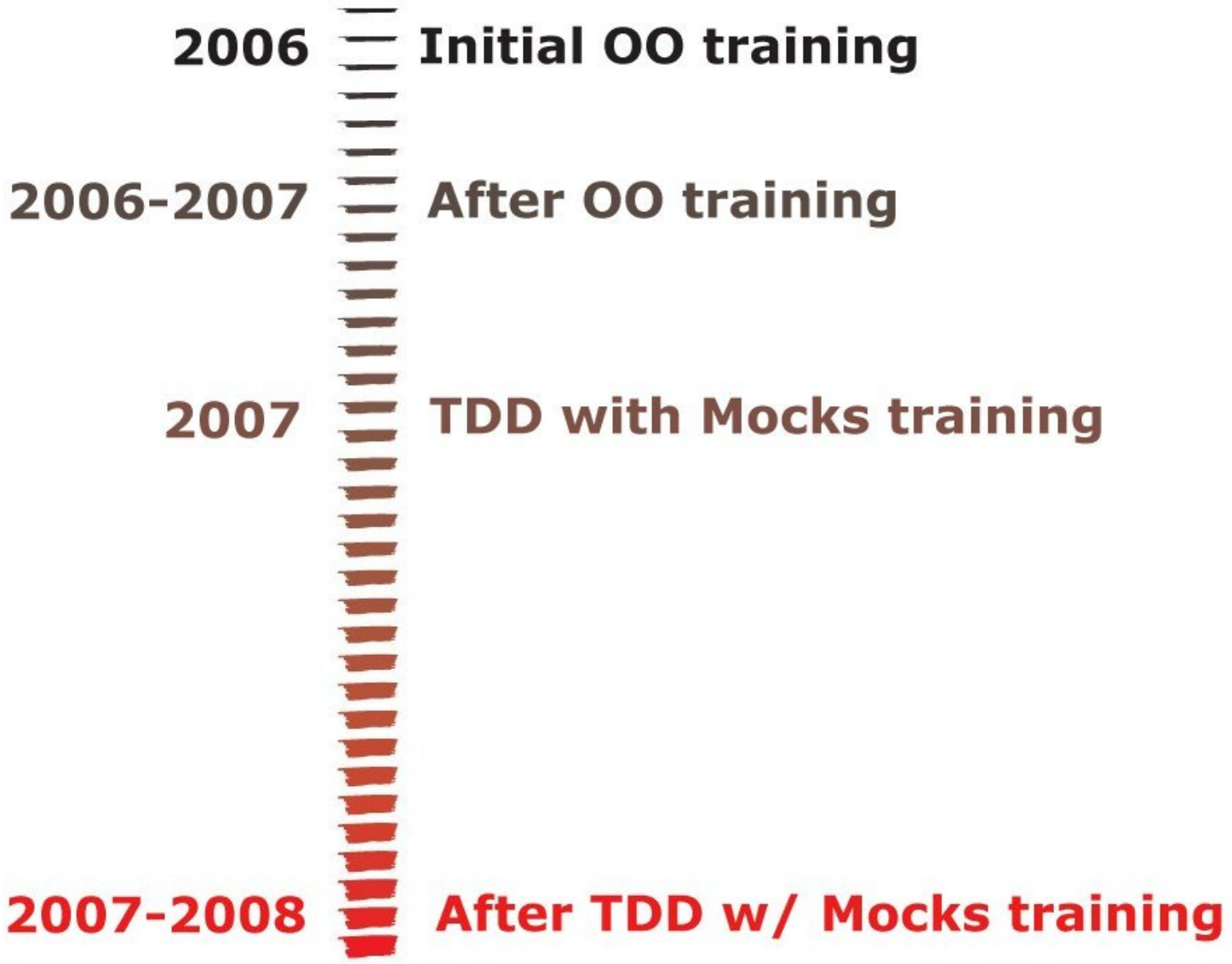
Luca Minudel

Exploratory study

based on :

- what observed in this experience
- data collected from experiments since now


2006
Initial OO training
2006-2007
After OO training
2007
TDD with Mocks training
2007-2008
After TDD w/ Mocks training

Alan Turing:

«

The popular view that scientists proceed inexorably from well-established fact to well-established fact, never being influenced by any improved conjecture, is quite mistaken

*Provided it is made clear which are **proved facts** and which are **conjectures**, no harm can result*

*Conjectures are of great importance since they **suggest useful lines of research***

»

DESIGN ~ TDD ?

