Testing Upside Down

Your tests may be broken

Jiří Jakeš

February 15, 2017

Scala Taiwan Meetup

Warm-up

```scala
object Math {
  def average(nums: Seq[Int]): Double =
    nums.sum.toDouble / nums.size
}

class MathSpec extends FlatSpec {

  "Average" must "be correct" in {
    Math.average(List(1, 2, 3)) must be(2.0)
  }

}
```

Conclusions

Conclusions

1. Unit tests don’t prove correctness of implementation

Conclusions

1. Unit tests don’t prove correctness of implementation
2. Code coverage target makes code worse

Conclusions

1. Unit tests don’t prove correctness of implementation
2. Code coverage target makes code worse
3. More integration tests make code less trustworthy

Conclusions

1. Unit tests don’t prove correctness of implementation
2. Code coverage target makes code worse
3. More integration tests make code less trustworthy
4. Mocks are sign of poor design

Unit Tests & Correctness

Unit Test

A unit test is an automated test that tests a unit in isolation from its dependencies.

Unit Test

Unit tests serve only purpose:

Unit Test

Unit tests serve only purpose: detecting regressions.

Unit Test

Unit tests serve only purpose: detecting regressions.

They show that code is as correct as correct it was at first.

How About Correctness?

Unit tests can only show correctness if they are written correctly.

How About Correctness?

Unit tests can only show correctness if they are written correctly.

Unit tests of unit tests? Unit tests of unit tests of unit tests?

In Unit Tests We Trust

1. Easy review
    - Simple code
    - Deterministic
2. See test fail
3. Append only

In Unit Tests We Trust

1. Easy review
   - Simple code
   - Deterministic
2. See test fail
3. Append only

Tests are trustworthy if they are simple and you’ve seen them fail.

Code Coverage

Code Coverage

Good tool to see which parts are tested and which are not.

Code Coverage

Good tool to see which parts are tested and which are not.

Bad tool as a measure/criterion/goal. Targeting a specific percentage is counter-productive.

Code Coverage Percentage

It is no big problem to have 100 % coverage.

However, it does not say anything about *quality* of tests.
Code coverage has no correlation with code quality.

Code Coverage Percentage As Goal

Makes you waste energy on things that rarely go wrong.

Distracts you from testing things that really matter.

Code Coverage And Psychology

Lack of time and minimum code coverage is a dangerous combination. In order to meet percentage, people tend to do it by writing less meaningful tests.

Code Coverage And Psychology

If some improvement requires adding more code (and thus decreasing code coverage), human brain tends not to do it in order to avoid writing new tests.

If a part of your test suite is weak in a way that coverage can detect, it's likely also weak in a way coverage can't detect[1].

[1]Brian Marick: *How to Misuse Code Coverage*, 1997

Integration Tests

Integration Test

While unit tests focus on testing in isolation, integration tests work with clusters of units verifying their collaboration.

Cyclomatic Complexity (循環複雜度)

Cyclomatic complexity $M \in \mathbb{N}$ indicates complexity of a program. It is a number of independent paths through source code, roughly equals to number of loops and ifs $+ 1$.

Cyclomatic Complexity (循環複雜度)

Cyclomatic complexity $M \in \mathbb{N}$ indicates complexity of a program. It is a number of independent paths through source code, roughly equals to number of loops and ifs $+ 1$.

It determines number of tests needed to achieve coverage of a module.

Integration Tests & Maths

How many tests we need to write if:

… a module (e. g. function) has $M = 3$?

Integration Tests & Maths

How many tests we need to write if:

… a module (e. g. function) has $M = 3$?
… a module has $M = 5$?

Integration Tests & Maths

How many tests we need to write if:

… a module (e. g. function) has $M = 3$?
… a module has $M = 5$?
… a module has $M = 8$?

Integration Tests & Maths

How many tests we need to write if:

… a module (e. g. function) has $M = 3$?
… a module has $M = 5$?
… a module has $M = 8$?

… if we want to test integration of these three modules?

