CS2002 Software Development and Management

Rumyana Neykova
rumyana.neykova@brunel.ac.uk

Software Testing

Why?

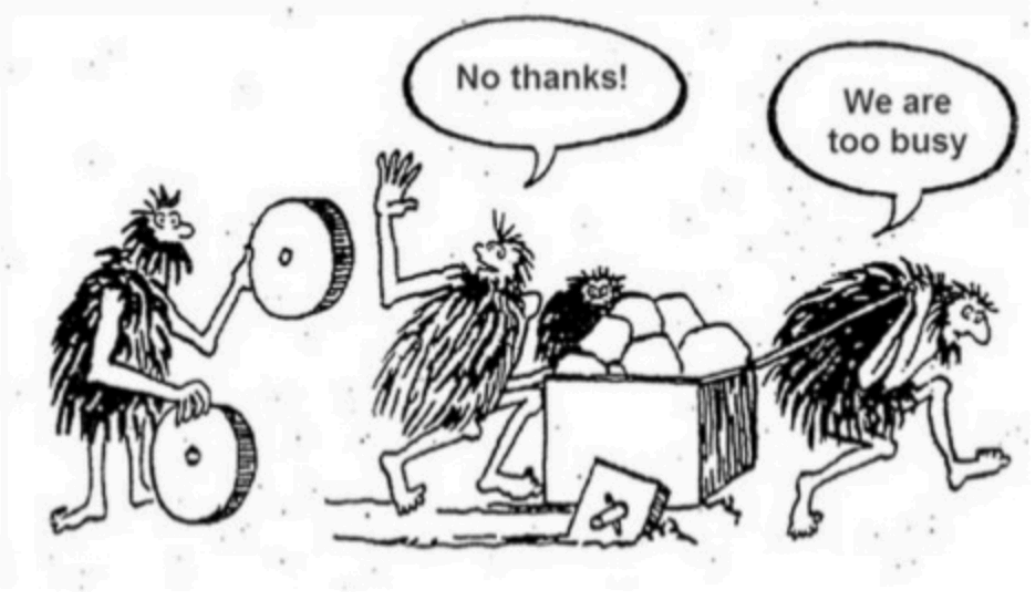
$370 for an integer overflow

Why Ariana 5 failed?

"The main task during the development of Ariane 5 was the reducing of the occasional accident. The exception thrown was not a random accident, but an error in the structure. The exception was detected, but handled incorrectly, because of the point of view that a program should be considered correct, until the opposite is shown. The Commission holds the opposite view, that the software should be considered erroneous, until the best practical current methods demonstrate its correctness."

Lecture outline

- Learning objective:
  - Why testing is important
  - What are the basic types of testing
  - How to write unit tests for a program
  - What is a test coverage and how to improve it?



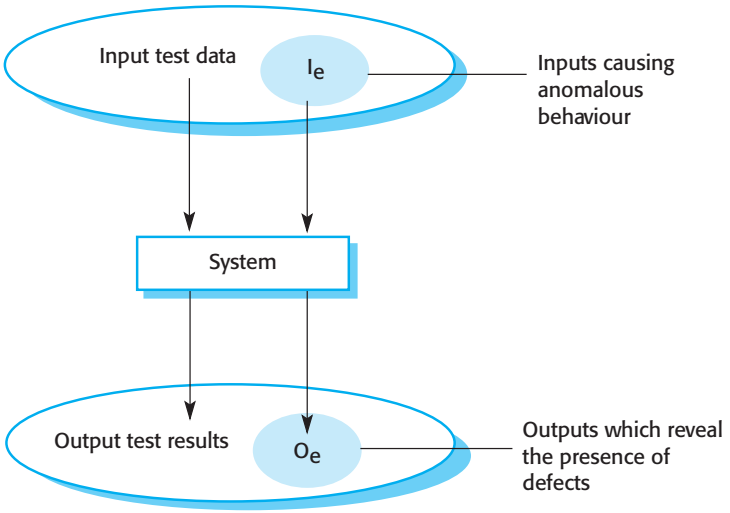
Software Testing

What?

Software Testing is

an activity designed to uncover faults in software

An input-output model of software testing



Example

Suppose the following is meant to calculate the product of the values in an array:

```
int s=0;
int n=0;
while (n<MAX) {
  s=s*a[n];
  n=n+1;
}
return s;
```

Example

Suppose the following is meant to calculate the product of the values in an array:

```
int s=0;
int n=0;
while (n<MAX) {
  s=s*a[n];
  n=n+1;
}
return s;
```

This program always returns 0!

Example

Suppose the following is meant to calculate the product of the values in an array:

```
int s=0;
int n=0;
while (n<MAX)  {
  s=s*a[n];
  n=n+1;
}
return s;
```

a = {0, 1, 2, 3}

Example

Suppose the following is meant to calculate the product of the values in an array:

```
int s=0;
int n=0;
while (n<MAX) {
  s=s*a[n];
  n=n+1;
}
return s;
```

a = {0, 1, 2, 3}
a = {1, 2, 3}

Example

Suppose the following is meant to calculate the product of the values in an array:

```
int s=1;
int n=0;
while (n<MAX) {
  s=s*a[n];
  n=n+1;
}
return s;
```

”Testing can only show the presence of errors, not their absence”