Endo-Testing: Unit Testing with Mock Objects

Tim Mackinnon (Connextra), Steve Freeman (BBST), Philip Craig (Independent)
(tim.mackinnon@pobox.com, steve@m3p.co.uk, philip@pobox.com)

This paper was presented at th
Software Engineering - XP20
be published in *XP eXamined*

Abstract

Unit testing is a fundamental
difficult to test in isolation. It
and difficult to maintain and i
domain code and test suites. T
structure, and avoid polluting

Keywords: Extreme Programm

1 Introduction

"Once," said the Mo

Unit testing is a fundamental
trivial code is difficult to test
time, and you want to be notif
because you are trying to test

We propose a technique calle
implementations that emulate
code which they test from insi
writing code stubs with two i
is usual, and we use our tests

Our experience is that develo
better structure of both domai
regular format that gives the d
should be written to make it e
technique to achieve this. We
cost of writing stub code.

In this paper, we first describe
the benefits and costs of Mock
brief pattern for using Mock

2 Unit testing with Mo

An essential aspect of unit tes
you are testing and where any
simply and clearly as possible

Mock Roles, not Objects

Steve Freeman, Nat Pryce, Tim Mackinnon, Joe Walnes
ThoughtWorks UK
Berkshire House, 168-173 High Holborn
London WC1V 7AA
{sfreeman, npryce, tmackinnon, jwalnes} @thoughtworks.com

ABSTRACT

Mock Objects is an extension to Test
supports good Object-Oriented design
a coherent system of types within a c
less interesting as a technique for isol
libraries than is widely thought. This p
of using Mock Objects with an extende
and worst practices gained from ex
process. It also introduces jMock, a Jav
our collective experience.

Categories and Subject Des
D.2.2 [**Software Engineering**]: Desig
Object-Oriented design methods

General Terms
Design, Verification.

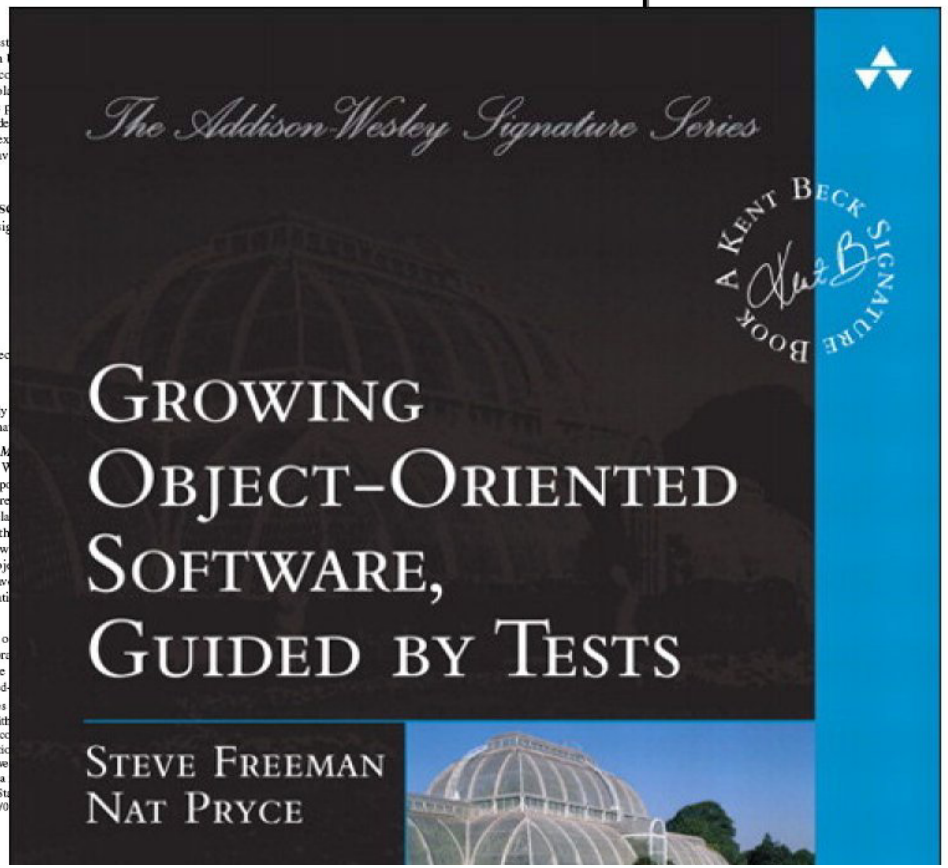
Keywords
Test-Driven Development, Mock Objec

