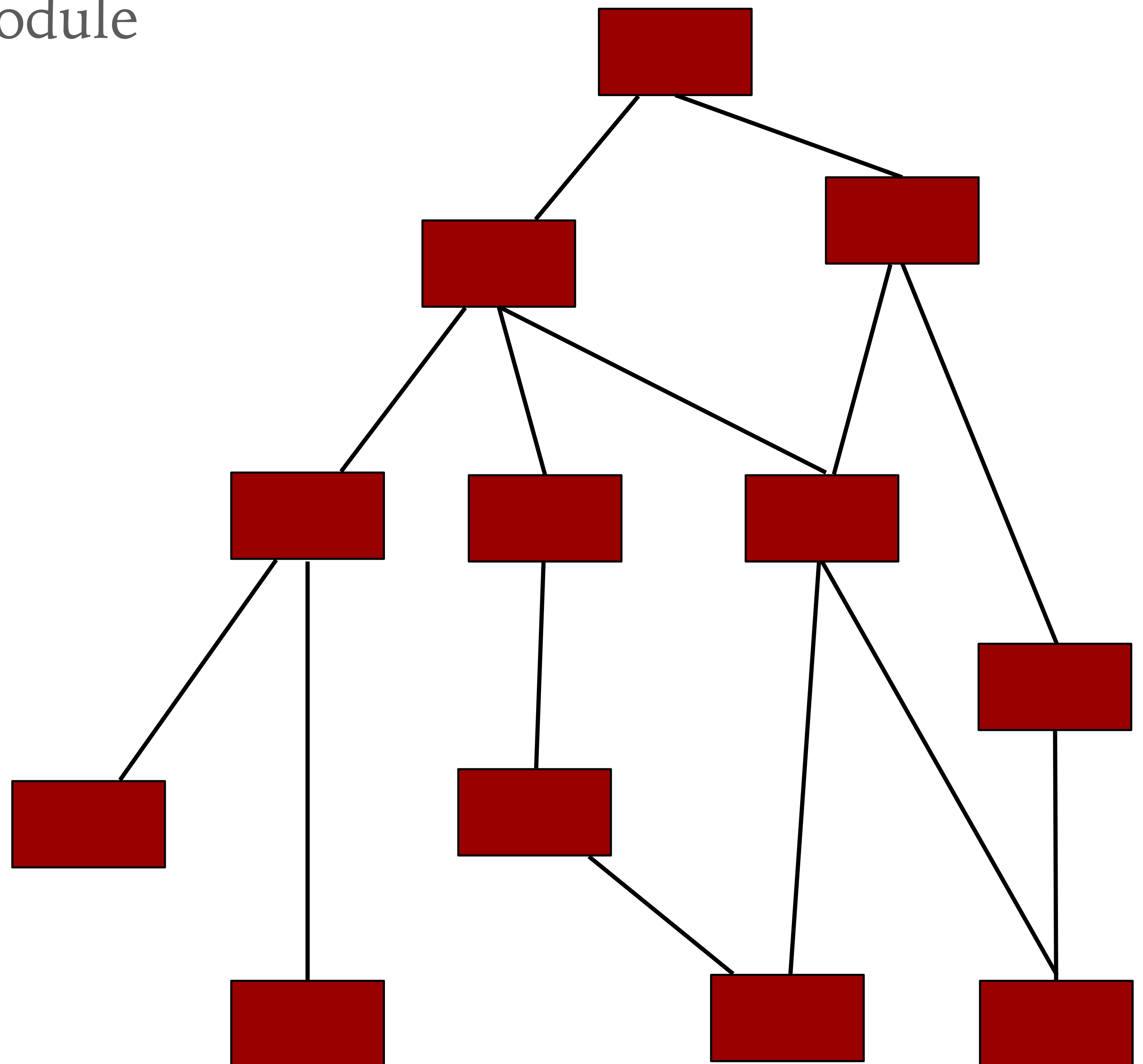
INTEGRATION TEST

- Find an order to test and integrate the modules of the system

© Dr Benoît Combemale

INTEGRATION TEST

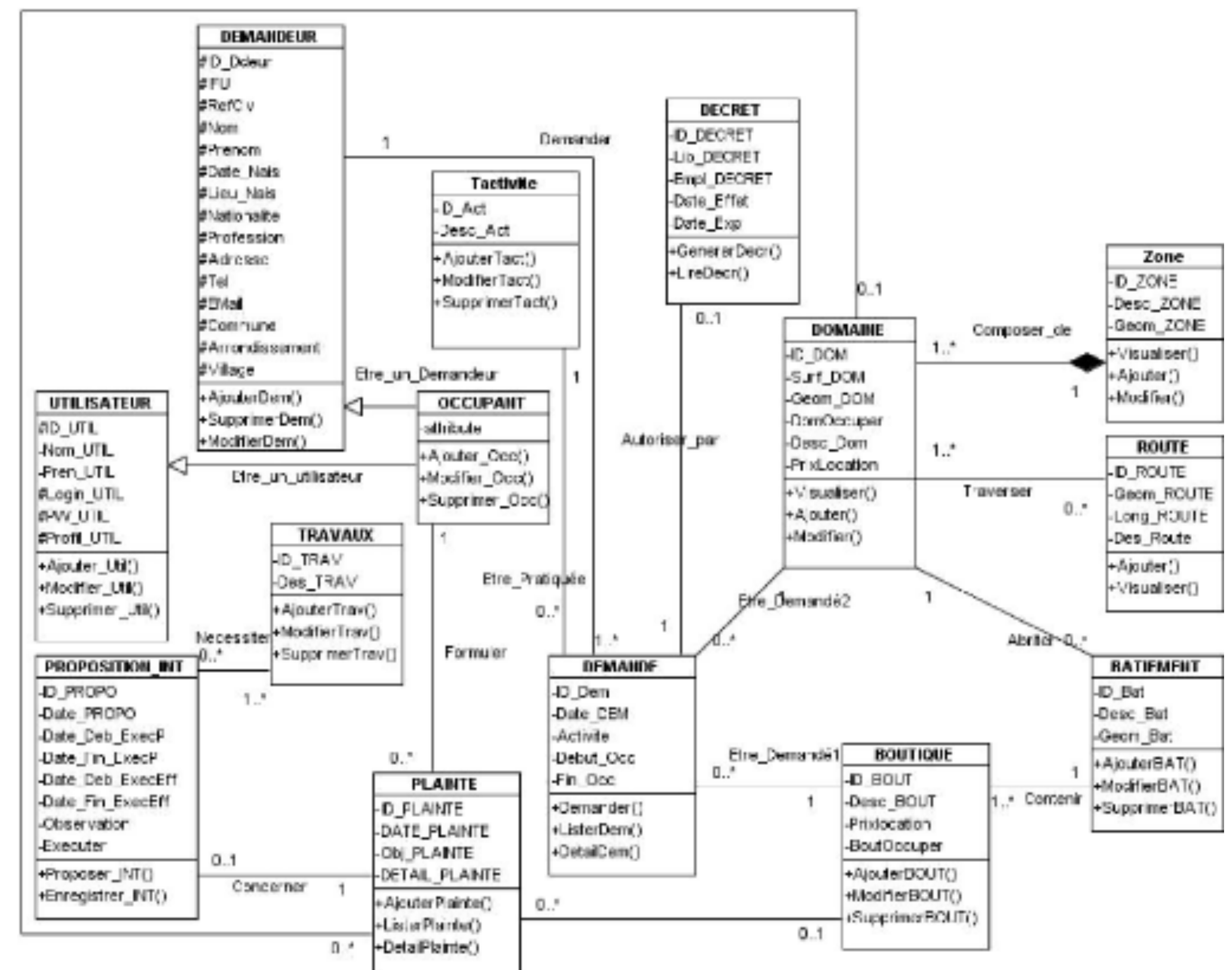
- Simple case:
  - There is no cycle in the module dependencies
  - Dependencies form a tree we can simply integrate modules from the bottom and up