Dijkstra, 1972

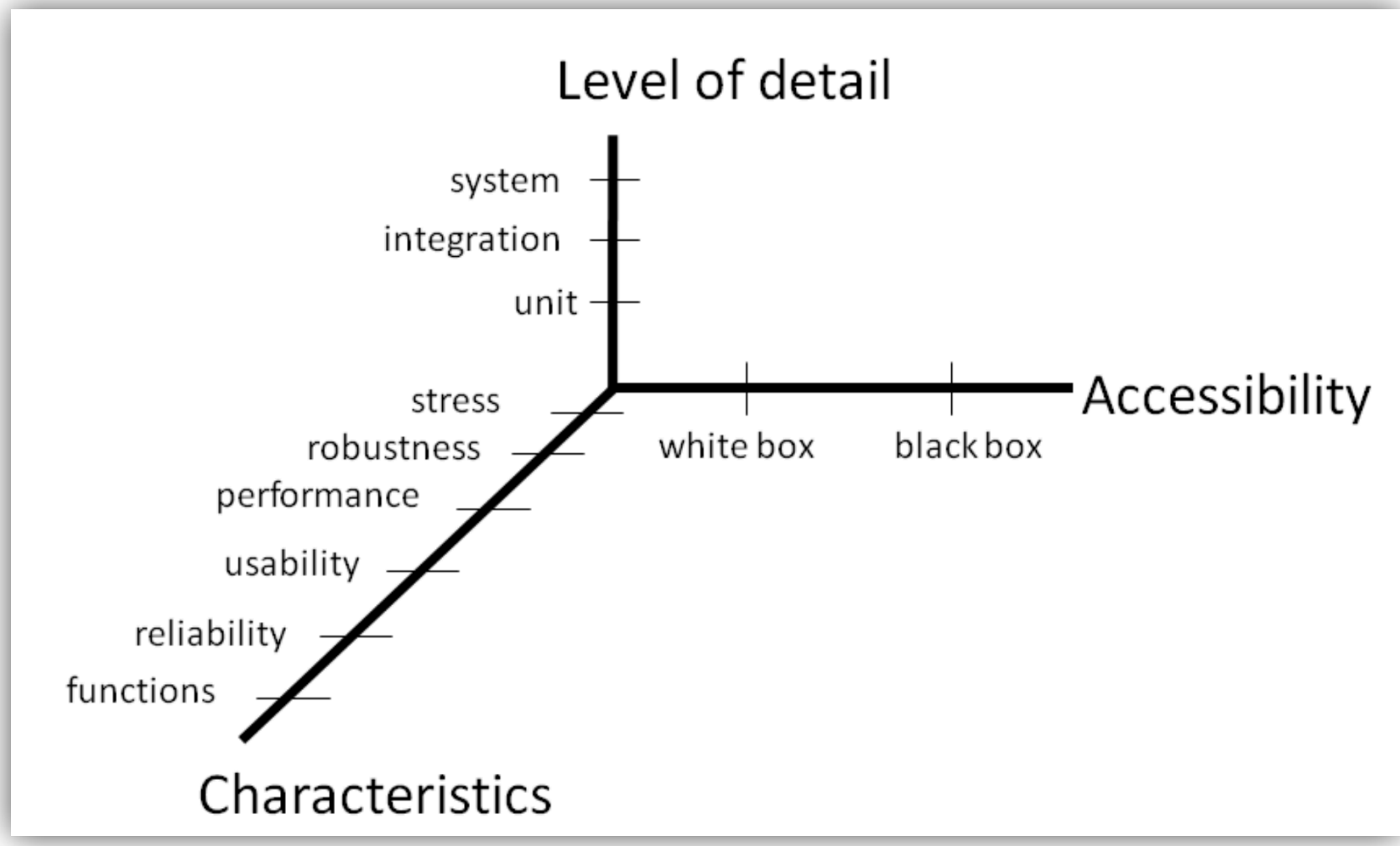
Software Testing

Types of Testing

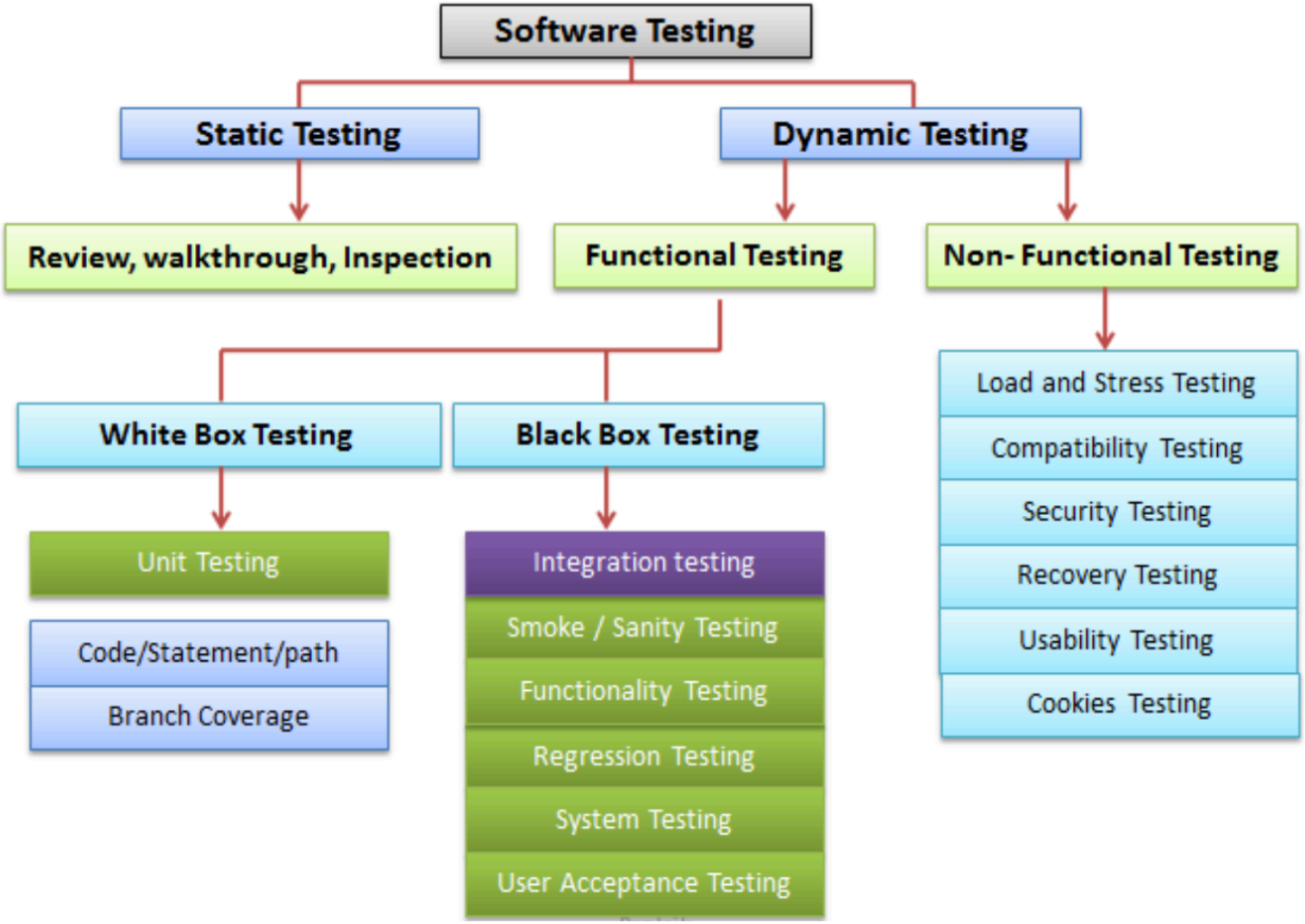
SOFTWARE TESTING AS A MARTIAL ART

~ There is no One Style ~

Types of software testing



Types of software testing



Types of software testing

```
public int fun(int param) {
   int result
   result = param/2;
   return result
}
```

Types of software testing

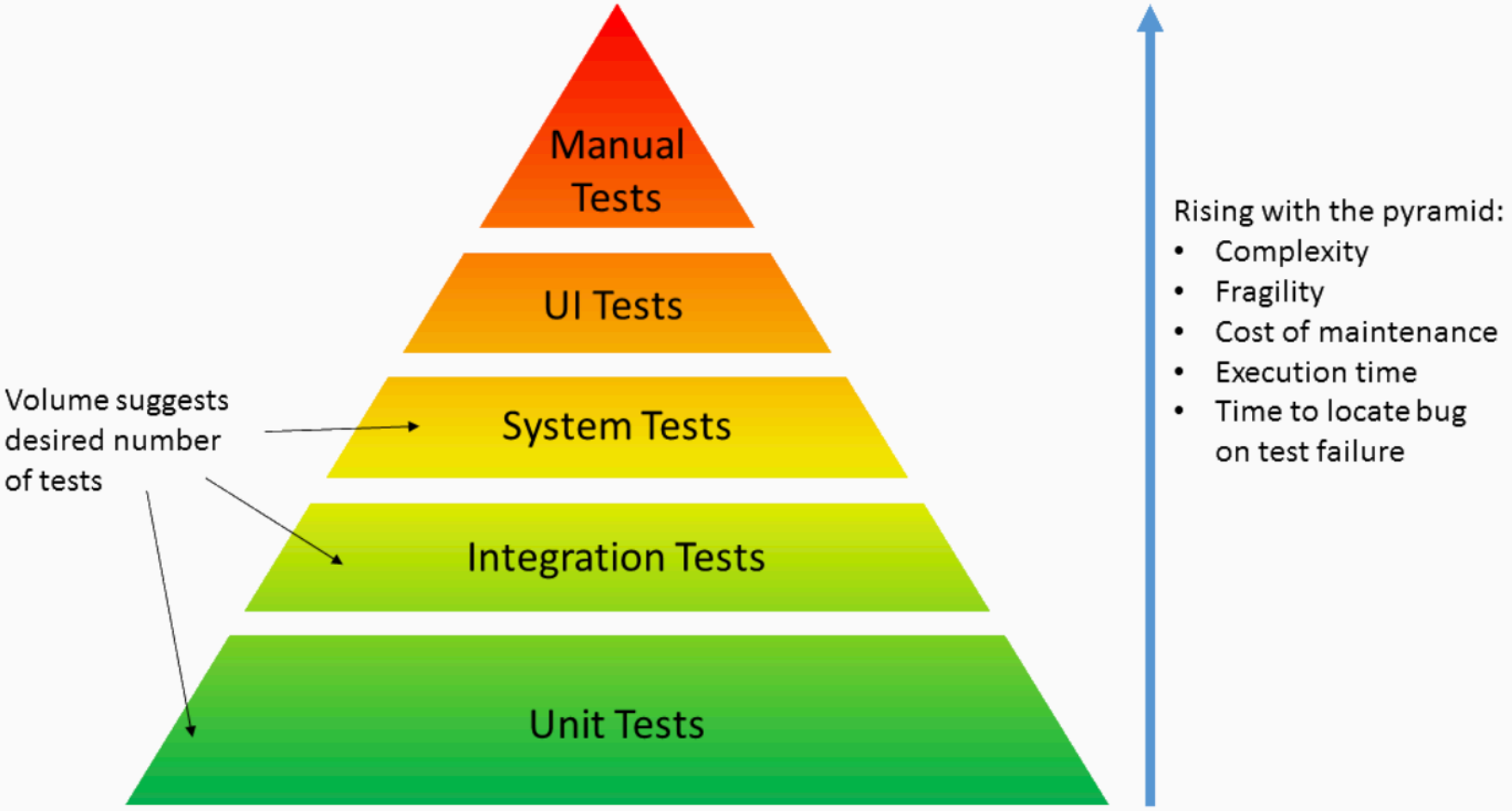
Specification: inputs an integer param and returns half of its value if even, its value otherwise

```
public int fun(int param) {
   int result
   result = param/2;
   return result
}
```

Stages of Testing

- Development testing, where the system is tested during development to discover bugs and defects
- Release testing, where a separate testing team test a complete version of the system before it is released to users.
- User testing, where users or potential users of a system test the system in their own environment

