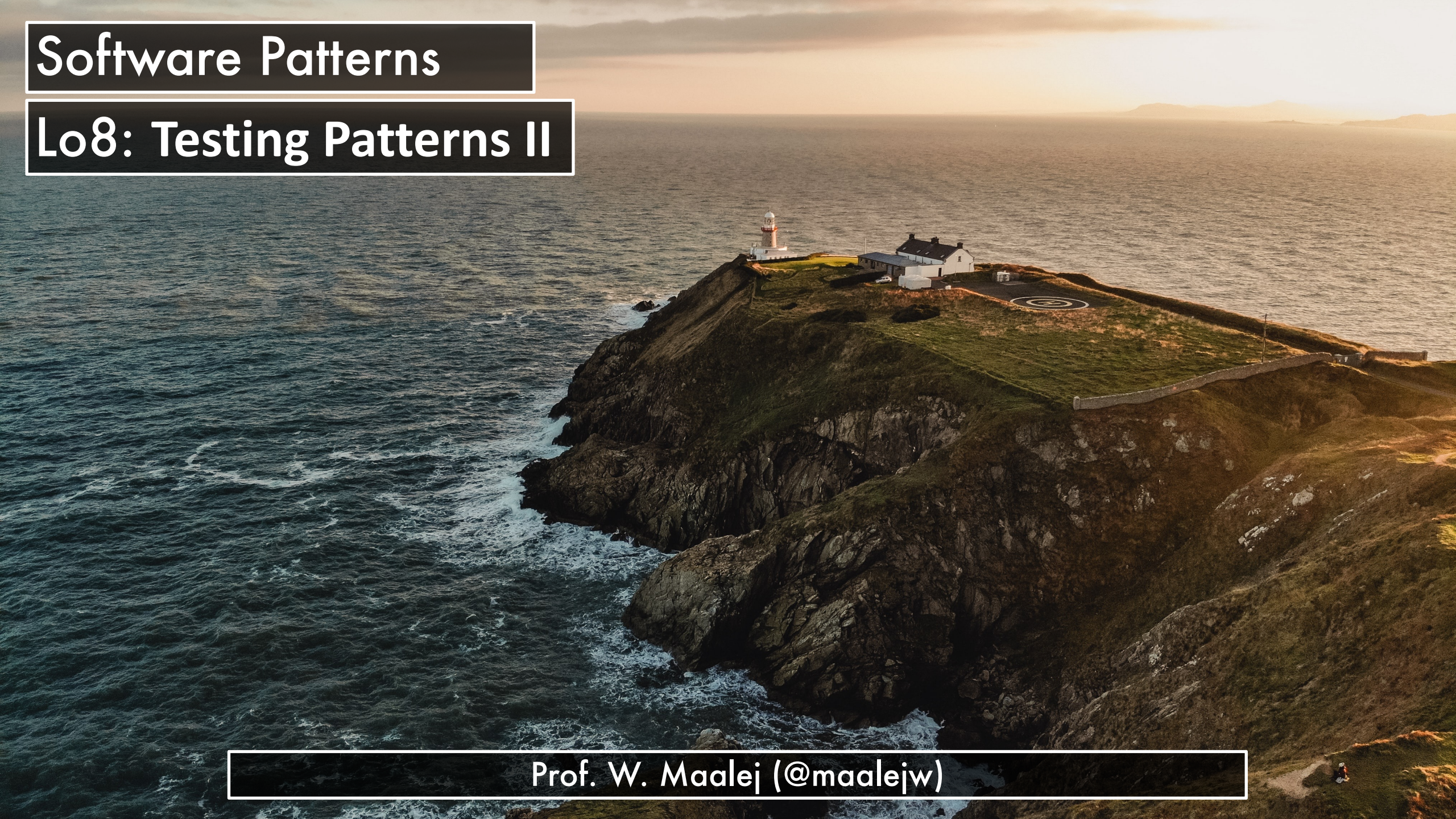
Software Patterns

Lo8: Testing Patterns II



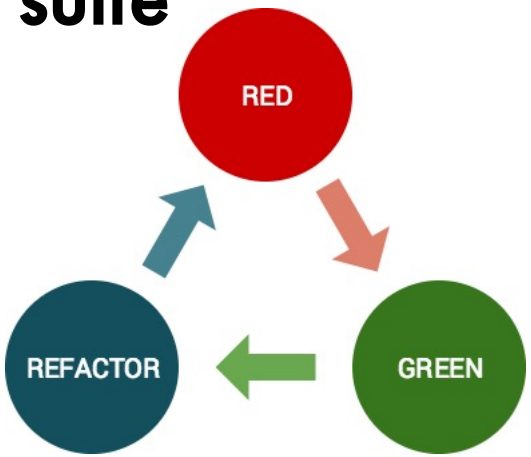
Prof. W. Maalej (@maalejw)

Outline of the talk

1. Test-driven Development
2. Acceptance Testing

Introducing Test-driven Development

- An iterative *development* technique
- Focus on *design* rather than validation
- **Creates a large *regression* test suite**



(Claimed) TDD effects

- Improved and faster software development