© Dr Benoît Combemale

INTEGRATION TEST

- More complex case:
  - There are cycles in the module dependencies
  - It is really frequent in object systems
  - Heuristics have to be found to find an integration order



© Dr Benoît Combemale

SYSTEM TEST

- Validate the whole system
  - The proposed functionalities
  - The system quality
    - Charge, ergonomy, security, etc.
  - From the GUI

© Dr Benoît Combemale

NON REGRESSION TEST

- Check that the modifications made have not introduced new errors
  - Check that what worked still works
- In the software maintenance phase
  - After refactoring, add/removal of functionalities
- After a fault correction

© Dr Benoît Combemale

SOME VIDEOS TO GO FURTHER

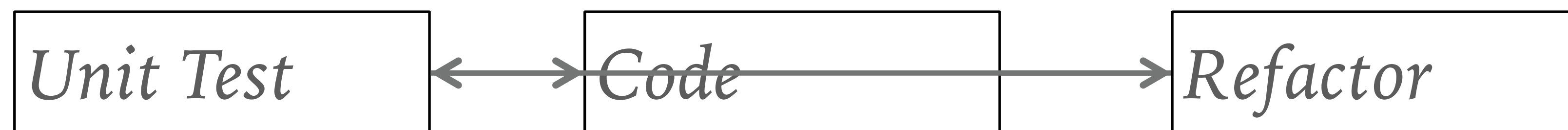
- https://www.youtube.com/watch?v=hBCaoN421Qs in French

© Prof David Janzen

TEST DRIVEN DEVELOPMENT

WHAT IS TEST-DRIVEN DEVELOPMENT?

- TDD is a design (and testing) approach involving short, rapid iterations of



© Prof David Janzen

ADVANTAGES

- Write tests first => program is used even before it exists
- Reduce design conception
- Increase the self-confidence of the programmer during code revision
- Joint design of the program and a set of non-regression tests
- Estimate the progress of project development (velocity)

© Dr Gauthier Picard

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

© Prof David Janzen

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!

© Prof David Janzen

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

© Prof David Janzen

JUNIT TEST INSTRUCTIONS

Instruction	Description
fail(String)	Make fail the test method
assertTrue(true)	Always true
assertsEquals(expected, actual)	Test if the values are the same
assertsEquals(expected, actual,	Proximity test with tolerence
assertNull(object)	Check if the object is null
assertNotNull(object)	Check if the object is not null
assertSame(expected, actual)	Check if the variables refer the same
assertNotSame(expected, actual)	Check if the variables do not refer the
assertTrue(boolean condition)	Check if the boolean condition is true

© Dr Gauthier Picard

JUNIT TEST ANNOTATIONS

Annotation	Description
@Test	Test method
@Before	Method executed *before each test*
@After	Method executed *after each test*
@BeforeClass	Method executed *before the first test*
@AfterClass	Method executed *after the last test*
@Ignore	Method not run as test
@Test(expected=XXException)	Specify the expected exception

Annotations have to be put before the methods of the unitary test class

© Dr Gauthier Picard

PHPUNIT TEST INSTRUCTIONS

Instruction	Description
fail(String)	⚠ Does not exist in Php
assertTrue(true)	Always true
assertsEquals(expected, actual)	Test if the values are the same
assertsEquals(expected, actual,	Proximity test with tolerence
assertNull(object)	Check if the object is null
assertNotNull(object)	⚠ Does not exist in Php
assertSame(expected, actual)	Check if the variables refer the same
assertNotSame(expected, actual)	Check if the variables do not refer the
assertTrue(boolean condition)	Check if the boolean condition is true

© Dr Gauthier Picard

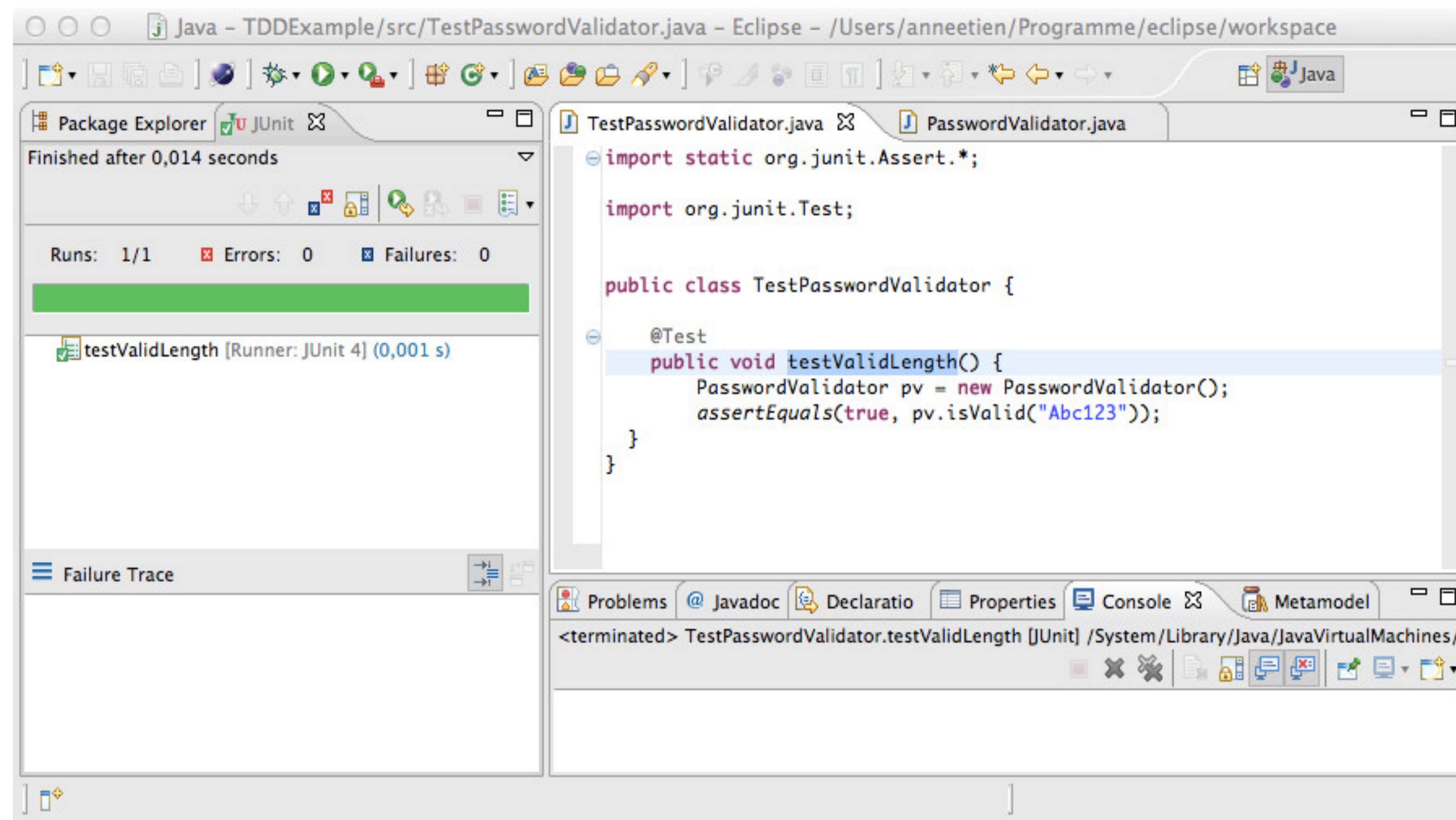
PHP UNIT ANNOTATIONS

Annotation	Description
@test	Test method
@before	Method executed *before each test*
@after	Method executed *after each test*
@beforeClass	Static method executed *before each test*
@afterClass	Static method executed *after each test*
@expectedException	Specify the expected exception