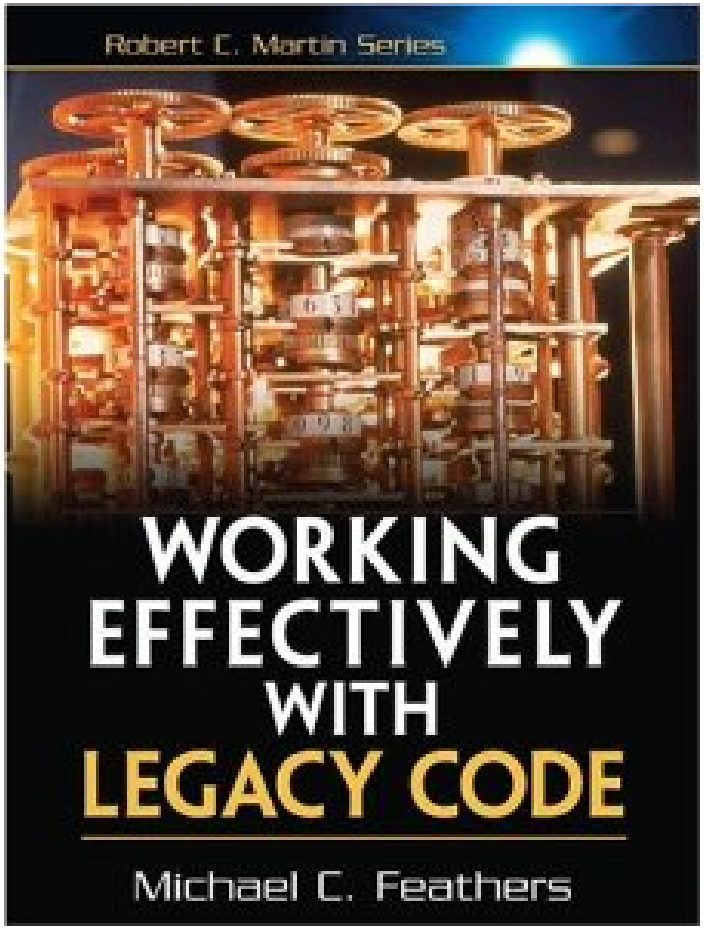
1. INTRODUCTION

Mock Objects is misnamed. It is really
types in a system based on the roles tha

In [10] we introduced the concept of *M*
to support Test-Driven Development. W
better structured tests and, more imp
code by preserving encapsulation, re
clarifying the interactions between cla
how we have refined and adjusted th
experience since then. In particular, w
most important benefit of Mock Obj
called "interface discovery". We hav
framework to support dynamic generati
on this experience.

The rest of this section establishes o
Driven Development and good pra
Programming, and then introduces the
rest of the paper introduces Need-





- Parametrize Constructor
- Parametrize Method
- Extract Interface
- Introduce Instance Delegator
- Skin and Wrap the API
- Responsibility-Based Extraction
- Adapt Parameter
- …

No technique can survive inadequately trained developers - Steve Freeman

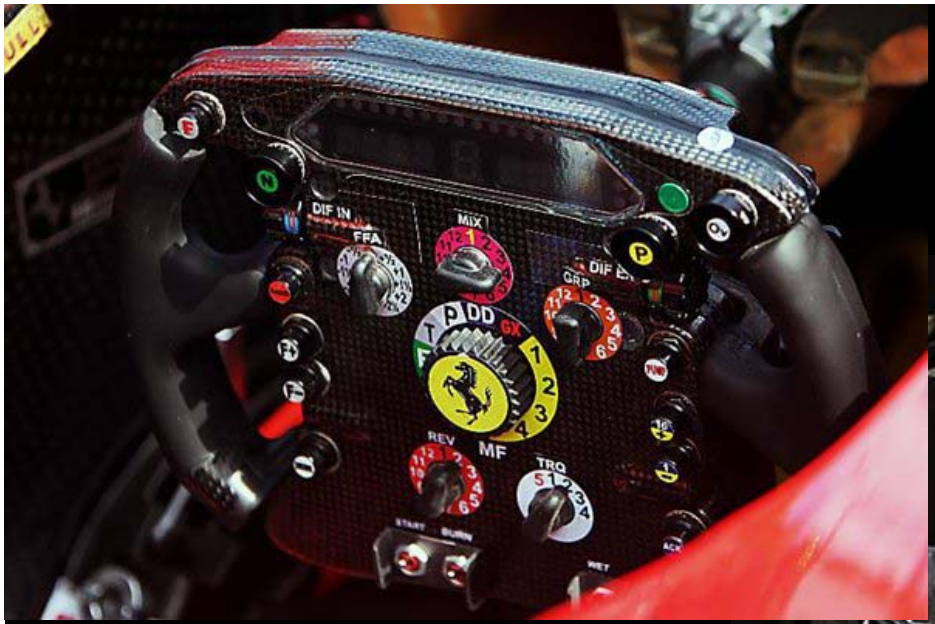
1st Law of Software Process:

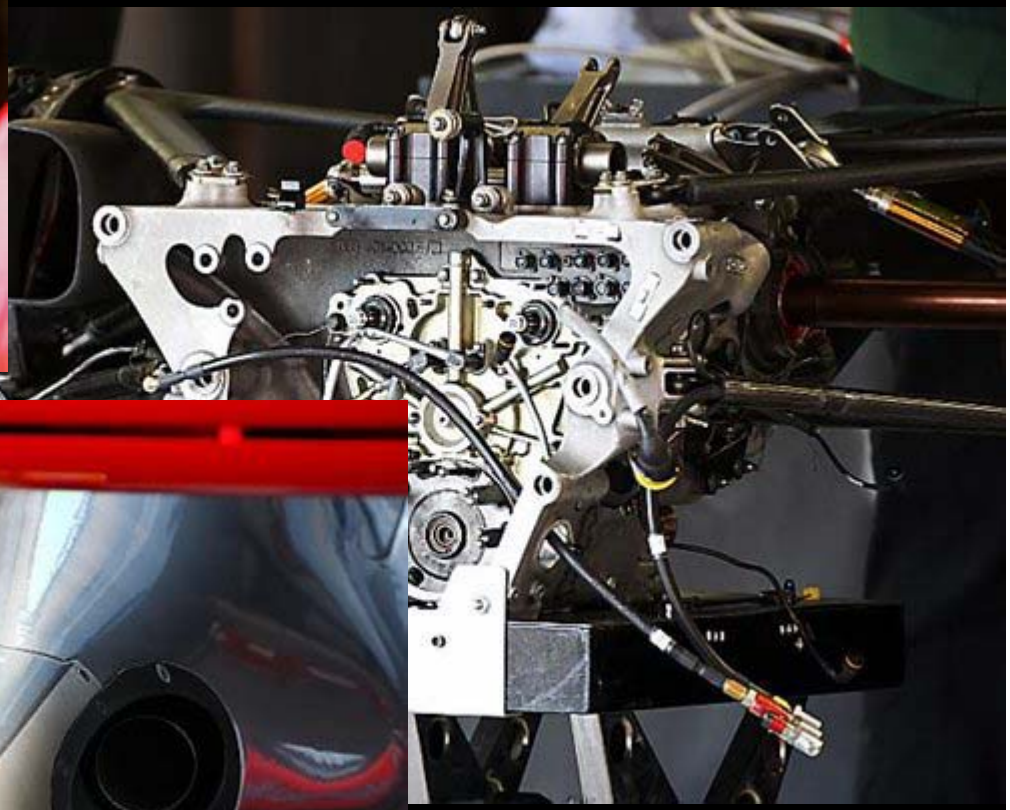
Process only allows us to do things we already know how to do - Philip G. Armour

TDD does not drive towards good design, it drives away from a bad design. If you know what good design is, the result is a better design - Nat Pryce

TDD doesn't drive good design. TDD gives you immediate feedback about what is likely to be bad design - Kent Beck

All science is experiential; but all experience must be related back to and derives its validity from the ***conditions and context*** *of consciousness in which it arises, i.e., the totality of our nature* - Wilhelm Dilthey


DIF IN
FFA
MIX
DIF
GRP
REV
MF
TRQ




BRIDGESTONE
BRIDGESTONE

Social Complexity

As soon as you introduce people, things become complex - Joseph Pelrine

Software Systems Evolution

To the degree that a software system is large and distributed enough that there is no effective single point of control, we must expect evolutionary forces

…

The strategies that we adopt to understand, control, interact with, and influence the design of computational systems will be different once we understand them as ongoing evolutionary processes

D. H. Ackley et al. 2002

Terms check: mock ?

- **Test Double**
  In automated **unit testing** replaces an object on which the class under test depends

  Can be a **Stub**, a **Mock** a **Spy**

Terms check: mock ?

- **Test Stub**
  Has configurable canned responses.
  Is used to control the indirect input to the class under test

Terms check: mock ?

- **Strict Mock**
  Has configurable expectations.