- Results in flexible and extensible code
- Reduce bugs
- Improve documentation

TDD Mantra

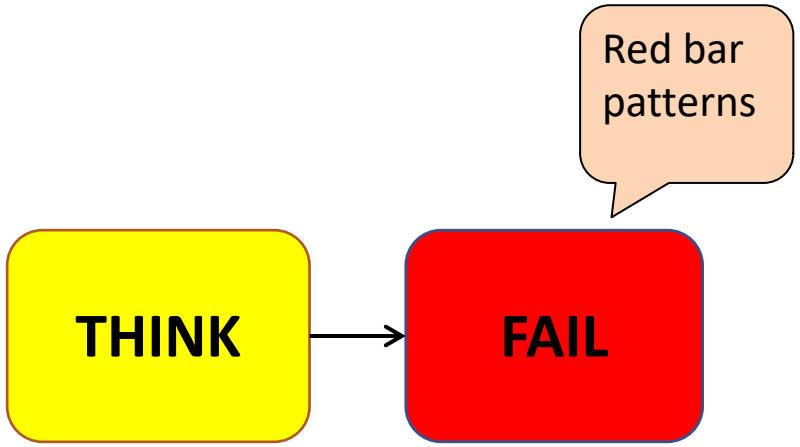
- Write test before implementation
- Refactor, refactor, refactor
- Relies on unit testing and unit testing frameworks
  - *"Unit tests run fast, if they do not run fast they are not unit tests"*

TDD: Step 1

THINK

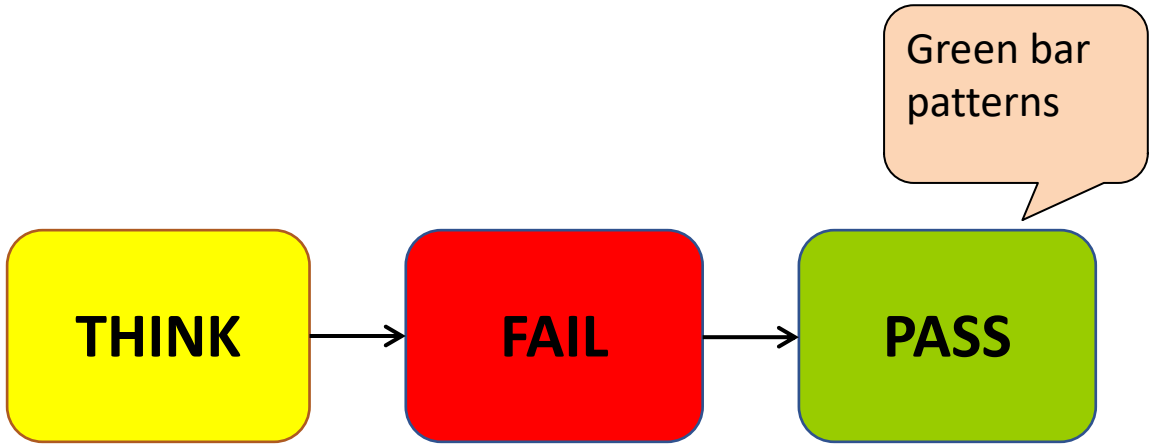
- Think in terms of *baby steps*
- Focus on the *behaviour* you wish your code will have