Integration Tests & Maths

How many tests we need to write if:

… a module (e. g. function) has $M = 3$?
… a module has $M = 5$?
… a module has $M = 8$?

… if we want to test integration of these three modules?

$$M = 3 \cdot 5 \cdot 8 = 120$$

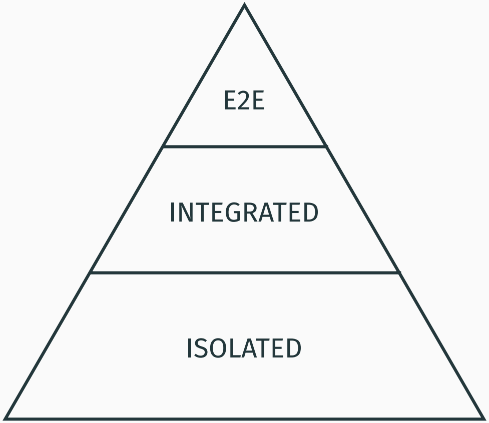
Integration Tests & Maths

In other words, in order to thoroughly test integration of n modules, we need to write N tests where:

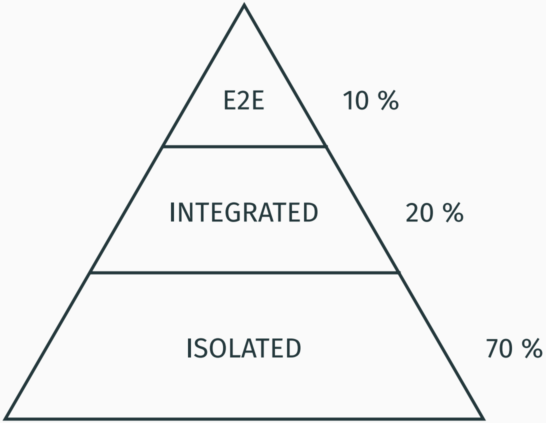
$$N = \prod_{i=1}^{n} M_i$$

That is **product** of cyclomatic complexities of all modules.

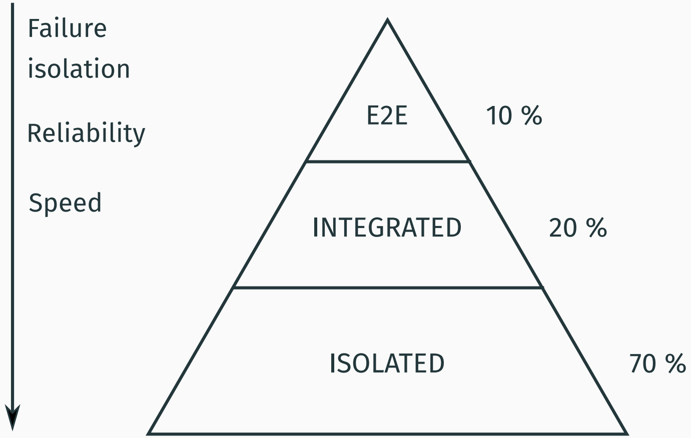
Test Pyramid



Test Pyramid



Test Pyramid



Integration Tests Danger

Necessity to write too many integration tests is a sign of design weaknesses.

When a bug is found and we write more integration tests to prevent similar bugs in the future, we are ignoring the weaknesses.

Solution is **not** to write more integration tests but less and focus on design problems.

Focus on writing unit tests. They will often guide you towards better design.

Test Doubles

Test Doubles

Dummy	Fill-in for parameters that are not needed
Stub	Provides prepared (hard-coded) *input* for tested code
Mock	Verifies *output* of tested code by specifying expectations, e. g. *when method1 is called, expect method2 be called with parameter ”hello”*
Fake	Models complex functionality using simpler implementation, e. g. map in memory instead of database

Problems With Test Doubles

- More code to maintain (who tests mocks?)
- Dependencies on implementation details
- Refactoring hell
- Distract attention from real problem

Problems With Test Doubles

In general, when you feel mock/fake is the only way how to test code, it probably means your code is coupled too tightly and you are mixing *business logic* and *execution*.