Annotations have to be put before the methods of the unitary test class

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;
    }
  }
}
```



© Prof David Janzen

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?

© Prof David Janzen

TDD EXAMPLE: REFACTOR THE TEST

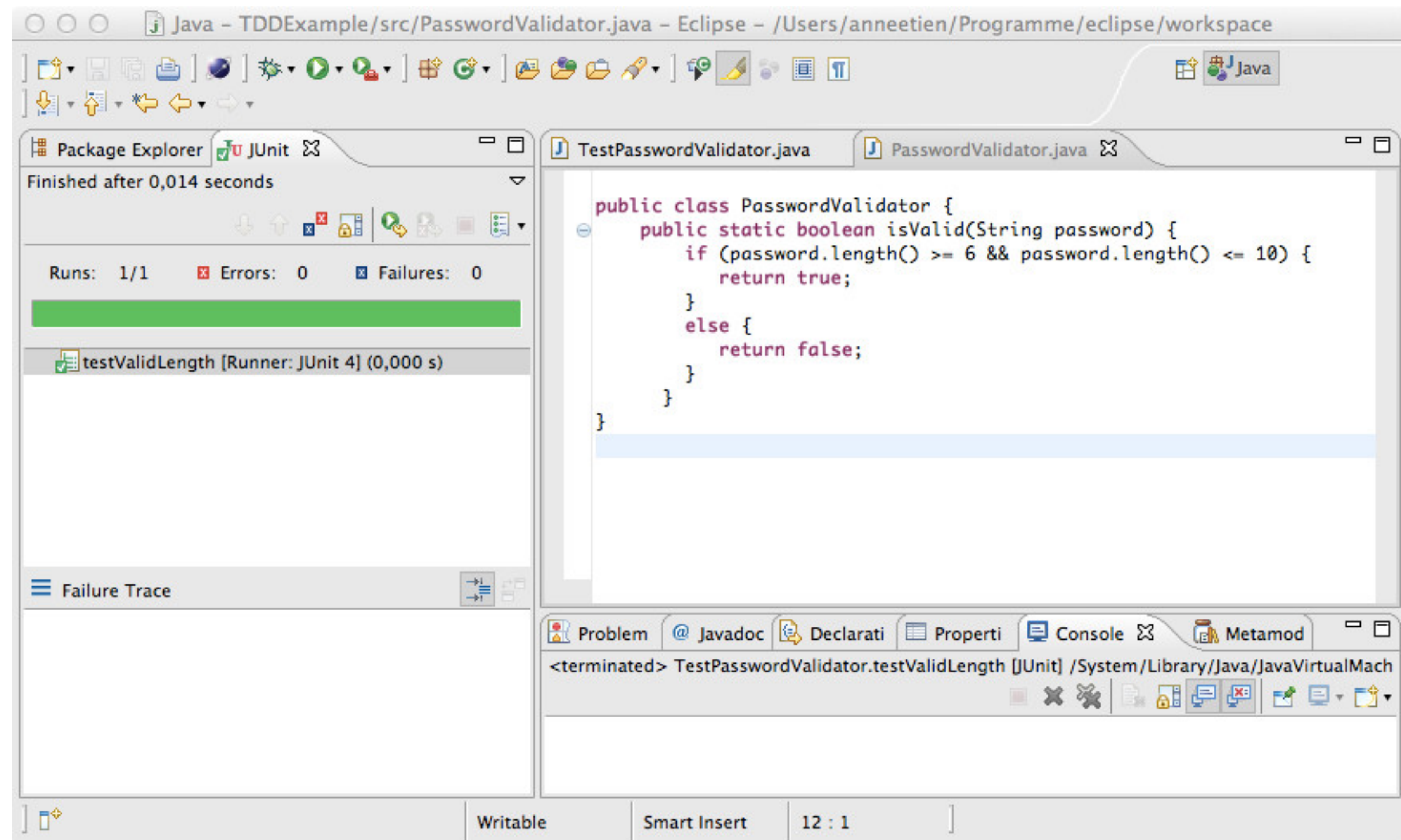
```
import static org.junit.Assert.*;
import org.junit.Test;
public class TestPasswordValidator {
  @Test
  public void testValidLength() {
    assertEquals(true,
PasswordValidator.isValid("Abc123"));
  }
}
```

Design decision: static method

© Prof David Janzen

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;
    }
  }
}
```



© Prof David Janzen

TDD EXAMPLE: REFACTORING #1

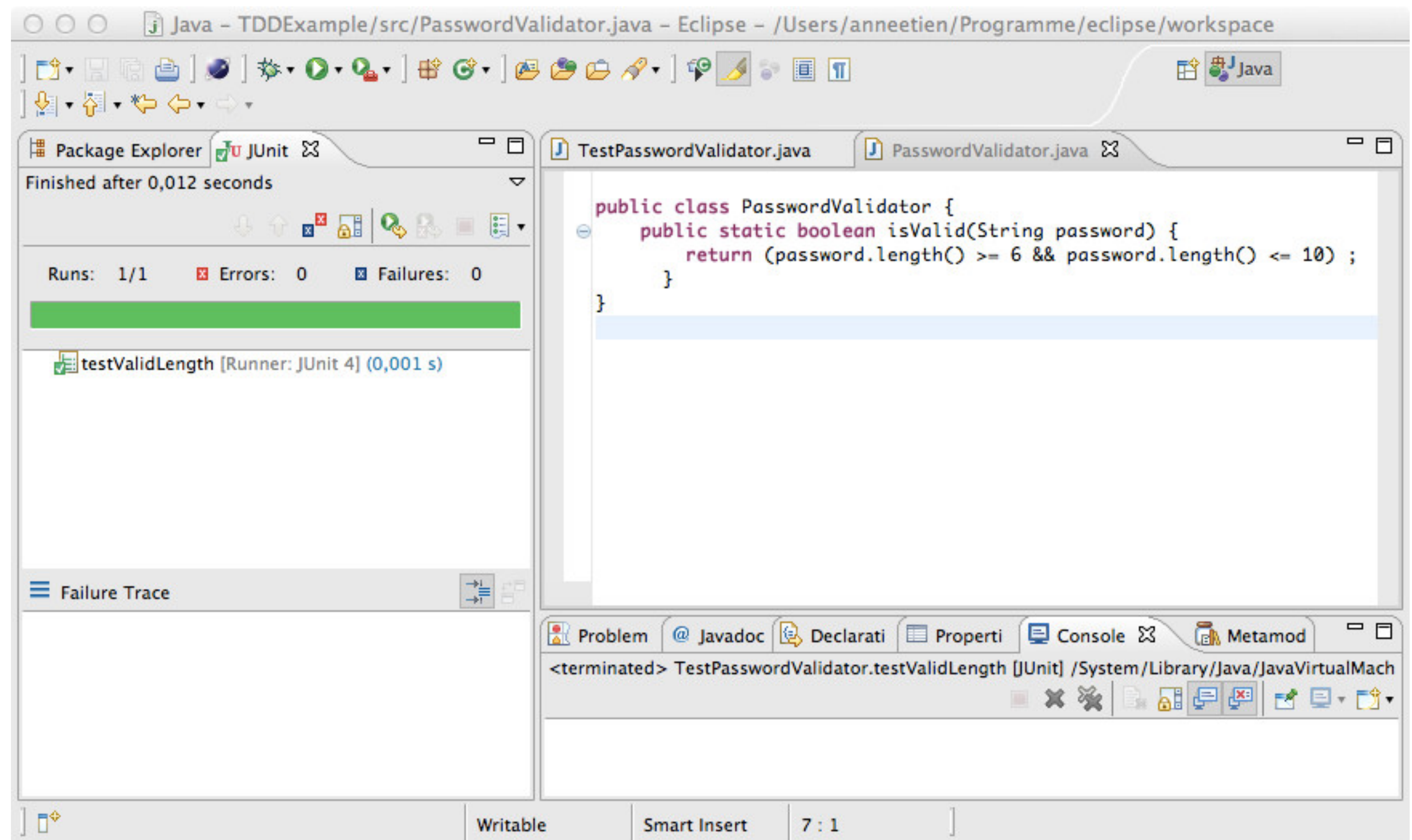
```
public class PasswordValidator {
  public static boolean isValid(String password) {
    if (password.length() >= 6 &&
          password.length() <= 10) {
      return true;
    }
    else {
      return false;
    }
  }
}
```

Can we simplify this?