TDD: Step 2



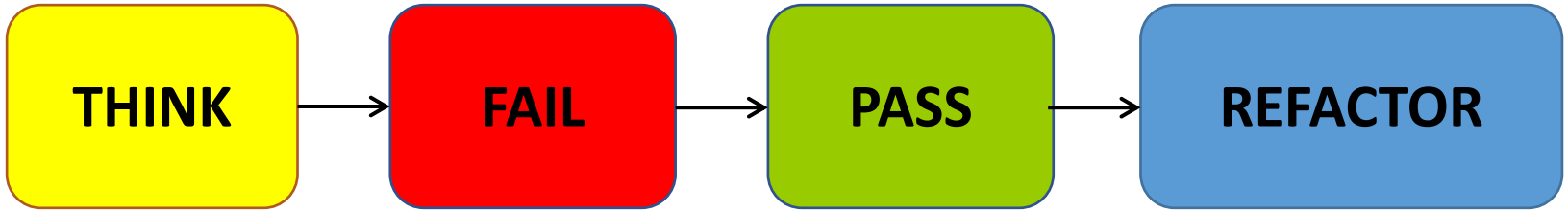
- Write a test, watch it fail
- Focus on the *behaviour* your class and method will have
  - Name your API properly

TDD: Step 3



- Write *enough production* code to make the test pass
  - No matter how ugly
- Don't try to make things perfect the first time
  - "Learn" design from the tests

TDD: Step 4

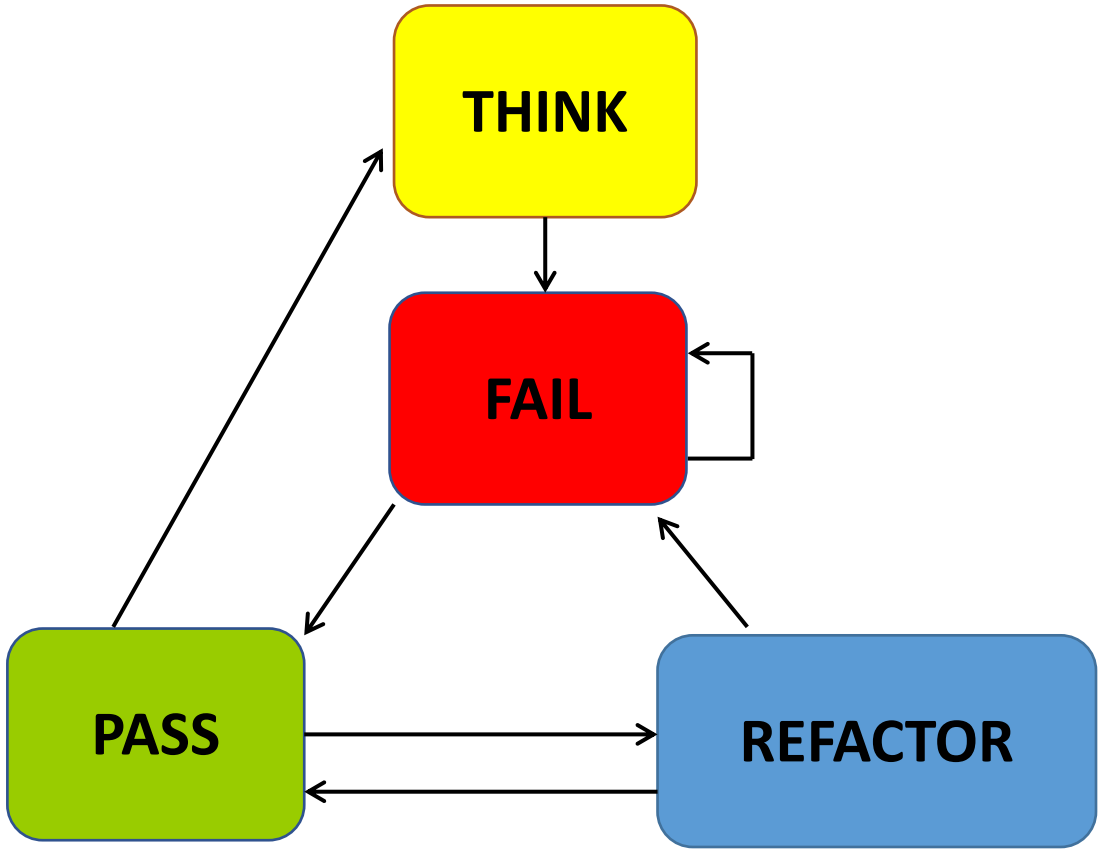


- We are finally allowed to refactor

Remember:

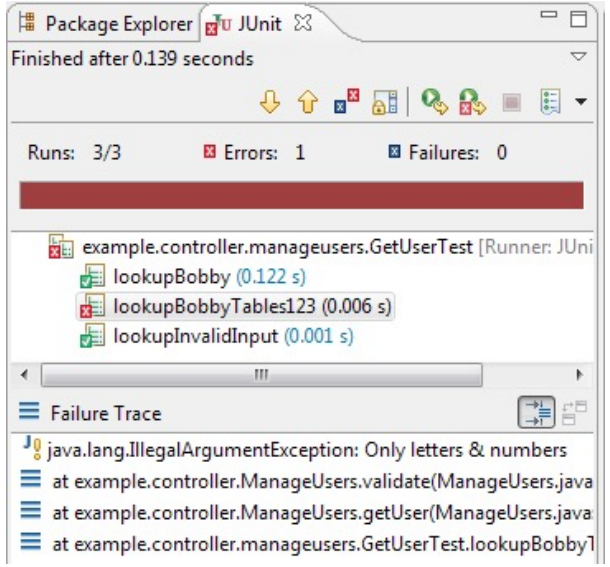
- Simplify and improve the design
- Do NOT modify the behaviour!

TDD Cycle



Context: Red Bar Patterns

- Starter test
  - Which test to start with
- One step test
  - Which test to pick next
- Learning test
  - When to test for external software
- Regression test
  - How to keep focused
- Child test
  - How to run a big test case



Starter Test

Problem

- Which test should you start with?

Solution

- Start by testing a variant of an operation that doesn't do anything
- Answer the question "Where does it belong?"
- Beginning with a realistic test will leave you too long without feedback
- You can shorten the loop by choosing inputs and outputs that are trivially easy to discover

One-step Test

Problem

- Which test should you implement next?

Solution

- Pick a test that will teach you something
- Pick a test you are confident you can implement

Learning Test (spikes)

- **Problem**

When to write tests for externally produced software?

Solution

- Before the first time you are going to use a new utility in the package

- We are going to develop something on top of a library we never used before.
  - Do we just write the code and expect it to work?
  - Instead of just using it, we write a little test that verifies that the API works as expected

Regression Test

- **Problem**

What's the first thing you do when a bug is reported?

Solution

- Write the smallest possible test that fails and that, once run, the bug will be repaired
- *Regression* tests are tests that, with perfect foreknowledge, you would have written *when* coding
- You may have to refactor the system before you can easily isolate the bug

Child Test

Problem

- How do you get a test case running that turns out to be too big?