Tight Coupling

```
def updateUserAge(id: Long): Unit = {
  val user: User = db.getUser(id)
  val realAge =
       Period
        .between(user.birthdate, LocalDate.now())
        .getYears
  if (realAge > user.age) {
      db.updateUser(user.withAge(realAge))
  }
}
```

Consequences

Mixing business logic and execution leads to:

- Lower reusability
- More difficult to change
- Harder to reason about
- Harder to test properly

Solution

Solution is to separate business logic and execution in two functions. Business logic would ideally be pure. Execution would contain no decisions.

Model Example

Story With Moral

Situation in a big company:

Story With Moral

Situation in a big company:

New task arrived – monthly reports will be printed on new printers that can also dispatch letters.

Story With Moral

Situation in a big company:

New task arrived – monthly reports will be printed on new printers that can also dispatch letters.

Requirement: the printed envelopes must have return address in correct format – if letter goes abroad, return address must be in international format.

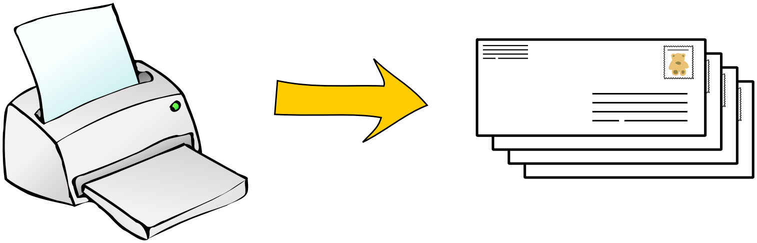
12345
臺北市信義區信義路1號1樓
我們最棒有限公司

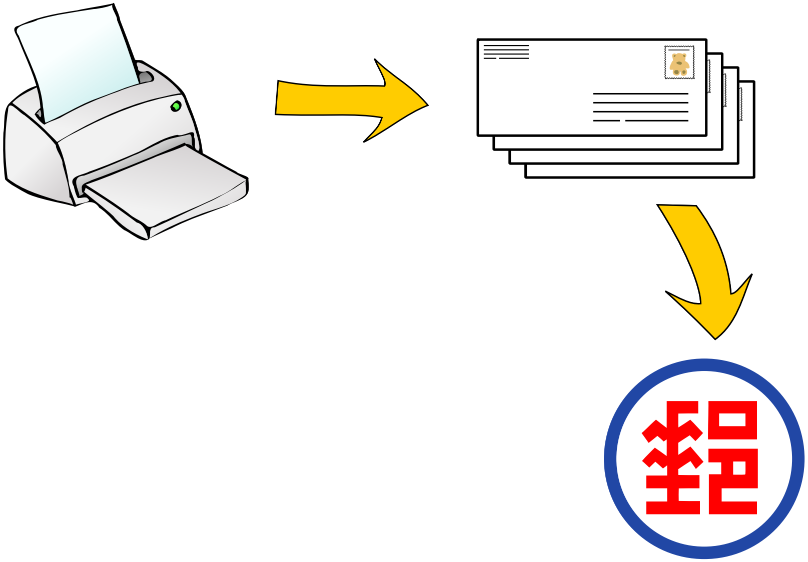
11255
臺北市內湖區中山路25號4樓
王小明 先生收

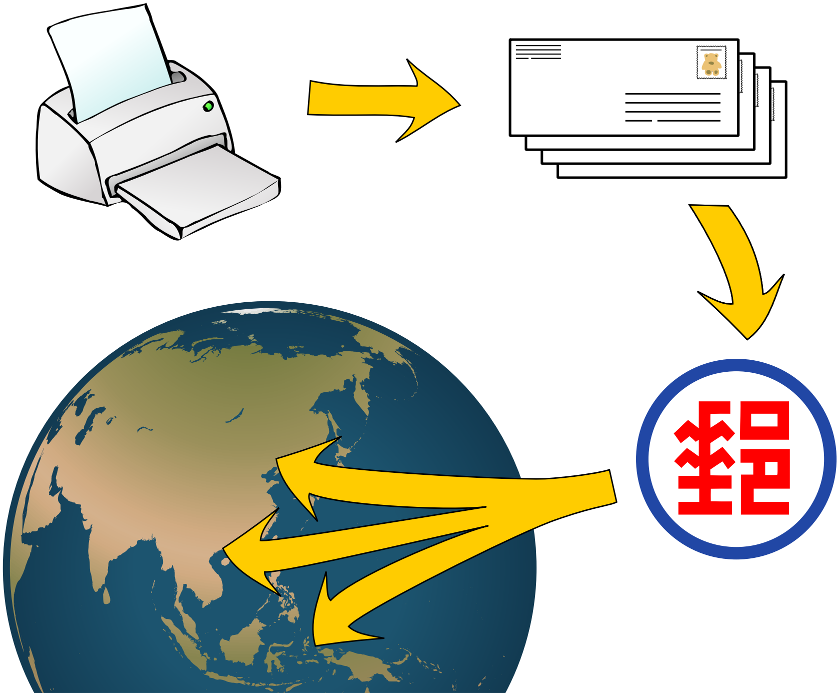
Our Best Company Ltd.
1F, No. 1 Hsin-I Road
12345 Hsin-I District, Taipei City
TAIWAN

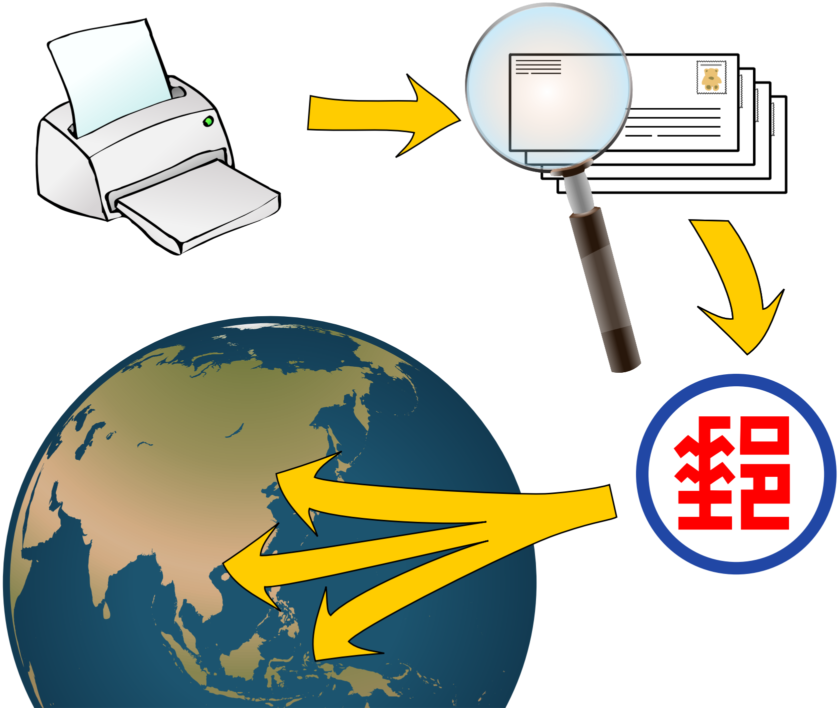
John Smith
321 N Washington Blvd Apt 15
Smallville, TX 151000-0120
UNITED STATES OF AMERICA

```scala
class PrintService(mail: MailDeliveryService) {
  def printAndSend(addr: Address, letter: Letter): Unit = {
    val env = new Envelope()
    this.printAddress(env, addr)
    if(addr.isInternational)
      this.printReturnAddressInt(env) // test that address
    else                              // in correct format
      this.printReturnAddress(env)
    env.insertLetter(letter)
    mail.send(env)
  }
}
```









How Can We *Unit Test* Return Address?

How Can We *Unit Test* Return Address?