© Prof David Janzen

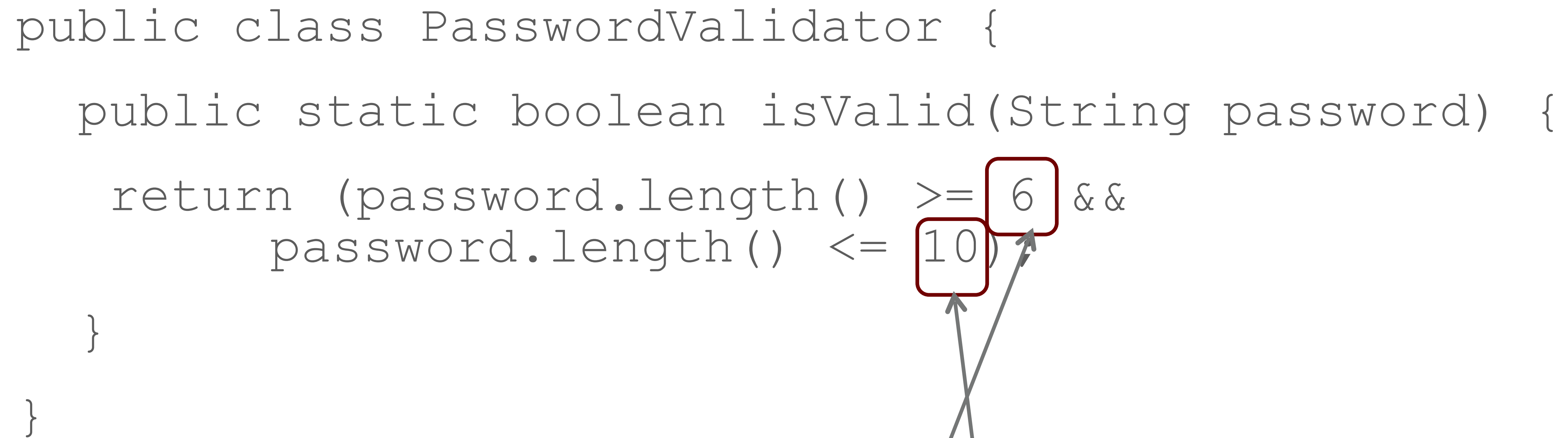
TDD EXAMPLE: REFACTORING #1

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



© Prof David Janzen

TDD EXAMPLE: REFACTORING #1



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

© Prof David Janzen

TDD EXAMPLE: REFACTORING #2

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

Java – TDDExample/src/PasswordValidator.java – Eclipse – /Users/anneetien/Programme/eclipse/workspace

Package Explorer | JUnit

Finished after 0,014 seconds

Runs: 1/1 Errors: 0 Failures: 0

testValidLength [Runner: JUnit 4] (0,000 s)

Failure Trace

TestPasswordValidator.java | PasswordValidator.java

```
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);

    }

}
```

Problem | Javadoc | Declarati | Properti | Console | Metamod

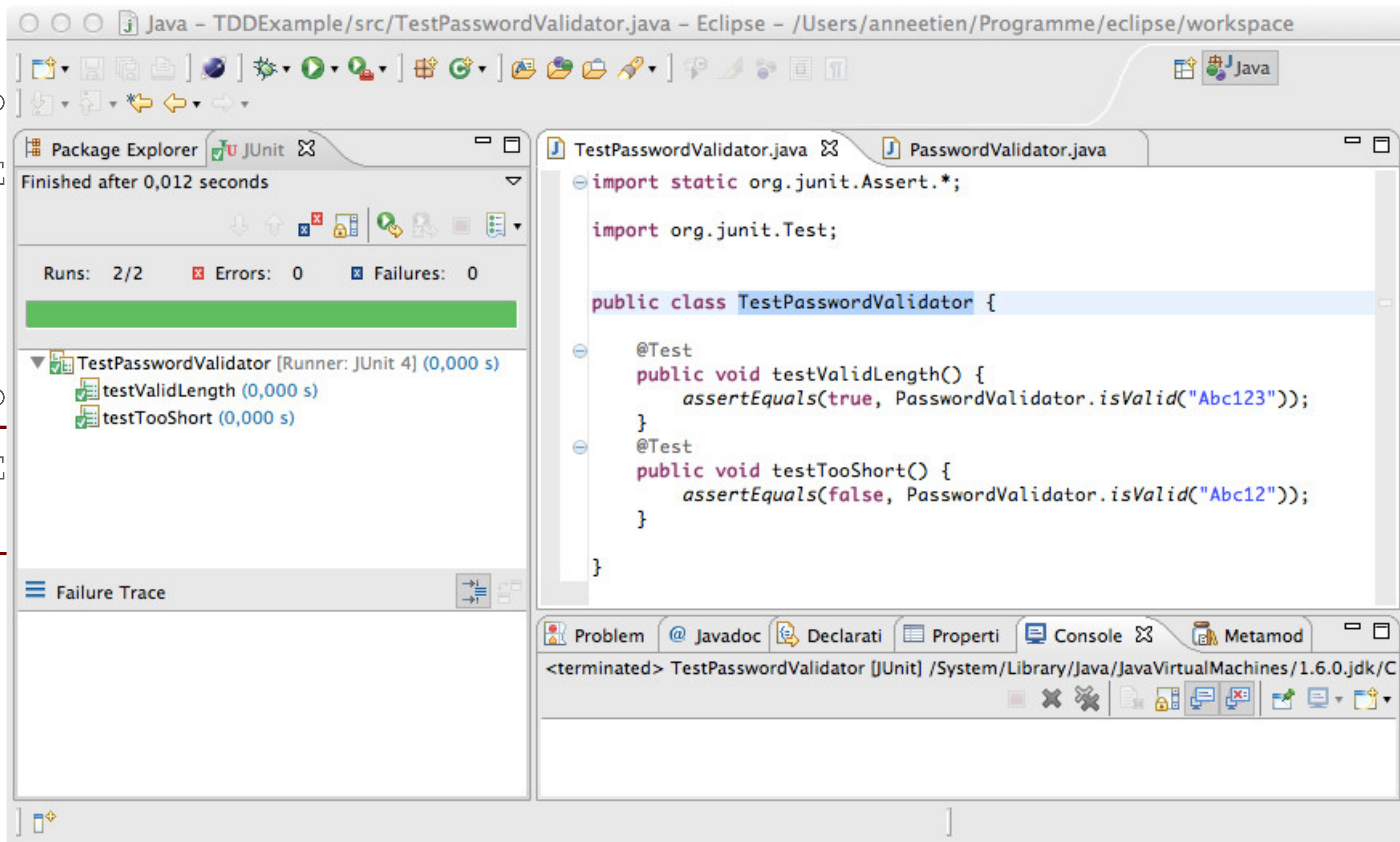
<terminated> Rerun TestPasswordValidator.testValidLength [JUnit] /System/Library/Java/JavaVirtua