Solution

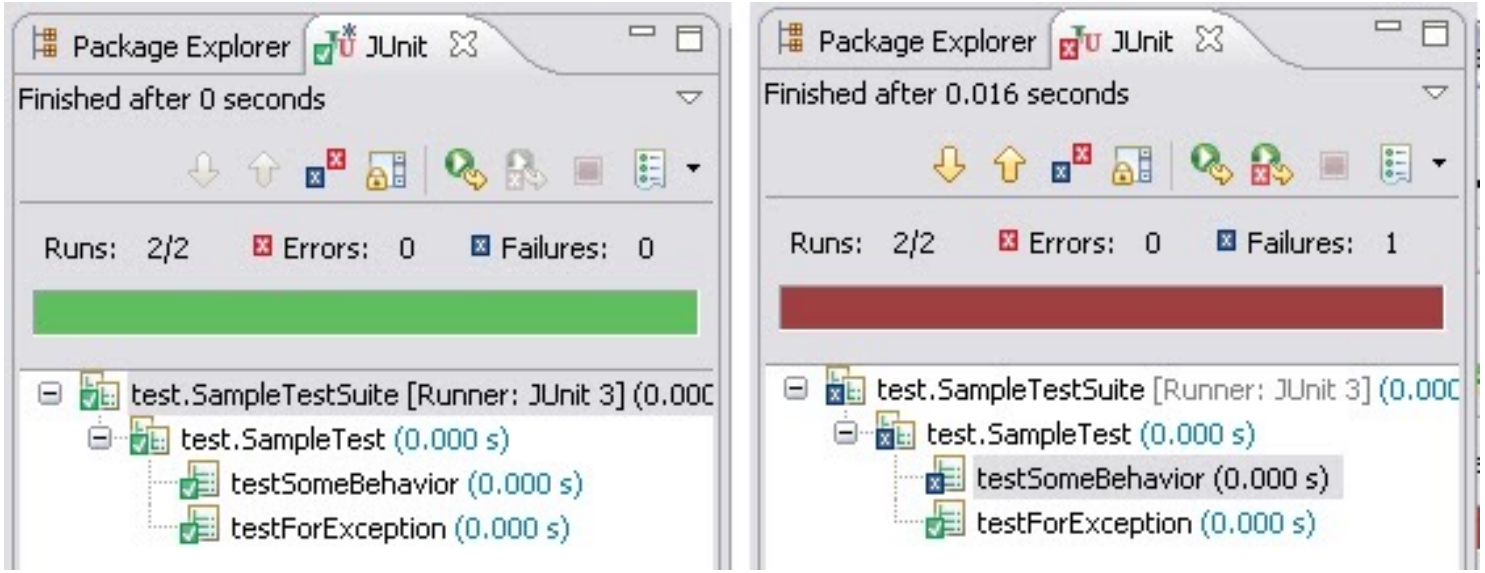
- Write a smaller test case that represents the broken part of the bigger test case. Get the smaller test case running. Reintroduce the larger test case.

- Continuous success >> Extra effort to maintain more tests.

- A test that is too big teaches a lesson. Why was it too big? What could I have done differently that would have made it smaller?

Context: Green Bar Patterns

- Fake it
  - How to implement a broken test?
- Triangulation
  - How to abstract from tests?
- Obvious implementation
  - How to implement simple operations?



Fake It

Problem

- What is your first implementation once you have a broken test?

Solution

- Return a constant. Once you have the test running, gradually transform the constant into an expression using variables.
- Having something running is better than not having something running

Triangulation

Problem

- How do you drive abstraction with tests?

Solution

- Abstract only when you have two or more examples.

Triangulation

Problem

- How do you drive abstraction with tests?

Solution

- Abstract only when you have two or more examples.

```
1 public void testSum() {
2     assertEquals(4, plus(3, 1));
3 }
4
```

```
1 private int plus(int augend, int addend) {
2     return 4;
3 }
4
```

Triangulation

Problem

- How do you drive abstraction with tests?

Solution

- Abstract only when you have two or more examples.

```
1 public void testSum( ) {
2     assertEquals(4, plus(3, 1));
3     assertEquals(7, plus(3,4));
4 }
```

```
1 private int plus(int augend, int addend) {
2     return augend + addend;
3 }
```

Obvious implementation

Problem

- How do you implement simple operations?

Solution

- Just implement them
- *Fake It* and *Triangulation* imply very small steps.
- Sometimes you know how to implement an operation

