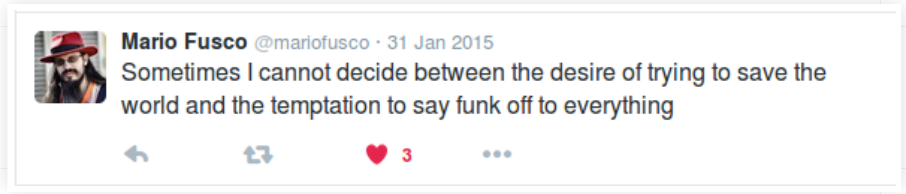
TDD MYTHS

Bartosz Mikulski

THE SPEAKER

@mikulskibartosz

- Java / Scala / sometimes JS ;)
- programmer for 5 years
- co-founder of Poznan Scala User Group meetup.com/poznanscala
- currently works at GFT



[1] https://twitter.com/mikulskibartosz

WHAT YOU MAY EXPECT

- common excuses
- bad practices
- personal experiences and opinions
- creepy stories about code
- blasphemy
- a few fails
- homework

WHAT IS ~~TESTING~~ TEST DRIVEN DEVELOPMENT?

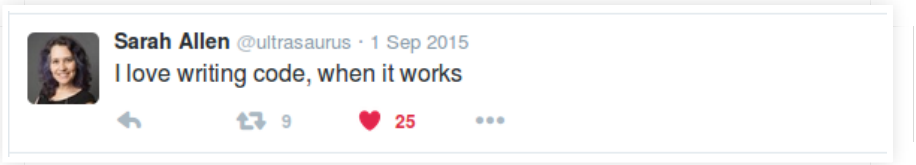
- a process that relies on a very short development cycle
- a discipline based on three laws:
  - You are not allowed to write any production code unless it is to make a failing unit test pass.
  - You are not allowed to write any more of a unit test than is sufficient to fail; and compilation failures are failures.
  - You are not allowed to write any more production code than is sufficient to pass the one failing unit test.
- a design tool: the red-green-**refactor** cycle

WHAT IS ~~TESTING~~
TEST DRIVEN DEVELOPMENT?

- writing tests == TDD ?
- do not know how to test == TDD does not work

WE DO NOT NEED TESTS AT ALL

does your software need to work correctly?



A QUICK REMINDER

"Never attribute to malice that which is adequately explained by stupidity."

Hanlon's razor

WE DO NOT CARE ABOUT QUALITY

(...) I was hired as a team leader for a safety critical embedded system that controlled a medical surgery device. (...)
We had been asked to finish a four year project in less than 1 year (...)
When we told our manager, he began to worry about his bonus. (...)
he began to cut my responsibilities.
He eventually gave full control of the project to the youngest and most impressionable programmer on my team. (...)
So I terminated my contract with the company.
A few days later the first bug occurred during a human trial. (...)
Even obvious bugs like mixing up the surgery directions (upwards/downwards) were found during the first human treatments.

Uncle Bob - The Letter

[1] https://blog.8thlight.com/uncle-bob/2012/01/12/The-Letter.html

EMBARRASSMENT

NO COMMENTS


Chris Williams
@diodesign
Follow
Programming bug sets free 3,200 crims from prison too soon in the US
seattletimes.com/seattle-news/p …
RETWEETS
7
LIKES
2
11:35 PM - 22 Dec 2015

... EVEN IF I TEST THE CODE, IT MAY HAVE SOME BUGS

- it is not a bulletproof process
- people make mistakes, but someone may review their code

Do you have a better idea?

THE TESTS GET OUTDATED

- only if YOU let them
- tests may BE your specification
- acceptance tests

FITNESSE

```
|eg.Division|
|numerator|denominator|quotient?|
|10       |2          |5        |
|12.6     |3          |4.2      |
|100      |4          |33       |
```

```
public class Division {
  private double numerator, denominator;
(...)
  public double quotient() {
    return numerator/denominator;
  }
}
```

Fitnesse user guide - Two minute example

[1] http://www.fitnesse.org
/FitNesse.UserGuide.TwoMinuteExample

CUCUMBER

```
Feature: Refund item

  Scenario: Jeff returns a faulty microwave
  Given Jeff has bought a microwave for $100
  And he has a receipt
  When he returns the microwave
  Then Jeff should be refunded $100
```

```
public class MyStepdefs implements En {
  public MyStepdefs() {
    Given("Jeff has bought a microwave for $(\\d+)",
    (Integer dolars) -> {
      System.out.format("Price: $%n\n", dolars);
    });
  }
}
```

cucumber.io

[1] https://cucumber.io/docs

THE TESTS GET OUTDATED

- only if YOU let them
- tests may BE your specification
- acceptance tests
- www.fitnesse.org
- cucumber.io
- "It is a silly idea, stakeholders will not write tests!" I hope so!
- it is a responsibility of developers + QA

I DO NOT NEED TO TEST EVERYTHING

- that code is too simple, I know it works
  So it should be easy to test!
- "That code does not do anything, it is a middle-man calling another application and returning the result."
  Write an integration test!