  Test fails when expected msg is not sent.
  Test fails when unexpected msg is sent.

  Used for verifying messages sent by the class under test

Terms check: SOLID ?

- **SRP – Single Responsibility Principle**

  there should never be more than one reason for a class to change: a class should have one and only one responsibility

Terms check: SOLID ?

- **ISP – Interface Segregation Principle**

  clients should not be forced to depend upon interface members that they don't use: interfaces that serve only one scope should be preferred over fat interfaces.

Terms check: SOLID ?

- **OCP – Open Closed Principle**

  classes and methods should be open for extensions and strategically closed for modification:

  so that the behavior can be changed and extended adding new code instead of changing the class

Terms check: SOLID ?

- **DIP – Dependency Inversion Principle**

  low level classes and high level classes should both depend on abstractions: high level classes should not depend on low level classes.

Terms check: SOLID ?

- **LSP – Liskov Substitution Principle**

  methods that use a base class must be able to use instances of derived classes without knowing it: all the derived classes must honor the contract defined by the base class

Unit Test & Design

- OCP: All dependencies are passed into a parametric constructor or to a method, no singleton no static methods call no new object instantiated inside the class

- DIP: All dependencies implement an interface used type of the constructor/method arguments

Unit Test & Design

- SRP: composition over inheritance promoted by TDD with Mocks prevent abuse of inheritance that is a common case of violation of SRP
- ISP: interfaces extracted from a class that conform to SRP tend to conform also to ISP

Unit Test & Design

- LSP: inheritance in TDD change from a process of invention into a process of discovery: this prevents many violations of the LSP
- LSP: unit test can be run on all the derived class so violations of the contract can be discovered