Outline of the talk

1. Test-driven Development
2. Acceptance Testing

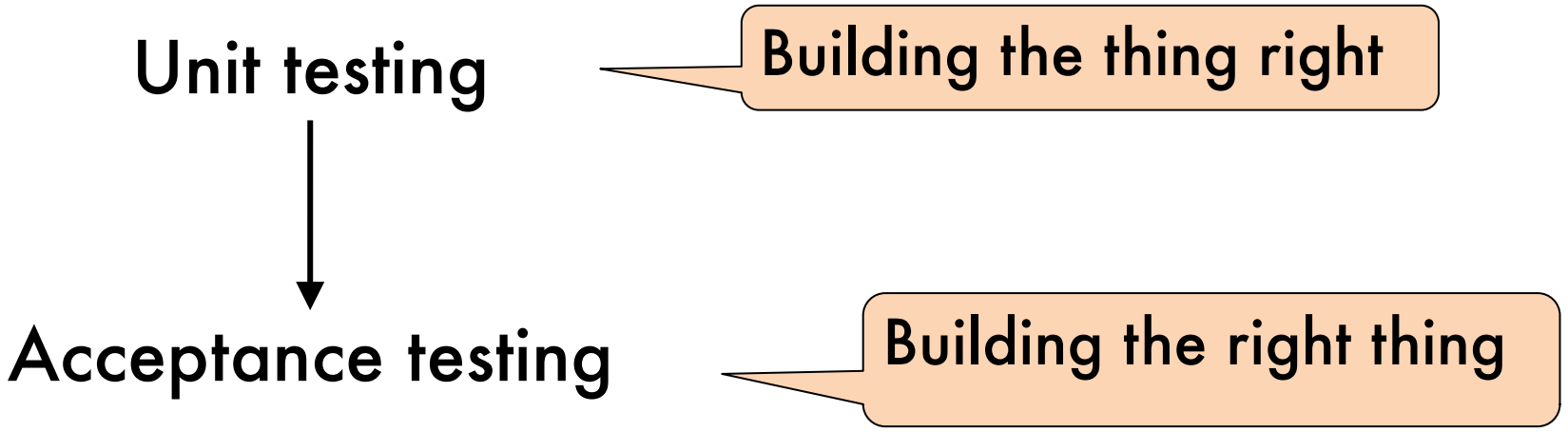
Acceptance testing: a definition

*"Formal testing with respect to **user needs**, requirements, and business processes conducted to determine whether or not a system satisfies the **acceptance criteria** and to enable the **user**, customers or other authorized entity to determine whether or not to **accept the system**"*

International Software Testing
Qualifications Board

Acceptance test

- Blurring the line between (parts of) RE and (parts of) testing
- Focus on *what* and not *how*
- Stories and Examples are written using the domain language
- Works well for precise and unambiguous requirements



Acceptance testing: a definition

"Formal testing with respect to
user needs, requirements, and business processes
determine
whether or not
a system satisfies the acceptance criteria and to enable the user, customers or other authorized entity
to determine whether or not to accept the system"

Is this "software under test"?

Is this the Oracle?

Green vs. Red bar?

Not developers anymore

International Software Testing Qualifications Board

Acceptance testing: closing the gap

- A "test" as a mean of communication between customers and developers
- Done "a priori" → the user has already expectations!
  - The customer describes a need to the developers in simple terms
  - Customer provides example of the behavior → Tests!
- (Semi-) automatically transform textual examples in actual code tests
- Leverages the concept of *user story*

User Stories as communication means

- Use of examples to describe the behavior of the system
- Written together by customers and developers
- Automated acceptance criteria

As a student
I want to access my grades
So *that* I do not need to disturb the teacher

The GWT template

Given I am logged in STiNe
When I press the "My courses" button
Then I should see the grades of my courses