Writable | Smart Insert | 12 : 1

© Prof David Janzen

TDD EXAMPLE: WRITE ANOTHER TEST

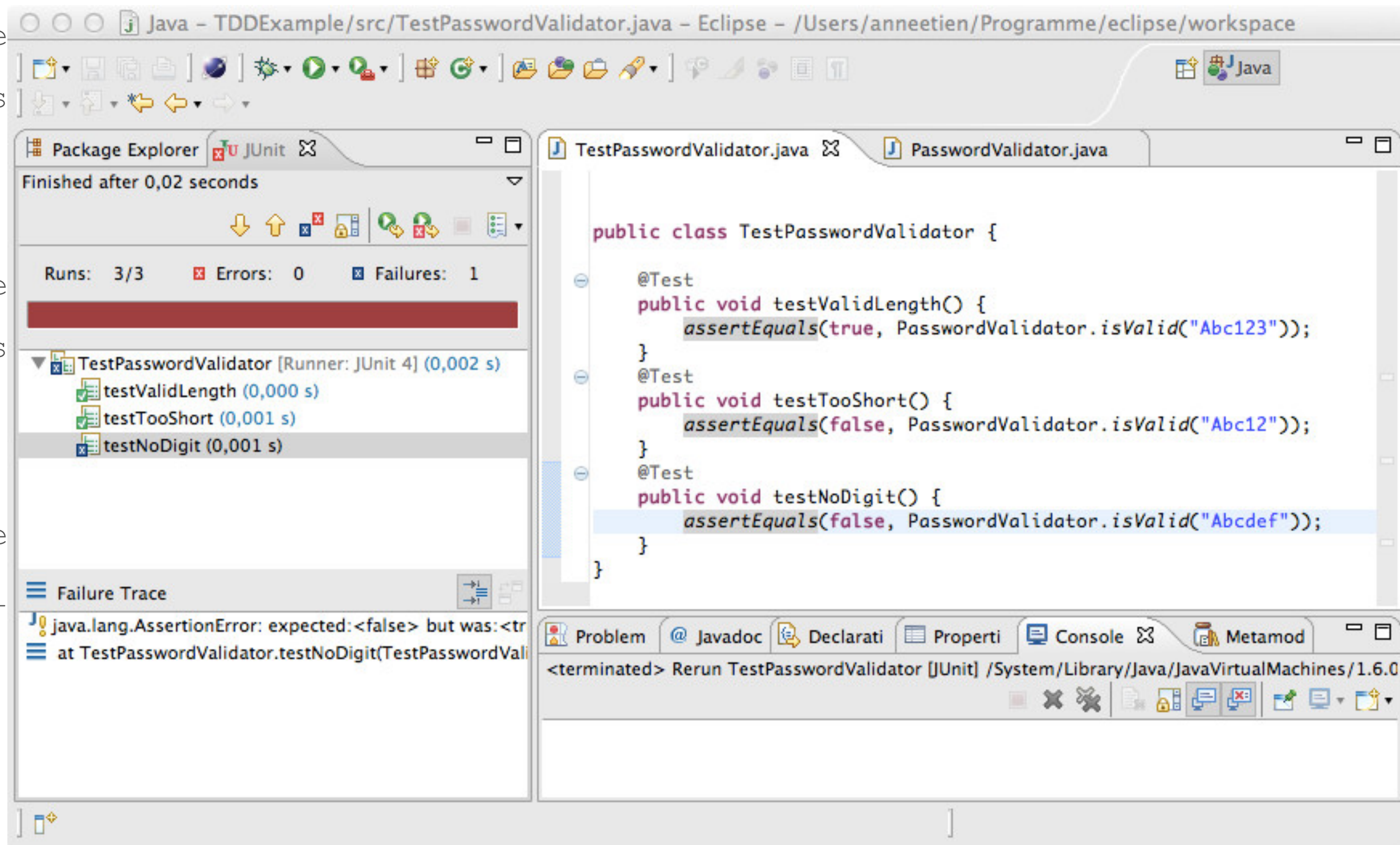
```
import static org.junit.Assert.*;
import org.junit.Test;
public class TestPasswordValidator {
   @Test
   public vo
      assertE
   }
   @Test
   public vo
      assertE
   }
}
```



© Prof David Janzen

TDD EXAMPLE: WRITE ANOTHER TEST

```
import static org.junit.Assert.*;
import org.junit.Test;
public class TestPasswordValidator {
  @Test
  public void te
    assertEquals
  }
  @Test
  public void te
    assertEquals
  }
  @Test
  public void te
     assertEqual
  }
}
```



© Prof David Janzen

TDD EXAMPLE: MAKE THE TEST PASS

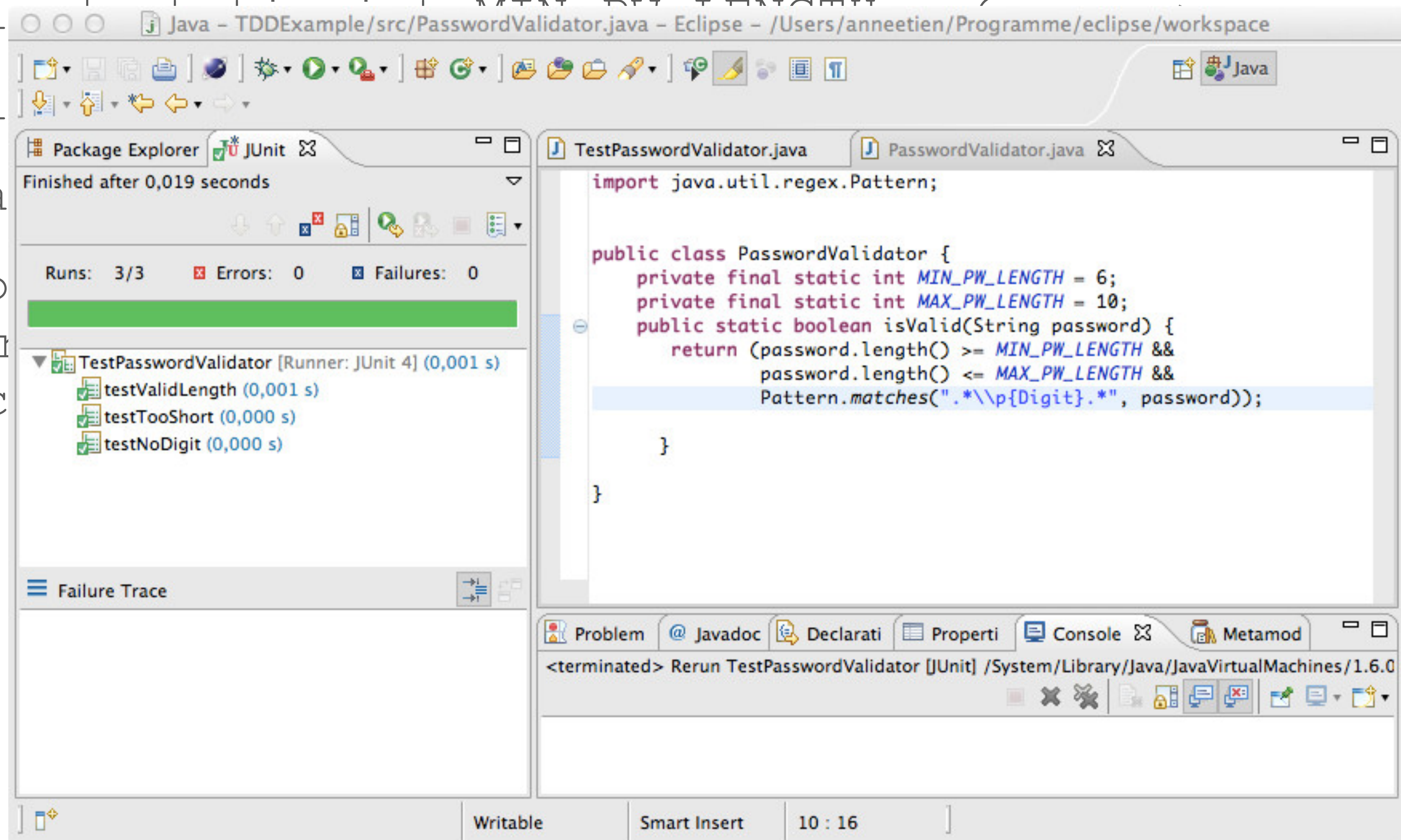
```
public class PasswordValidator {
  private final static int MIN_PW_LENGTH = 6;
  private final static int MAX_PW_LENGTH = 10;
  public static boolean isValid(String password) {
   return (password.length() >= MIN_PW_LENGTH &&
     password.length() <= MAX_PW_LENGTH);
  }
}
```