The test pyramid



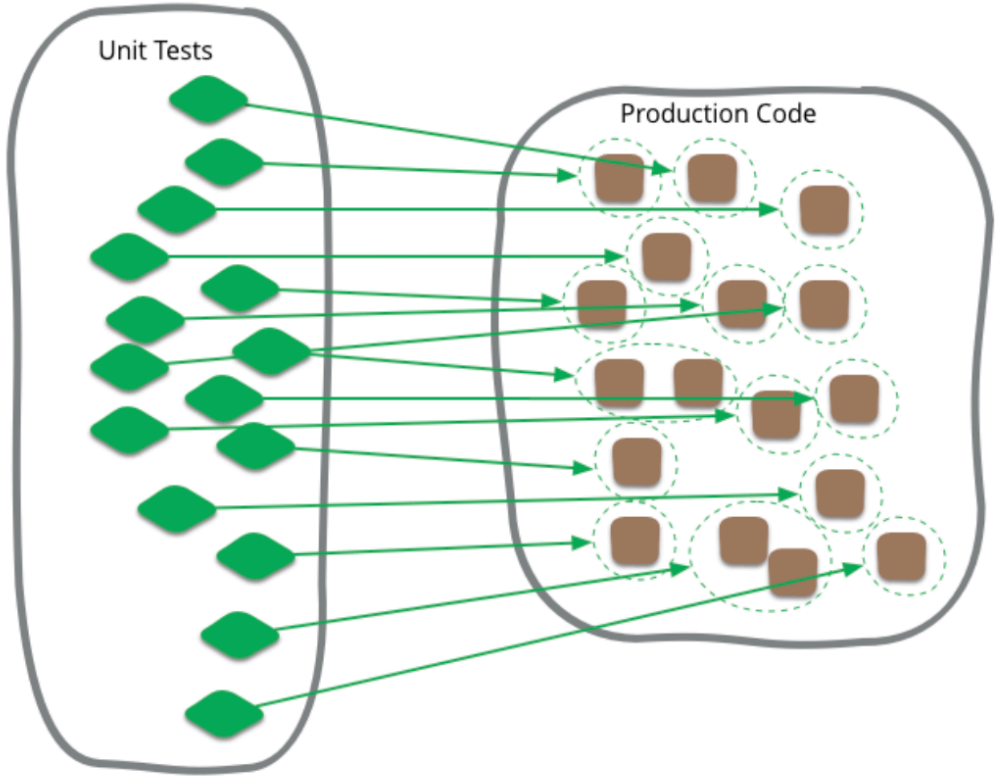
Software Testing

Unit Testing

Unit tests: Why?

"More than the act of testing, the act of designing tests is one of the best bug preventers known. ***The thinking that must be done to create a useful test can discover and eliminate bugs before they are coded*** *– indeed, test-design thinking can discover and eliminate bugs at every stage in the creation of software, from conception to specification, to design, coding and the rest." – Boris Beizer*

Unit tests: What?



- Low-level focusing on a **small** part of the system
- **Written by the software developers**
- Should be **fast**, so they can be repeated many times

Unit testing: How?

1. **Arrange:** setup the test data
2. **Act:** call your method under test
3. **Assert** that the expected results are returned

Unit testing: How?

1. **Arrange:** setup the test data
2. **Act:** call your method under test
3. **Assert** that the expected results are returned

```
[TestMethod]
public void IsPalindrome_ForPalindromeString_ReturnsTrue()
{
    // In the Arrange phase, we create and set up a system under test.
    // A system under test could be a method, a single object, or a graph of connec
    // It is OK to have an empty Arrange phase, for example if we are testing a sta
    // in this case SUT already exists in a static form and we don't have to initia
    PalindromeDetector detector = new PalindromeDetector();

    // The Act phase is where we poke the system under test, usually by invoking a
    // If this method returns something back to us, we want to collect the result t
    // Or, if method doesn't return anything, we want to check whether it produced
    bool isPalindrome = detector.IsPalindrome("kayak");

    // The Assert phase makes our unit test pass or fail.
    // Here we check that the method's behavior is consistent with expectations.
    Assert.IsTrue(isPalindrome);
}
```

Unit testing: How?

1. **Arrange:** setup the test data
2. **Act:** call your method under test
3. **Assert** that the expected results are returned

```
[TestMethod]
public void IsPalindrome_ForPalindromeString_ReturnsTrue()
{
    // In the Arrange phase, we create and set up a
    // A system under test could be a method, a single object, or a graph of connec
    // It is OK to have an empty Arrange phase, for example if we are testing a sta
    // in this case SUT already exists in a static form and we don't have to initia
    PalindromeDetector detector = new PalindromeDetector();

    // The Act phase is where we poke the system under test, usually by invoking a
    // If this method returns something back to us, we want to collect the result t
    // Or, if method doesn't return anything, we want to check whether it produced
    bool isPalindrome = detector.IsPalindrome("kayak");

    // The Assert phase makes our unit test pass or fail.
    // Here we check that the method's behavior is consistent with expectations.
    Assert.IsTrue(isPalindrome);
}
```

Setup test data

Unit testing: How?

1. **Arrange:** setup the test data
2. **Act:** call your method under test
3. **Assert** that the expected results are returned

```
[TestMethod]
public void IsPalindrome_ForPalindromeString_ReturnsTrue()
{
    // In the Arrange phase, we create and set up a system under test.
    // A system under test could be a method, a single object, or a graph of connec
    // It is OK to have an empty Arrange phase, for example if we are testing a sta
    // in this case SUT already e                                    ave to initia
    PalindromeDetector detector =

    // The Act phase is where we poke the system under test, usually by invoking a
    // If this method returns something back to us, we want to collect the result t
    // Or, if method doesn't return anything, we want to check whether it produced
    bool isPalindrome = detector.IsPalindrome("kayak");

    // The Assert phase makes our unit test pass or fail.
    // Here we check that the method's behavior is consistent with expectations.
    Assert.IsTrue(isPalindrome);
}
```

Call the method under test

Unit testing: How?