Another way to express user stories

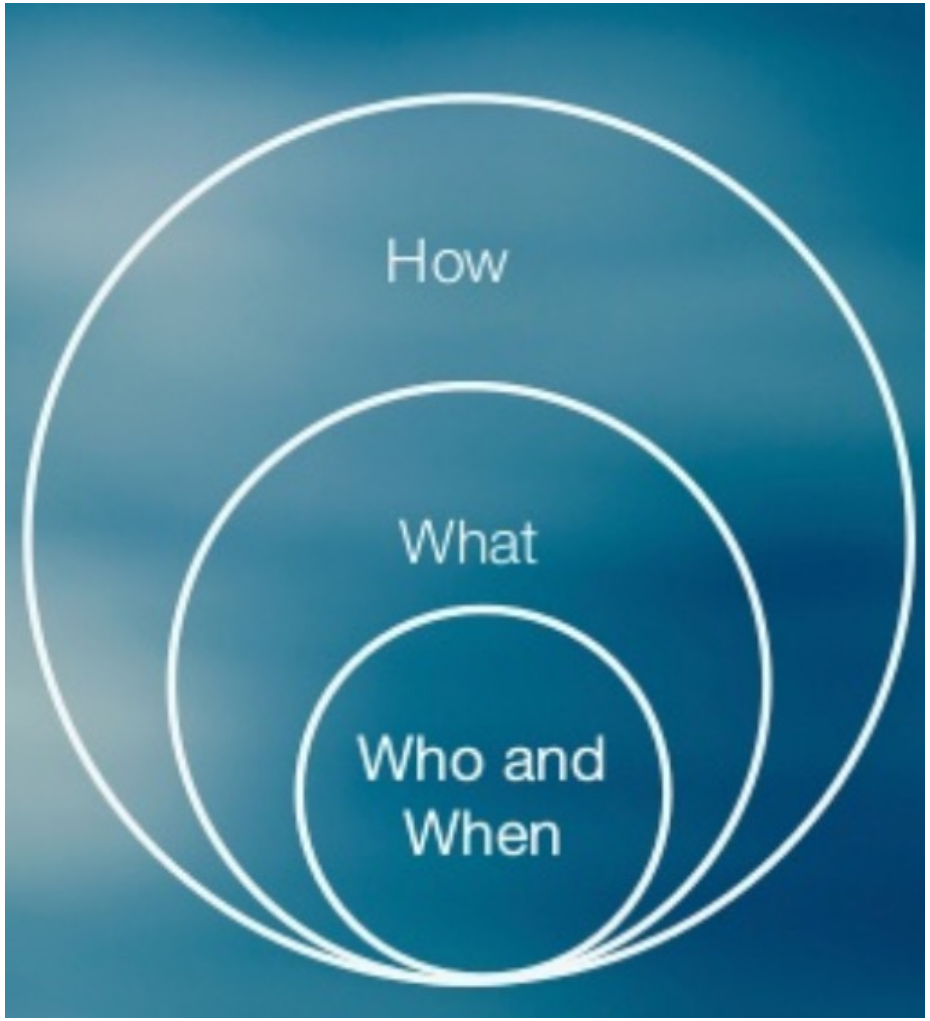
```
Feature: Login
  As a website user
  In order to access the website content
  I need to log in to the website

  Scenario: valid credentials
   Given I am on the login page
   When I provide the email address "test@mywebsite.com"
   And I provide the password "Foo!bar1"
   Then I should be successfully logged in

  Scenario Outline: invalid credentials
   Given I am on the login page
   When I provide the email address <email>
   And I provide the password <password>
   Then I should not be logged in

  Examples
    | email                | password |
    | test@mywebsite.com   | Foo!bar2 |
    | test@mywebsite.com   | Foo      |
```

Behavior Driven Development

- Focus on the when and who

Principles

- Built on top of ATDD
- Focuses on encouraging simple languages to be used across teams
- This simple language is also used with business persons
- Avoid “that’s not what I wanted” situations



Behavior-driven testing: an example

- BDD has support for several platforms
- Leverage the Gherkin Domain Specific Lang
- Here using JBehave

Scenario: A trader is alerted of status

Given a stock and a threshold of 15.0
When stock is traded at 5.0
Then the alert status should be OFF
When stock is traded at 16.0
Then the alert status should be ON

Scenario: A trader is alerted of status

Given a stock and a threshold of 15.0
When stock is traded at 5.0
Then the alert status is OFF
When stock is traded at 16.0
Then the alert status is ON

```
public class TraderSteps {
    private TradingService service;   // Injected
    private Stock stock; // Created

    @Given("a stock and a threshold of $threshold")
    public void aStock(double threshold) {
        stock = service.newStock("STK", threshold);
    }
    @When("the stock is traded at price $price")
    public void theStockIsTraded(double price) {
        stock.tradeAt(price);
    }
    @Then("the alert status is $status")
    public void theAlertStatusIs(String status) {
        assertThat(stock.getStatus().name(), equalTo(status));
    }
}
```

Summary

1. In Test-driven development, development is guided by test cases, for which there are green and red bar patterns

2. Acceptance Test checks whether the developed system fulfils the customers' expectations