Idea: Send 100 envelopes to incorrect addresses,
wait and then count!

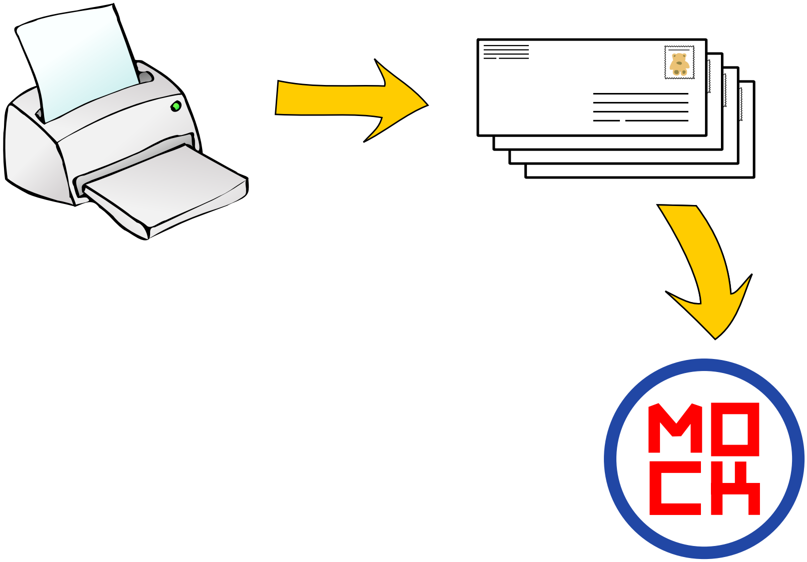
How Can We *Unit Test* Return Address?

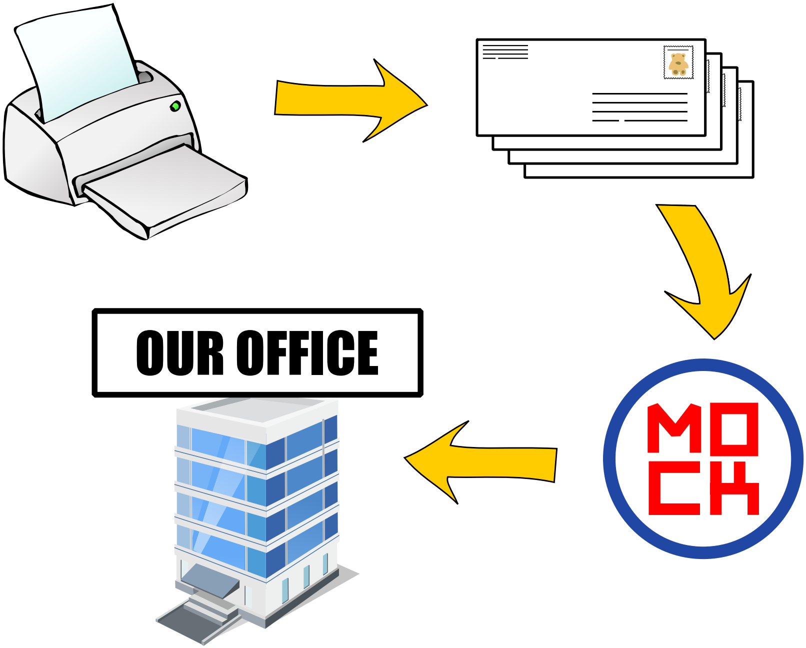
Idea: Send 100 envelopes to incorrect addresses,
wait and then count!

Kind of. . . weird.
But testing with real resources (e. g. database) very common.