1. **Arrange:** setup the test data
2. **Act:** call your method under test
3. **Assert** that the expected results are returned

```
[TestMethod]
public void IsPalindrome_ForPalindromeString_ReturnsTrue()
{
    // In the Arrange phase, we create and set up a system under test.
    // A system under test could be a method, a single object, or a graph of connec
    // It is OK to have an empty Arrange phase, for example if we are testing a sta
    // in this case SUT already exists in a static form and we don't have to initia
    PalindromeDetector detector = new PalindromeDetector();

    // The Act phase is wh                                              ing a
    // If this method retu                                              sult t
    // Or, if method doesn't return anything, we want to check whether it produced
    bool isPalindrome = detector.IsPalindrome("kayak");

    // The Assert phase makes our unit test pass or fail.
    // Here we check that the method's behavior is consistent with expectations.
    Assert.IsTrue(isPalindrome);
}
```

Assert the expected results are returned

```java
@Test
public void palindromString_returnTrue() {
    PalindromDetector detector = new PalindromDetector();
    boolean result = detector.isPalindrome("kayak");
    Assert.assertTrue(result);
}
```

```java
public boolean isPalindrome(String inputString) {
    int length  = inputString.length();
    int i, begin, end, middle;

    begin  = 0;
    end    = length-1;
    middle = (begin + end)/2;

    for (i = begin; i <=middle; i++) {
      if (inputString.toLowerCase().charAt(begin) ==
              inputString.toLowerCase().charAt(end)) {
        begin++;
        end--;
      }
    }

    if (i == middle+1) {return true;}
    else {return false;}
  }
```

Unit Test: Best Practices

- Each unit test should be able to run independent of other tests.
- Each test should test just one thing (single behavior, method, or function)
- All external dependencies should be mocked.
- The assumptions for each test should be clear and defined within the test method.
- Name of the test method should be meaningful.
- Unit tests should be fast, so that it could be run as often as required

What is wrong with this code?

```
public static string GetTimeOfDay()
{
    DateTime time = DateTime.Now;
    if (time.Hour >= 0 && time.Hour < 6)
    {
        return "Night";
    }
    if (time.Hour >= 6 && time.Hour < 12)
    {
        return "Morning";
    }
    if (time.Hour >= 12 && time.Hour < 18)
    {
        return "Afternoon";
    }
    return "Evening";
}
```

What is wrong with this code?

```
public static string GetTimeOfDay()
{
    DateTime time = DateTime.Now;
    if (time.Hour >= 0 && time.Hour < 6)
    {
        return "Night";
    }
    if (time.Hour >= 6 && time.Hour < 12)
    {
        return "Morning";
    }
    if (time.Hour >= 12 && time.Hour < 18)
    {
        return "Afternoon";
    }
    return "Evening";
}
```

This is a hidden input

Let's fix it!

```
public static string GetTimeOfDay(DateTime dateTime)
{
    if (dateTime.Hour >= 0 && dateTime.Hour < 6)
    {
        return "Night";
    }
    if (dateTime.Hour >= 6 && dateTime.Hour < 12)
    {
        return "Morning";
    }
    if (dateTime.Hour >= 12 && dateTime.Hour < 18)
    {
        return "Noon";
    }
    return "Evening";
}
}
```

```
[TestMethod]
public void GetTimeOfDay_For6AM_ReturnsMorning()
{
    // Arrange phase is empty: testing static method, nothing to initialize

    // Act
    string timeOfDay = GetTimeOfDay(new DateTime(2015, 12, 31, 06, 00, 00));

    // Assert
    Assert.AreEqual("Morning", timeOfDay);
}
```

Software Testing

Test Coverage

Statement Coverage

```
public void printSum(int a, int b) {
    int result = a+b;
    if (result>0)
        System.out.println("positive" + result);
    else if (result<0)
        System.out.println("negative" + result);
}
```

Consider:

Test Case 1: a=3, b = 9

Test Case 2 : a=-5, b =-8

Branch Coverage

```
public void printSum(int a, int b) {
    int result = a+b;
    if (result>0)
        System.out.println("positive" + result);
    else if (result<0)
        System.out.println("negative" + result);
    [else do nothing]
}
```

Consider:

Test Case 1: a=3, b = 9

Test Case 2 : a=-5, b =-8

Condition Coverage

```
public int[] main(int x, int y) {
    if ((x==0) || (y>2))
        y = y/x;
    else
        x = y/2;
    return new int[]{x, y};
}
```

Consider:

Test Case 1: x=5, y=6

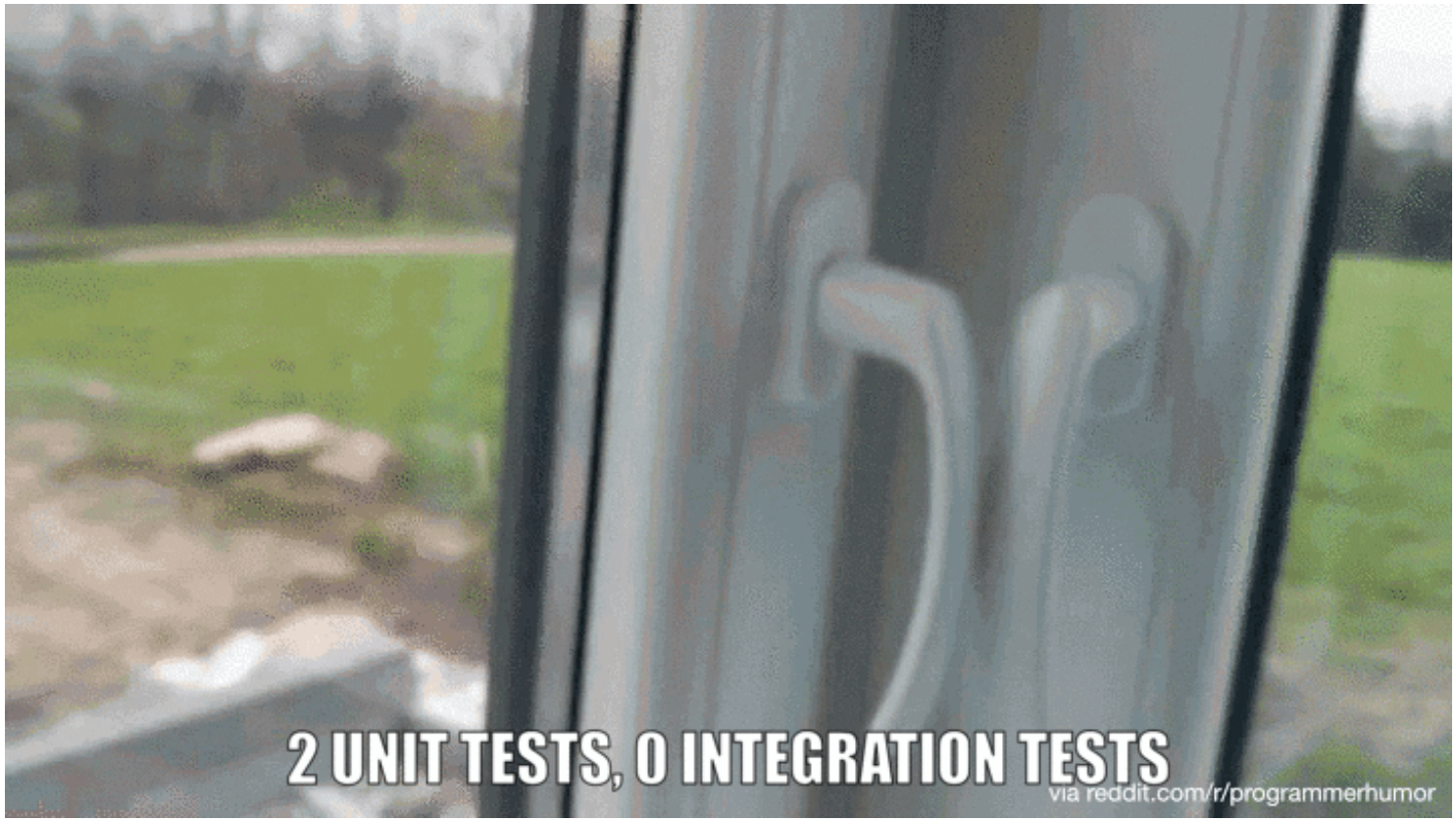
Test Case 2 : x=5, y=-5

Test Coverage: Why?

Test Coverage: Why?

- Measure used to describe how much the tests exercise the code
- Offers a quantitative measurement.
- Types of coverage:
  - Function (was a particular function called?)
  - Statement (was a particular line of code executed)
  - Branch (was an edge in the program executed
  - Other – increasingly expensive to calculate
- 100% test coverage DOES NOT guarantee 0 bugs

Unit Testing is NOT enough!



Release testing

- Release testing is:
  - testing a release that is intended for use outside of the development team
  - usually a black-box testing process - tests are only derived from the specification
  - done by a team not involved in development
- The goal is to convince the supplier that the system is good enough for use.
- In contrast, in system testing the development team focus on discovering bugs in the system.

User testing

- User or customer testing
  - is a stage in the testing process in which users provide input and advice on system testing
  - is essential, even when comprehensive system and release testing have been carried out
  - the influences from the user's working environment have an effect on the reliability, performance, usability and robustness. These cannot be replicated in a testing environment
- Two types of user testing
  - Alpha testing: users of the software work with the development team to test the software at the developer's sit
  - Beta testing: a release of the software is made available to users to experiment at their own site

Summary

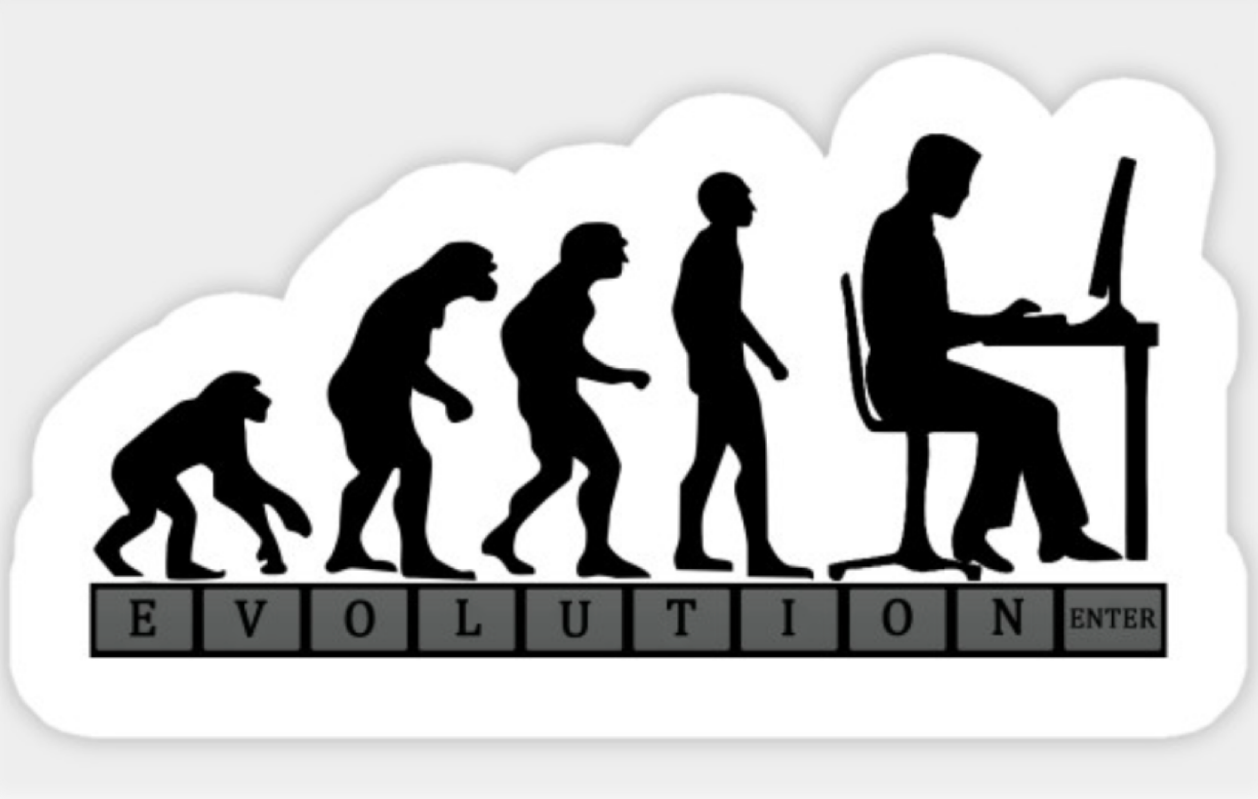
- *Testing is an activity designed to uncover faults in software*
- *In software testing we: execute the system under test on test input; observe the output produced; compare this output with what is expected*
- *Three stages of testing: development, release, user testing*
- *A unit test should be small, fast, written by the programmer*
- *Unit testing is part of the development testing stage, should be combined with other types of testing, such as integration, system testing, etc.*
- *Writing unit tests will make you a better developer; strive to write testable code*
- *Code coverage* measure the efficiency of test implementation