© Prof David Janzen

TDD EXAMPLE: MAKE THE TEST PASS

```
import java.util.regex.Pattern;
public class PasswordValidator {
  private fi
  private fi
  public sta
   return (p
      passwor
Pattern.matc
password));
  }
}
```

Check for a digit



© Prof David Janzen

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));
  }
}
```

© Prof David Janzen

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);
  }
}
```

© Prof David Janzen

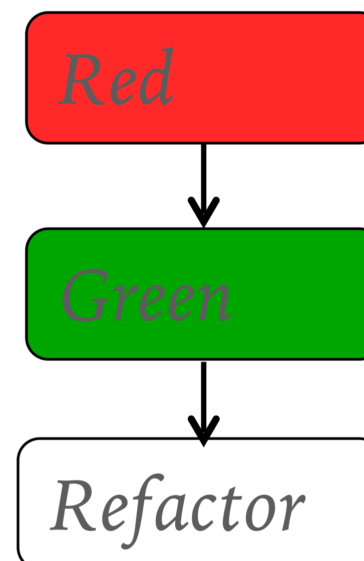
TEST DRIVEN DEVELOPMENT

- 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.

© Prof David Janzen

TEST DRIVEN DEVELOPMENT

- Definition
  - Test-driven Development (TDD) is a programming practice that instructs developers to write new code only if an automated test has failed.
  - The goal of TDD is to think in terms of behaviour, purpose, scenario
- The TDD Cycle[2]
  - Write a test
  - Make it run
  - Make it right



© Prof David Janzen

SOME TYPES OF TESTING

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

and may help here

… and here

© Prof David Janzen

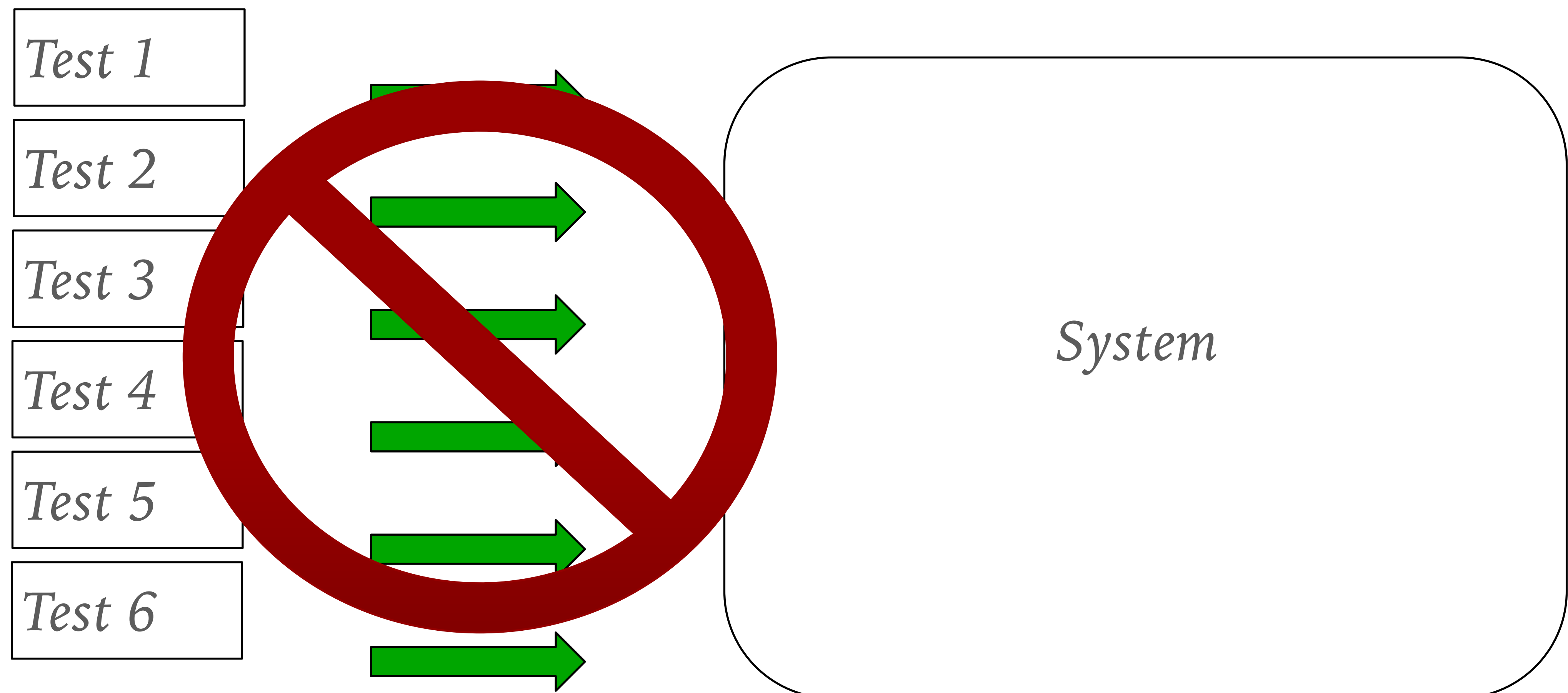
TDD MISCONCEPTIONS

- 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

© Prof David Janzen

TDD MISCONCEPTION #1

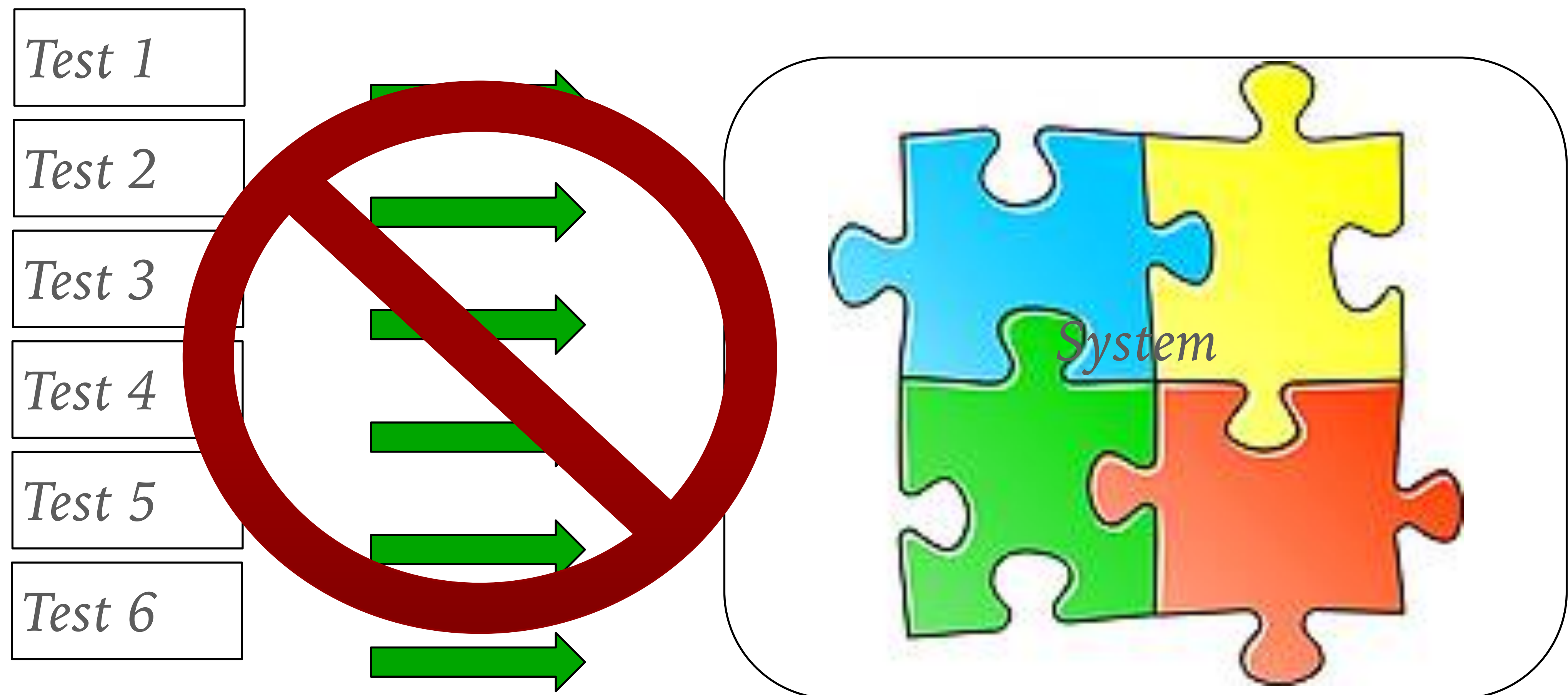
- TDD does not mean "write all the tests, then build a system that passes the tests"