Testing With Real Resources

- Sloooow
- Hard parallel testing
- Cannot run everywhere
- Testing someone else’s code
- Does not focus on business logic
- Sign of tight coupling
- Potentially expensive (cloud)


MOCK


OUR OFFICE
MOCK

Testing With *Doubles*

- Writing and maintaining doubles
- Dependency injection
- Defining abstraction
- Difference in implementations, subtle details
- Does not focus on business logic
- Sign of tight coupling

```scala
class PrintService(mail: MailDeliveryService) {
  def printAndSend(addr: Address, letter: Letter): Unit = {
    val env = new Envelope()
    this.printAddress(env, addr)
    if(addr.isInternational)
      this.printReturnAddressInt(env)
    else
      this.printReturnAddress(env)
    env.insertLetter(letter)
    mail.send(env)
  }
}
```

```scala
class PrintService(mail: MailDeliveryService) {
  def printAndSend(addr: Address, letter: Letter): Unit = {
    val env = new Envelope()
    this.printAddress(env, addr)
    if(addr.isInternational)
      this.printReturnAddressInt(env)
    else
      this.printReturnAddress(env)
    env.insertLetter(letter)
    mail.send(env)
  }
}
```

```scala
class PrintService(mail: MailDeliveryService) {
  def printAndSend(addr: Address, letter: Letter): Unit = {
    val env = new Envelope()
    this.printAddress(env, addr)
    if(addr.isInternational)
      this.printReturnAddressInt(env)
    else
      this.printReturnAddress(env)
    env.insertLetter(letter)
    mail.send(env)
  }
}
```

Does Not Focus On Business Logic?

We are focusing on the wrong place – printing/sending. But we are testing whether return address is always set and correct.

In fact, we don’t need any `MailDeliveryService` to test it. We even don’t need printer.

```scala
case class PrintRequest(
                to: Address, from: Address, letter: Letter)
```

```scala
case class PrintRequest(
                to: Address, from: Address, letter: Letter)


def makeRequest(to: Address, l: Letter): PrintRequest = ???
```

```scala
case class PrintRequest(
                to: Address, from: Address, letter: Letter)


def makeRequest(to: Address, l: Letter): PrintRequest = ???


class PrintService(mail: MailDeliveryService) {
  def printAndSend(pr: PrintRequest): Unit = {
    val env = new Envelope()
    this.printAddress(env, pr.to)
    this.printReturnAddress(env, pr.from)
    env.insertLetter(pr.letter)
    mail.send(env)
  }
}
```

General Good Practice

Feedback

Rapid feedback is very important in agile development. We want to get feedback about our code as soon as possible to save time (and money!).

1. **Compilation**
2. **Static code analysis** – Wartremover and similar
3. **Unit tests** – must be fast and easy to run
4. **Integration tests**
5. **System testing** – feedback slow, may take days

Compilation As Feedback

Make use of strong type system.

Algebraic Data Type

```scala
sealed trait Shape
case class Line(length: Int) extends Shape
case class Triangle(a: Int, b: Int, c: Int) extends Shape
case class Square(side: Int) extends Shape
case object Dot extends Shape
```

- Effective pattern matching
- Exhaustiveness check
- Perfect as type of function’s result

Compilation As Feedback

Make use of strong type system.

Value Classes

```scala
class Length(val value: Double) extends AnyVal
class Volume(val value: Double) extends AnyVal
```

- Zero runtime overhead
- Type-safe (cannot assign length to volume)
- Self-documenting

Compilation As Feedback

Make use of strong type system.

Refined Types

```
case class Person(names: NonEmptyList[NonEmptyString],
                  age: NonNegInt)

case class Bet(amount: IntMin100)
```

Property Based Tests

Use Scalacheck. Always.

Resources

!!! http://blog.ploeh.dk/tags/
!!! http://xunitpatterns.com/
https://testing.googleblog.com/
http://rea.tech/to-kill-a-mockingtest/
Mocks & Stubs: https://www.youtube.com/watch?v=EaxDl5NPuCA
https://martinfowler.com/articles/mocksArentStubs.html
https://en.wikipedia.org/wiki/Cyclomatic_complexity
https://martinfowler.com/bliki/TestCoverage.html
http://www.exampler.com/testing-com/writings/coverage.pdf
http://david.heinemeierhansson.com/2014/test-induced-design-damage.html
Integrated Tests Are Scam: https://www.youtube.com/watch?v=VDfX44fZoMc