OCP – DIP

SRP – ISP

LSP

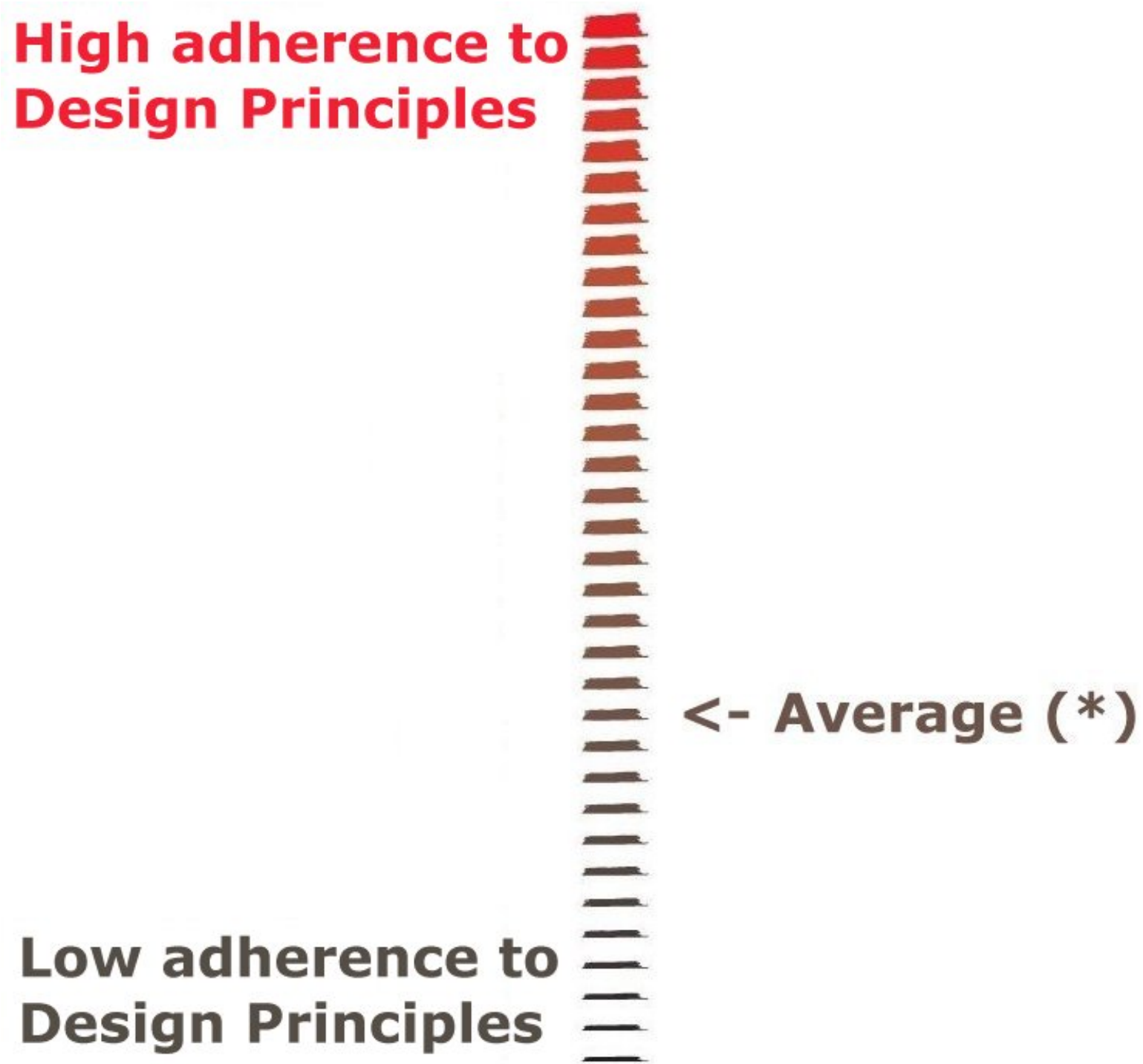
LoD

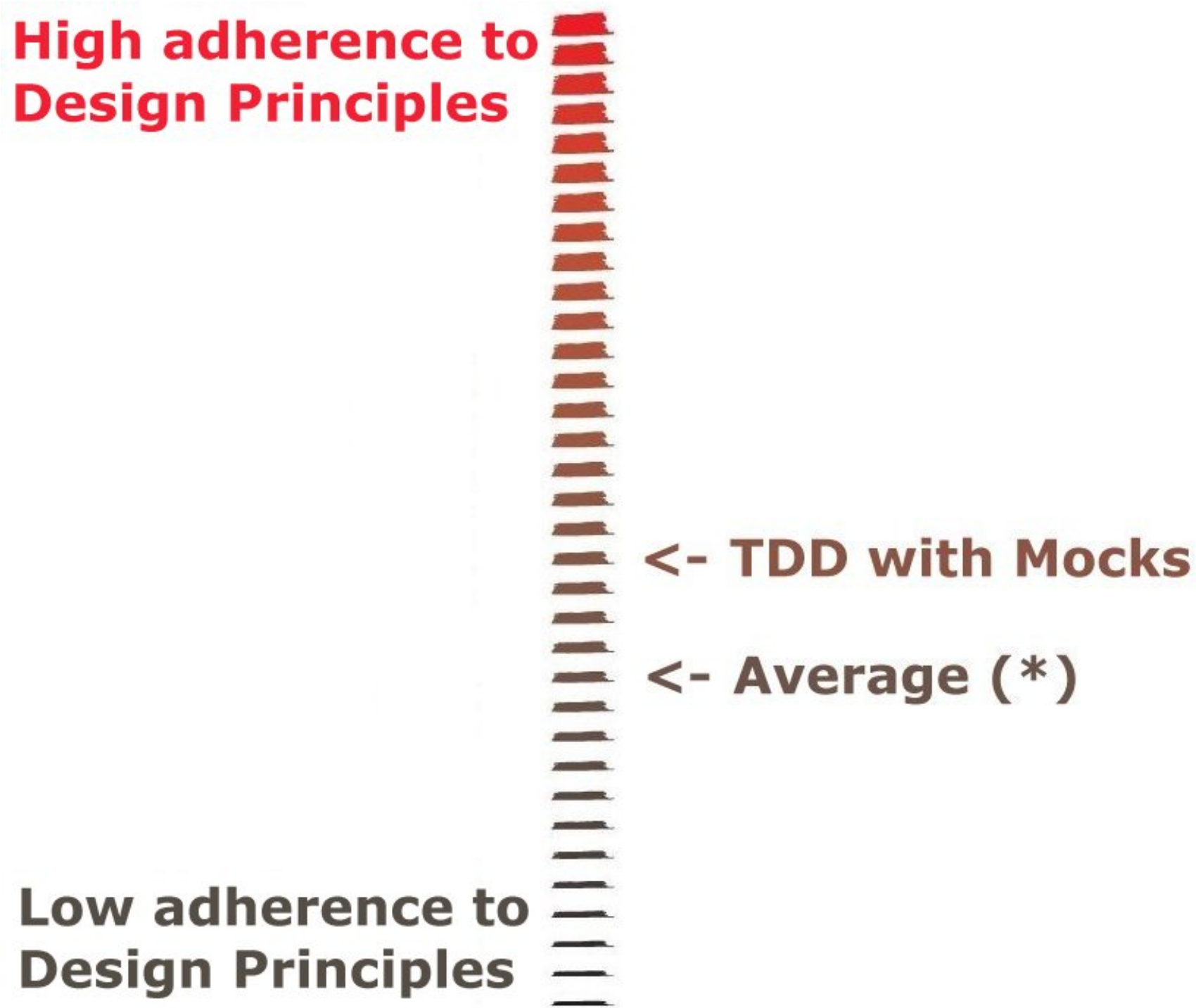
Difficulties experienced with TDD and mocks ?

