

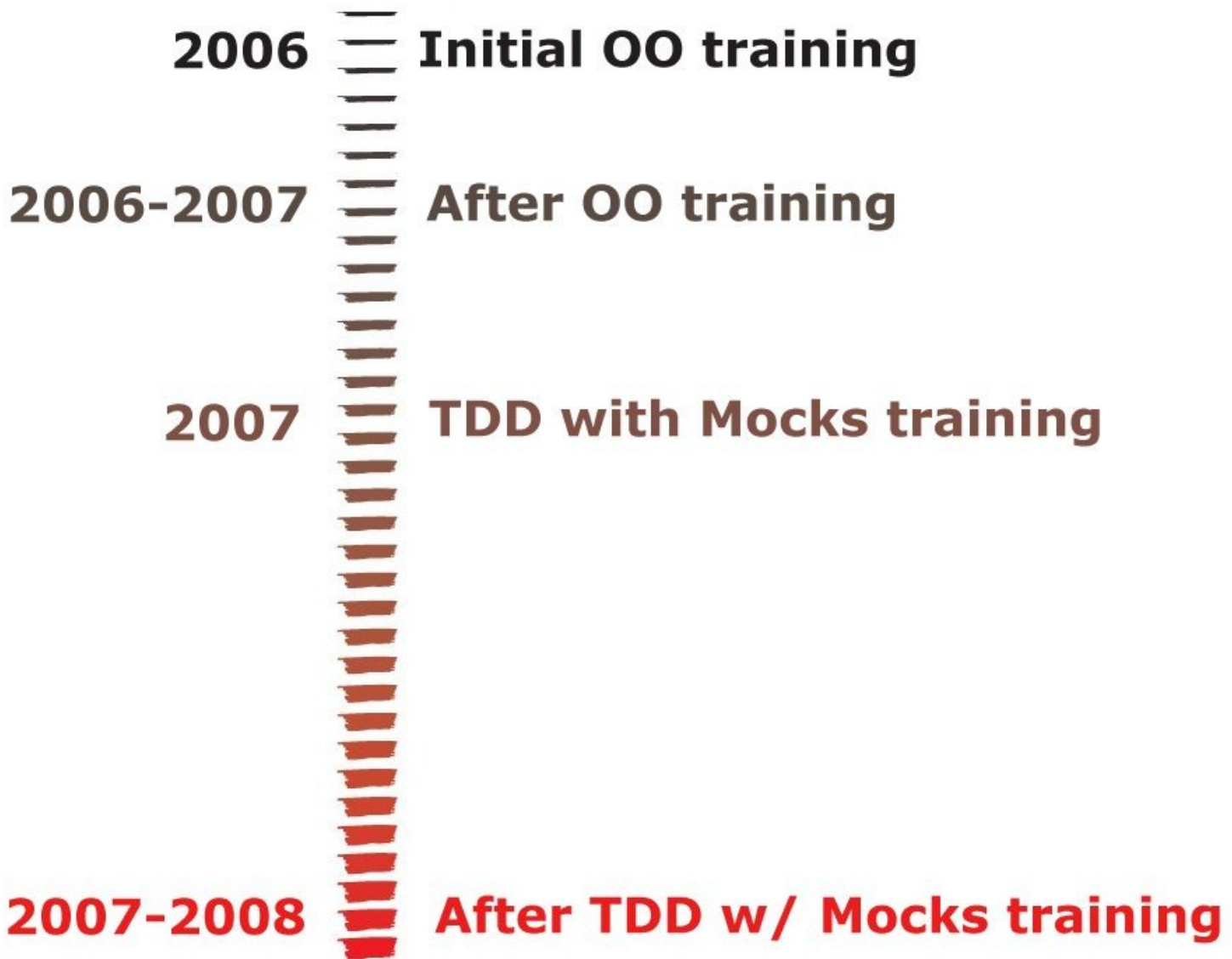


Luca Minudel

# Exploratory study

based on :

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



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 the Software Engineering - XP2004 conference and will be published in *XP eXamined*.