IT IS NOT POSSIBLE TO TEST EVERYTHING

- do not test UI
- if it is an experiment, you can write less tests
- but if your company usually releases POCs as production-ready software, always test it

THE CLIENT DOES NOT PAY FOR TESTS

- the client pays for working software
- the client does not want the bugs back
- the client pays for updates
- ... and wants them now!

I DO NOT KNOW WHAT IT IS SUPPOSED TO DO

HOW CAN I TEST IT?

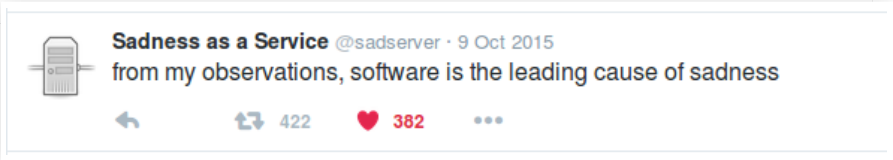
- lack of requirements
- lack of details
- write everything down!
- frequent changes
  "It is not a mess, it is a dynamic landscape."
- UI driven requirements
  "What was the artist thinking?"
- Wetware issues

IF I TESTED THE CODE, I COULDN'T REFACTOR IT

- tests that check whether a few methods were called in a specified order
- ~~over-mocking~~ over-stubbing
  (but I will call it a mock anyway...)"
- are you going to test mocks?
- are you sure it behaves like a real class?
- if it works, who will know about it?
- who will spot an error?

IF I TESTED THE CODE, I COULDN'T REFACTOR IT

- exposing implementation details in order to write a test
- how would you test the java.util.HashSet?



HAPPY MOCKER WAY

```
//given
HashSet<Long> objectUnderTest = ...
    //create a HashSet containing a mocked internal map
InOrder mockCallOrder = inOrder(theMock);

//when
objectUnderTest.add(1);
objectUnderTest.add(2);
objectUnderTest.add(1);

//then
mockCallOrder.verify(theMock).put(eq(1), any(Object.class));
mockCallOrder.verify(theMock).put(eq(2), any(Object.class));
mockCallOrder.verify(theMock).put(eq(1), any(Object.class));
```

TEST THE BEHAVIOUR!

```
@Test
public void shouldStoreUniqueValues() {
    //given
    HashSet<Long> objectUnderTest = new HashSet<>();

    //when
    objectUnderTest.add(1);
    objectUnderTest.add(2);
    objectUnderTest.add(1);

    //then
    assert(objectUnderTest).hasSize(2);
    assert(objectUnderTest).containsOnly(1, 2);
}
```

IF I TESTED THE CODE, I COULDN'T REFACTOR IT

- mock code accessing external resources
  (databases, web services, files)
- do not assert every single detail
- if it works, who will know about it?

... BUT YOU TEST SOME CLASSES MORE THAN ONCE!

- is it wrong?
- do I use them more than once?
- if a class depends on another one, its test should fail if the dependency behaves differently and the class has not been modified.

... BUT YOU TEST MORE THAN ONE CLASS!

- it is not a unit test
- a class is not a unit
- test your functional interfaces / functions separately
- and then test them again as a part of another type

... BUT IT IS NOT A UNIT TEST!

- so what?
- is a use case a unit?
- test user stories, use cases
- single responsibility principle
- write unit tests to verify error handling / validation / "alternative execution paths" / "functions"

... BUT THERE SHOULD BE ONLY ONE ASSERTION

- is assertEquals one assertion?
- write your own assertions
- one reason to change, one reason to fail
- it is like the pirate code, more what you call guidelines, than actual rules