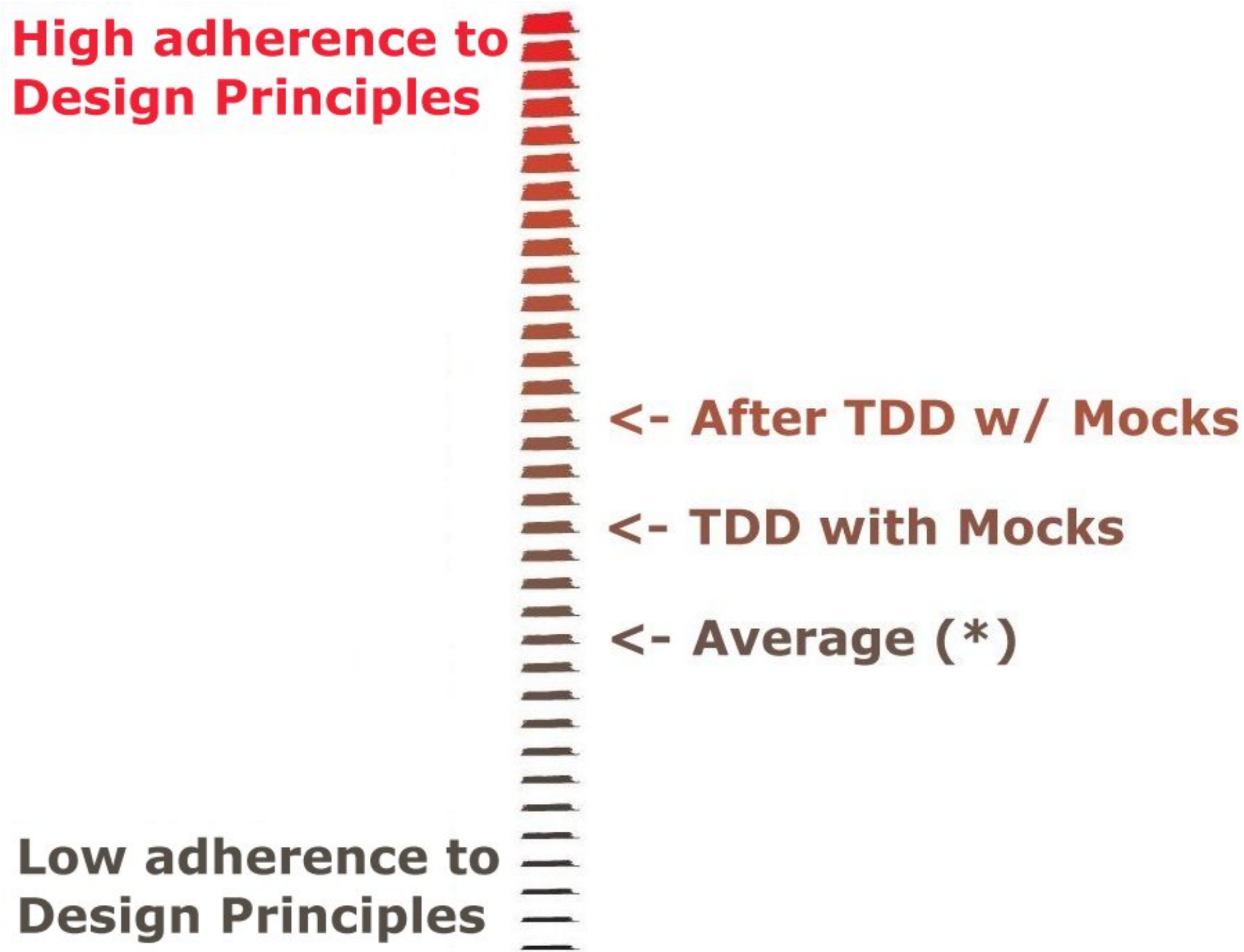
Downsides observed in the use of mocks ?