© Prof David Janzen

TDD MISCONCEPTION #2

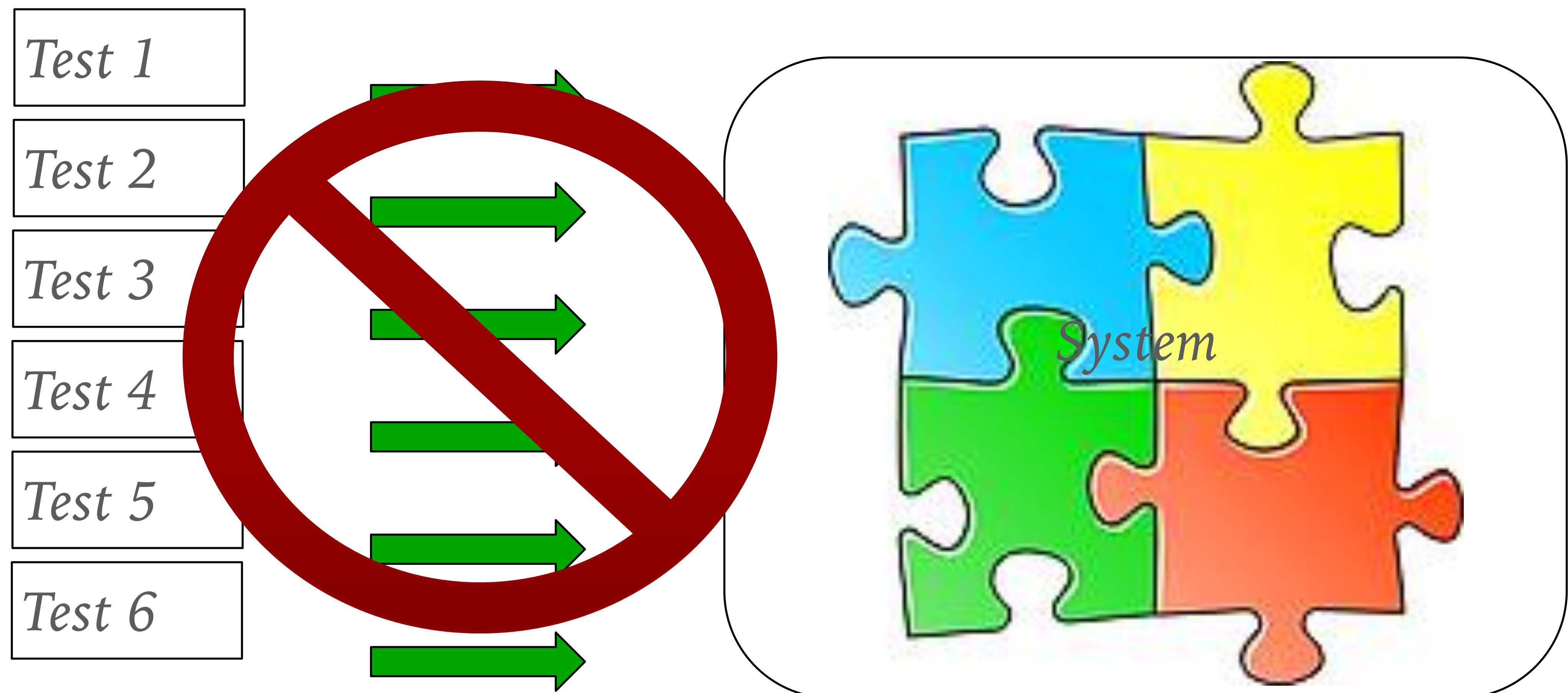
- TDD does not mean “write some of the tests, then build a system that passes the tests”



© Prof David Janzen

TDD MISCONCEPTION #3

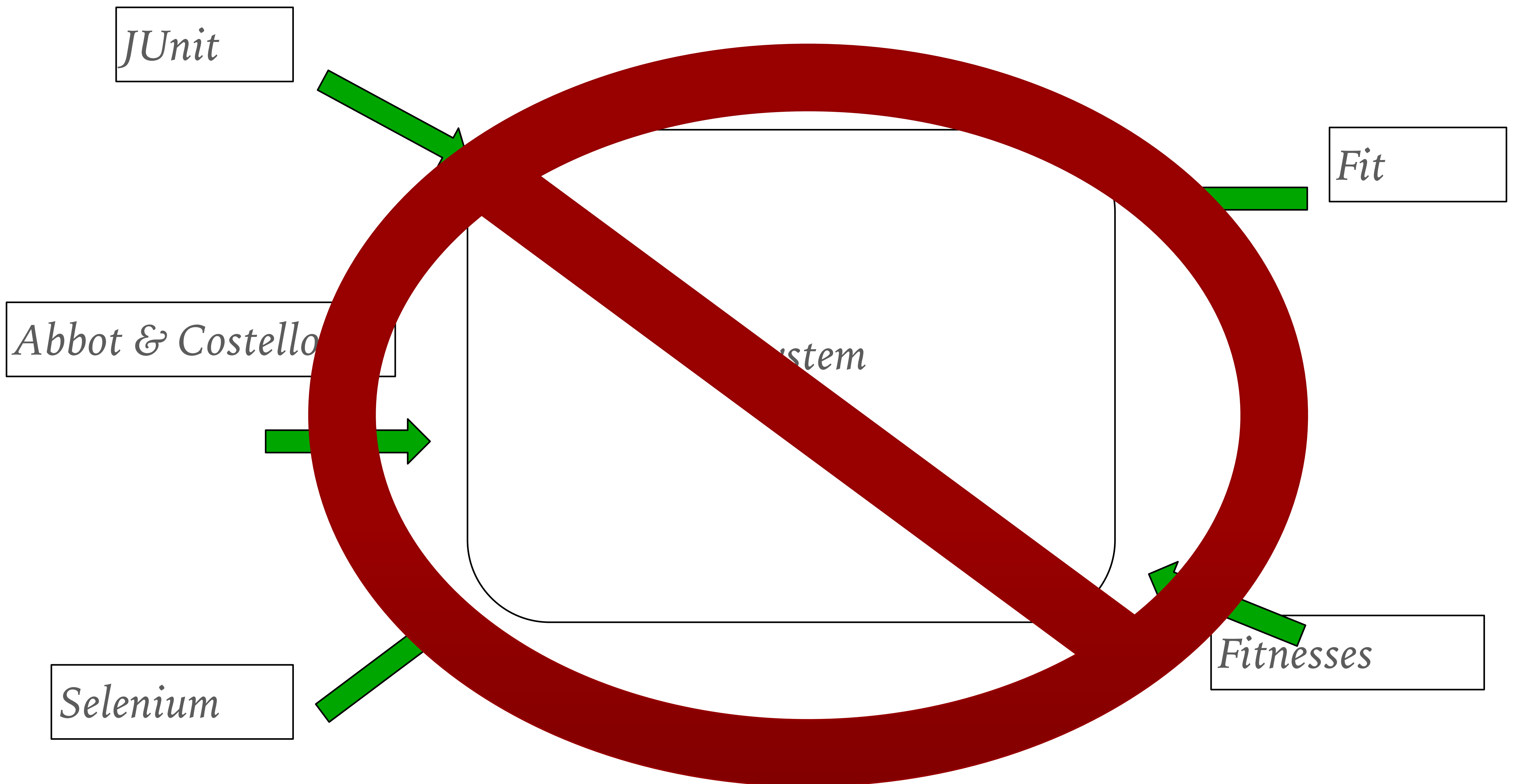
- TDD does not mean "write some of the code, then test it before going on"