Don’t sweep bugs under the carpet!

Test early, test often!

Next lecture…

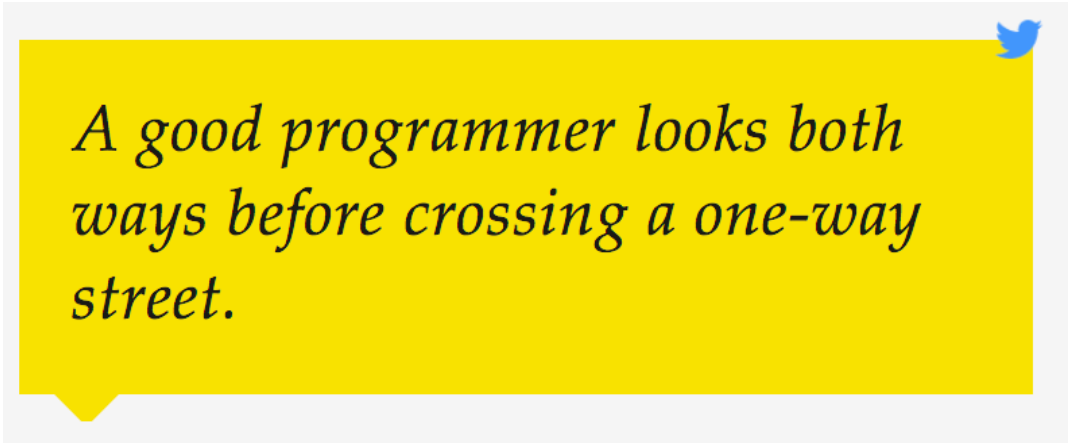
Software Evolution



by Dr Sarath Dantu

References&Conclusion

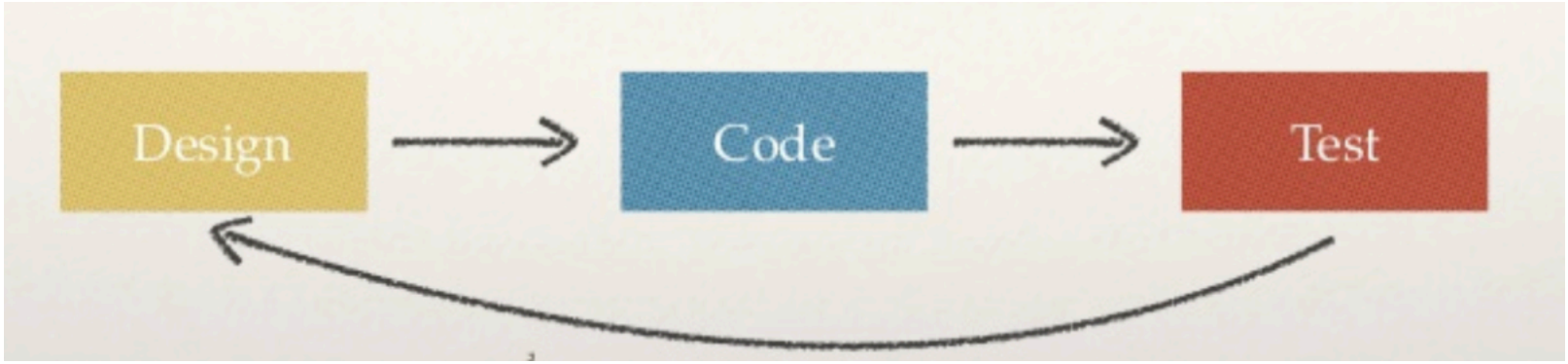
- Chapters 8 of Sommerville
- Chapter 9 of Pfleeger & Atlee
- Martin Fowler TDD series
- Check the many resources on the slides