High adherence to Design Principles

Low adherence to Design Principles


High adherence to
Design Principles
<- Average (*)
Low adherence to
Design Principles


High adherence to
Design Principles
<- TDD with Mocks
<- Average (*)
Low adherence to
Design Principles


High adherence to
Design Principles
<- After TDD w/ Mocks
<- TDD with Mocks
<- Average (*)
Low adherence to
Design Principles

Co-Evolution / Attractors / Barriers

Prof. Sugata Mitra speculation

education is a self organizing system where learning is an emergent phenomenon

Norwegian Developer Conference 2010

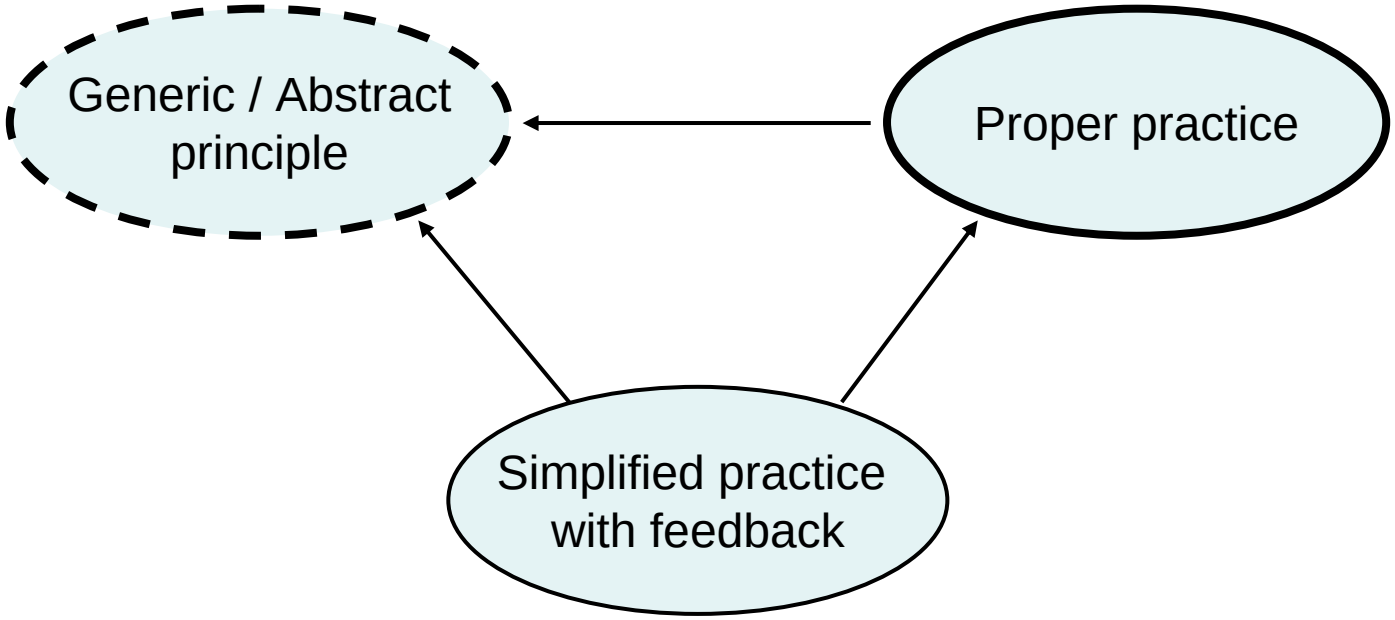
Session ‘The Deep Synergy Between Testability and Good Design’

Michael Feathers:

writing tests is another way to look the code and locally understand it and reuse it, and that is the same goal of good OO design. This is the reason of the deep synergy between testability and good design.

Lessons learned

- Learning SOLID and design principles
- Learning TDD with Mocks



- TDD relation to design
- Language ambiguity

About the conjectures

Observations, analysis and the experiment are compatible with the conjectures that the practice of TDD with Mocks Objects lead the team to :

- write code more conformant to the S.O.L.I.D. design principles and partially to the Law of Demeter.

- learn and develop a deeper understanding of the design principles and their practical applications

And they are compatible with the conjecture that the conformance to the design principle is an emergent property and the learning of the design principle is a process of coevolution.

Relevant variables

Prerequisites

- Motivation, autonomy, proper training
- Early frequent feedback from users and code

Expected outcome

- Number of violations of SOLID and LoD decrease after the training
- Then understanding of SOLID and LoD improve

Code / Paper / Slides :

http://github.com/lucaminudel

Feedback / Comments / Questions :

Luca Minudel - tdd@minudel.it


Formula 1
Formula 1™