© Prof David Janzen

TDD MISCONCEPTION #4

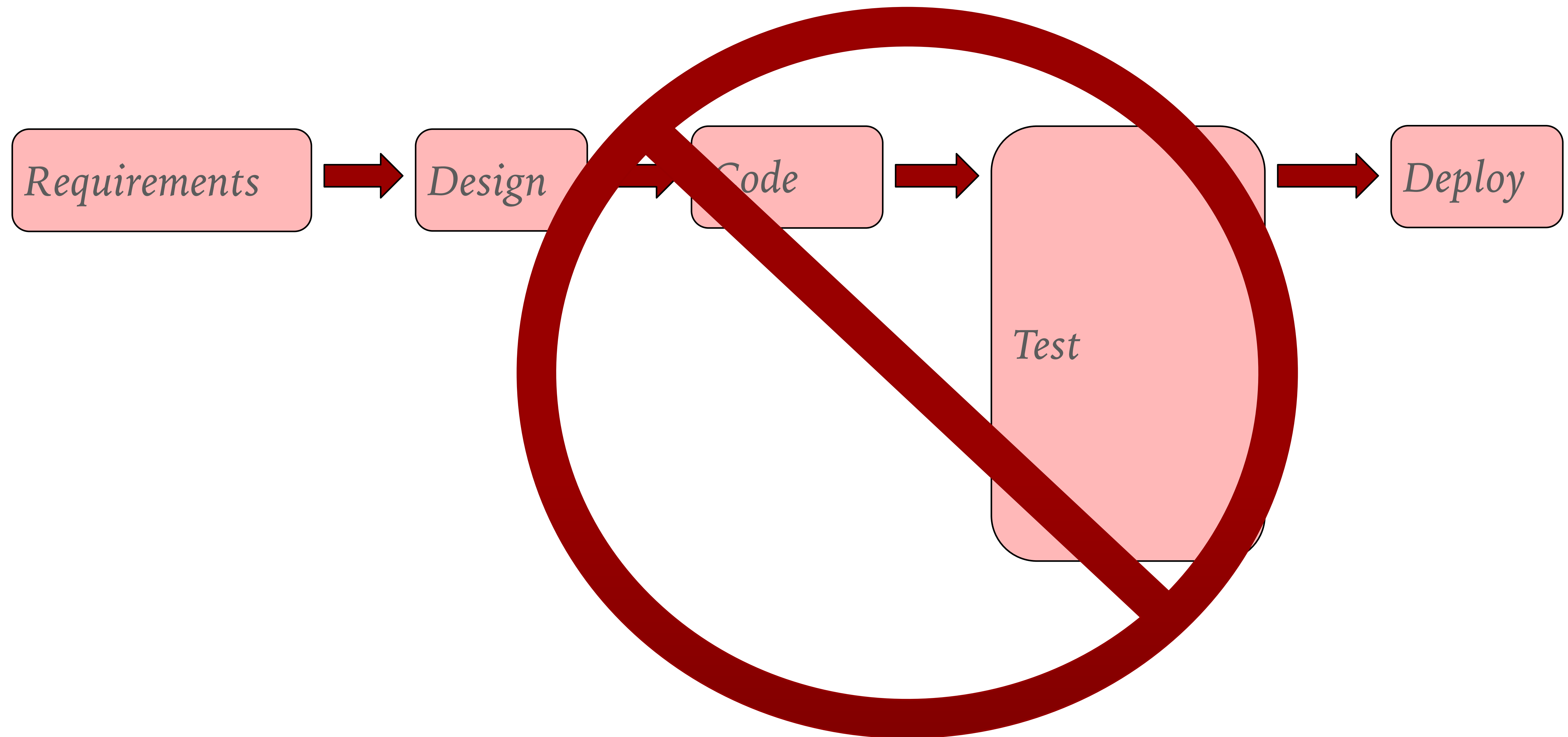
- TDD does not mean “do automated testing”



© Prof David Janzen

TDD MISCONCEPTION #5

- TDD does not mean “do lot of testing”



© Prof David Janzen

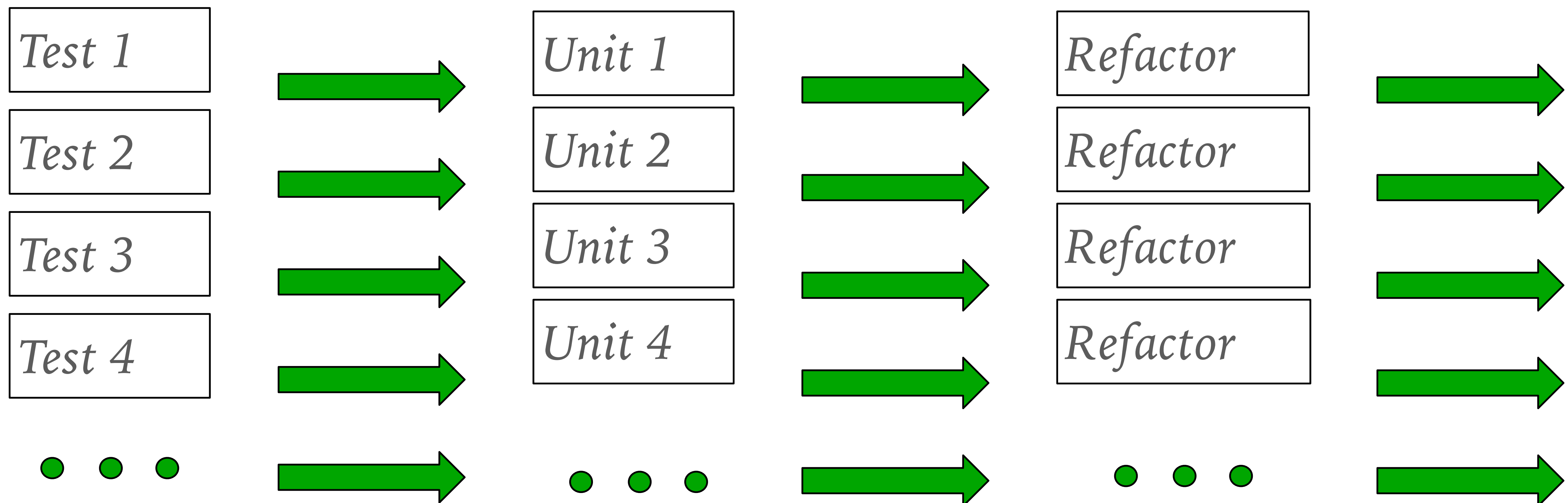
TDD MISCONCEPTION #6

- TDD does not mean "the TDD process"
- TDD is a practice
  - (like pair programming, code reviews, and standup meetings)
- not a process
  - (like waterfall, Scrum, XP, TSP)

© Prof David Janzen

TDD CLARIFIED

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



© Prof David Janzen

SOME VIDEOS TO HELP

- https://www.youtube.com/watch?v=T38L7A0xP-c in English, but 12 minutes.
- https://www.youtube.com/watch?v=nbSaq_ykOl4 in French, almost the same example, but in 45 minutes
- https://www.youtube.com/watch?v=yiCpfd-kz3g in French, an other example, still in 45 minutes
- https://www.youtube.com/watch?v=I8XXfgF9GSc in English about JUnit and Eclipse without TDD (just to understand that you can use JUnit without TDD)

© Prof David Janzen