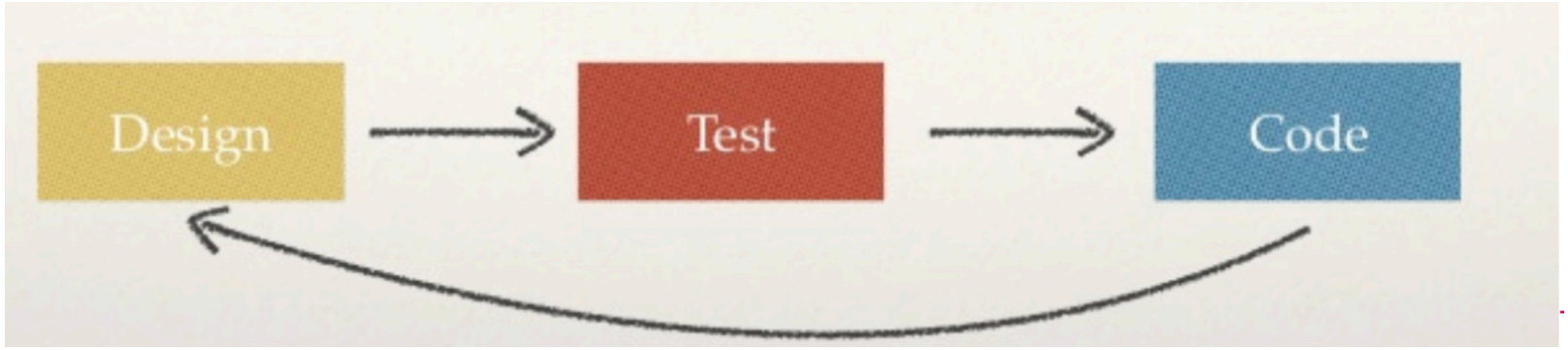
Bonus slides

Test driven development 1/2

Bonus slides

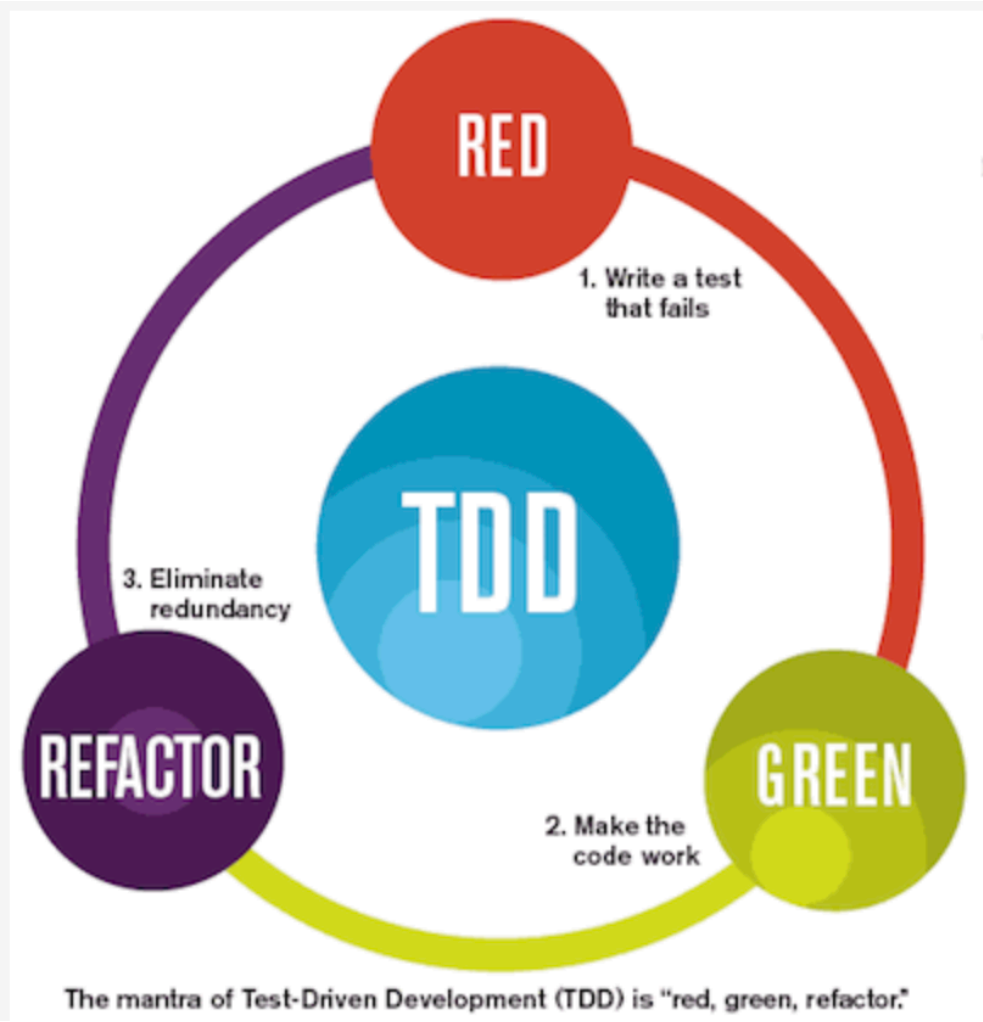


VS



Test Driven Development 2/2

Bonus slides



The mantra of Test-Driven Development (TDD) is "red, green, refactor."

The three rules of TDD

Bonus slides

1. You are not allowed to write any production code unless it is to make a failing unit test pass.
2. You are not allowed to write any more of a unit test than is sufficient to fail, and compilation failures are failures.
3. You are not allowed to write any more production code than is sufficient to pass the one failing unit test.

Challenge: TDD KATA

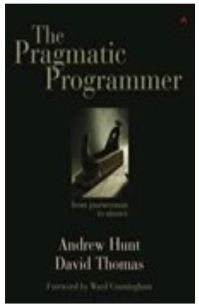
Bonus slides

Challenge: TDD KATA

Bonus slides

As a group, software developers don't practice enough. Most of our learning takes place on the job, which means that most of our mistakes get made there as well. Other creative professions practice: artists carry a sketchpad, musicians play technical pieces, poets constantly rewrite works. In karate, where the aim is to learn to spar or fight, most of a student's time is spent learning and refining basic moves. The more formal of these exercises are called kata.

Dave Thomas
http://codekata.pragprog.com, 2007



Challenge: TDD KATA

*Do a TDD kata **every day** for two weeks!*

Resources: https://github.com/mwhelan/Katas