### Abstract

Unit testing is a fundamental part of software development, but it is difficult to test in isolation. It is also difficult to maintain and extend domain code and test suites. This paper describes a technique for test structure, and avoid polluting the test code.

### 1 Introduction

*"Once," said the Mock Object,*

Unit testing is a fundamental part of software development, but it is difficult to test in isolation. It is also difficult to maintain and extend domain code and test suites. This paper describes a technique for test structure, and avoid polluting the test code.

We propose a technique called Mock Objects, which emulates the behavior of the code which they test from inside the test. We write stubs with two interfaces, one as usual, and we use our tests to verify the behavior.

Our experience is that developing a better structure of both domain code and test code in a regular format that gives the developer a clear path should be written to make it easier to achieve this. We describe a technique to achieve this. We describe the cost of writing stub code.

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

### 2 Unit testing with Mock Objects

An essential aspect of unit testing is to know you are testing and where any dependencies are, 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

{sfreean, npryce, tmackinnon, jwalnes} @thoughtworks.com

### ABSTRACT

Mock Objects is an extension to Test-Driven Development that supports good Object-Oriented design. It is a coherent system of types within a class library that is less interesting as a technique for isolating code than is widely thought. This paper describes the use of using Mock Objects with an extended Test-Driven Development process. It also introduces jMock, a Java framework for our collective experience.

**Categories and Subject Descriptors:** D.2.2 [Software Engineering]: Design and Analysis—Object-Oriented design methods

### General Terms

Design, Verification

### Keywords

Test-Driven Development, Mock Objects

### 1. INTRODUCTION

Mock Objects is misnamed. It is really a system based on the roles that

In [10] we introduced the concept of Mock Objects to support Test-Driven Development. We describe better structured tests and, more importantly, how to code by preserving encapsulation, reducing dependencies, and clarifying the interactions between classes. We describe how we have refined and adjusted the Mock Objects experience since then. In particular, we describe the most important benefit of Mock Objects, which we call "interface discovery". We have developed a framework to support dynamic generation of Mock Objects on this experience.

The rest of this section establishes the context of Test-Driven Development and good programming, and then introduces the rest of the paper introduces the Mock Objects experience.

Permission to make digital or hard copies of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use in any other manner requires prior specific permission and/or a fee. *Conference '04*, Month 1–2, 2004, City, State, Country. Copyright 2004 ACM 1-58113-000-0/000000.