TESTS ARE SLOW

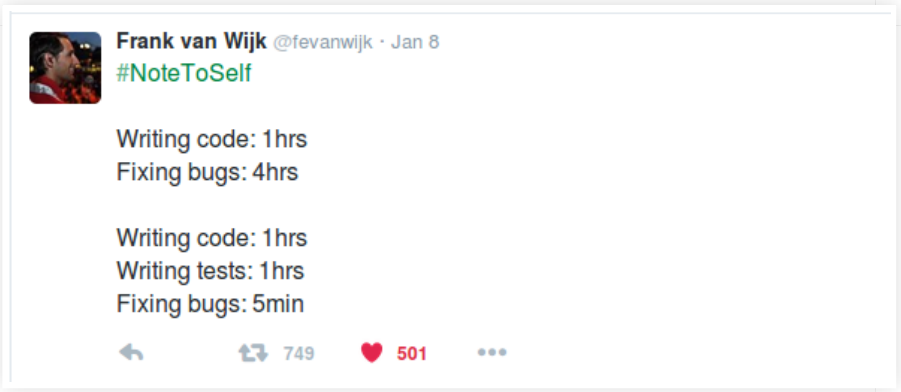
- don’t start your application
- say no to IO
- single thread
- no waiting / blocking

```
synchronized {
  wait(n); == something went wrong
}
```

TDD IS SLOW

- start with the easiest test
- plan all tests you need to write
- it is like learning to walk
- you will be slower at the beginning

... BUT DEADLINE!



TDD IS DIFFICULT

- everything is difficult
- you can learn it
- start with something easy
- but not 3 days before deadline!
- you can NOT learn by reading or watching other people doing it

IT IS HARD TO TEST LEGACY CODE

- everything you need is "Working effectively with legacy code"
- sometimes code is test-hostile
- you can always test something
- that is the difficult part of testing

TESTS ARE FRAGILE

- fail for “no reason”
- a test should not depend on other tests
- it should be possible to run all tests in random order

SOME PROGRAMMERS SHOULD WRITE TESTS

"Junior programmers can write tests, I am a senior engineer, so I do not need it."

No, you are not

QA SHOULD TEST CODE

- QA responsibilities: exploratory testing, usability tests
- never create a test script

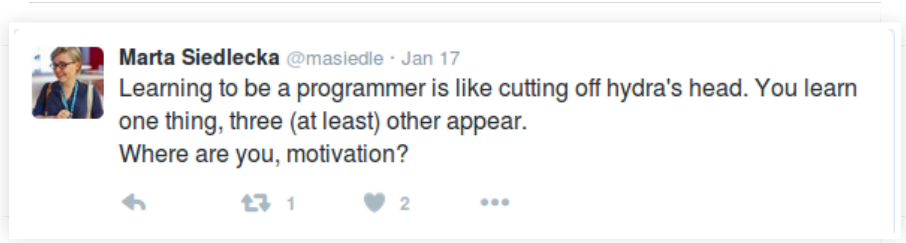
WE NEED TO INCREASE TEST COVERAGE

- 100%
- high code coverage == high code quality? No!
- mutation tests

```
@Test //high code coverage and no value!
public void shouldDoSthIDontCareAbout() {
  try {
    //a lot of code
    //no assertions
  } catch (Exception ex) {}
}
```

HOW TO DO IT?

- code katas - good for beginners
- code reviews
- side projects - no deadline, no manager, no problem
- maciejaniserowicz.com/daj-sie-poznac
- coding dojo - The global day of code retreat
- pair programming
  do not be a backseat driver!



SUMMARY

- test an interface (contract), not the implementation
- do not over-mock, but mock external resources
- independent tests
- do not be afraid to test something more than once
- the only excuse against TDD are bad tests

RESOURCES

- Clean code - Robert C. Martin
- Clean coder - Robert C. Martin
- Working effectively with legacy code - Michael Feathers
- Google tech talks
  The clean code talks: Inheritance, Polymorphism, & Testing, Global state and singletons, Don't look for things, Unit testing
  Google design tech talks: OO design for testability
- blog.8thlight.com

[1] https://www.youtube.com/watch?v=4F72VULWFvc
[2] https://www.youtube.com/watch?v=-FRm3VPhseI
[3] https://www.youtube.com/watch?v=RlfLCWKxHJ0
[4] https://www.youtube.com/watch?v=wEhu57pih5w
[5] https://www.youtube.com/watch?v=acjvKJiOvXw

HOMEWORK

- read / watch the resources
- write a code kata every day for a week
- write minesweeper following the described practices
- prove I am wrong, find three examples
- send me your feedback:
  @mikulskibartosz

IT WAS NOT ABOUT TDD

- you can explain TDD in 2 minutes
- TDD is not your problem...
- ... tests are
- TDD == writing tests + a few rules

QUESTIONS?


Bored Elon Musk @BoredElonMusk · 12 Nov 2015
People Who Will Ask Questions At The End Of A Panel To Avoid Awkward Silences as a Service.
271
548