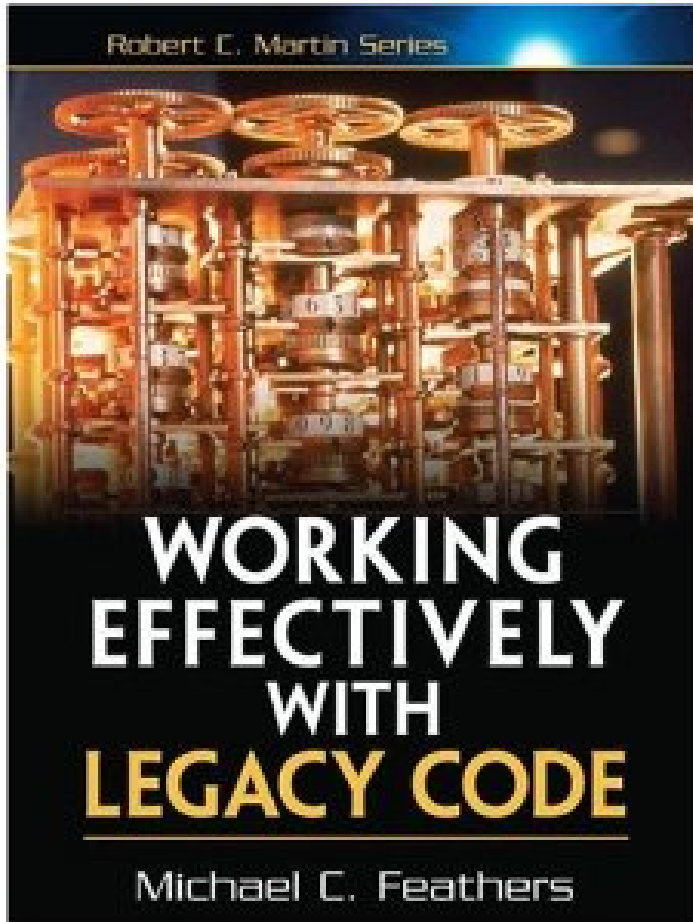
*The Addison-Wesley Signature Series*

A KENT BECK SIGNATURE BOOK

GROWING  
OBJECT-ORIENTED  
SOFTWARE,  
GUIDED BY TESTS

STEVE FREEMAN  
NAT PRYCE





- 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

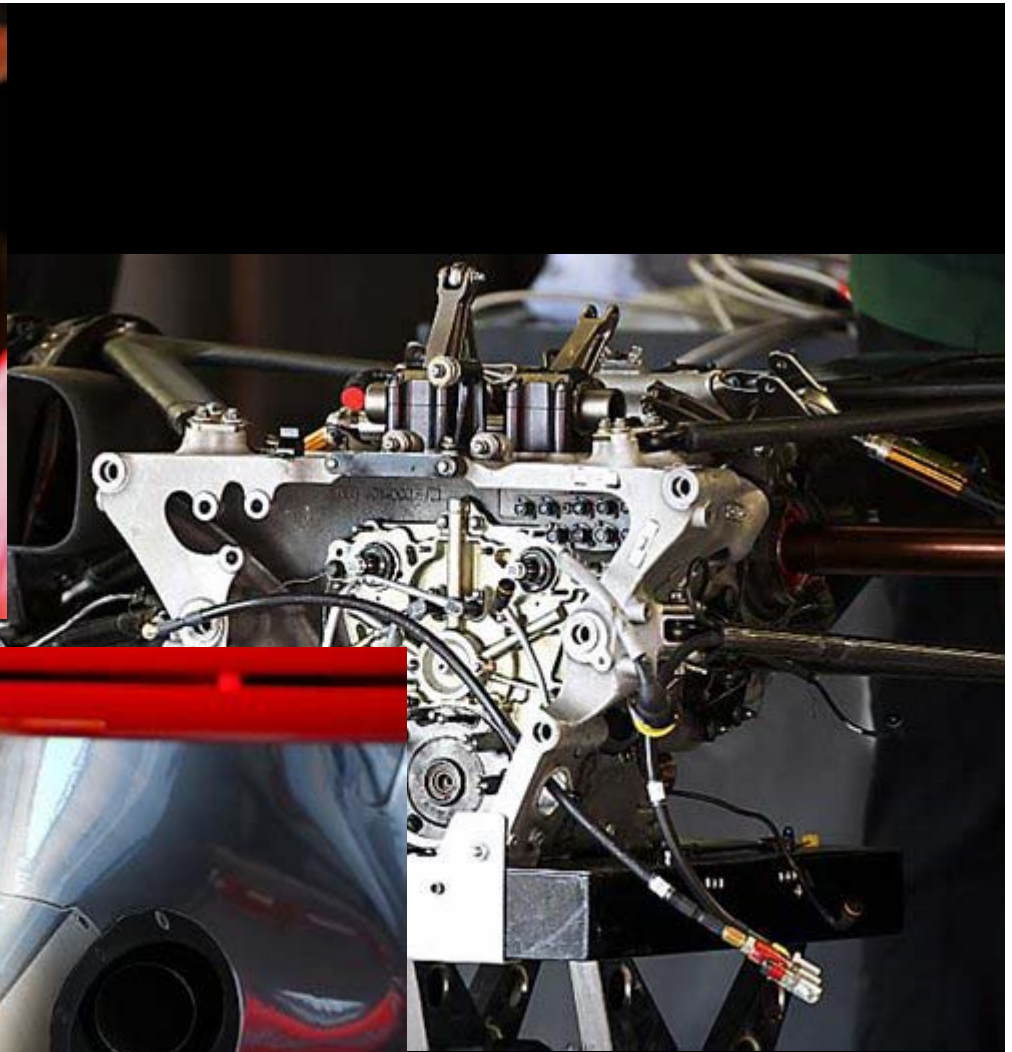
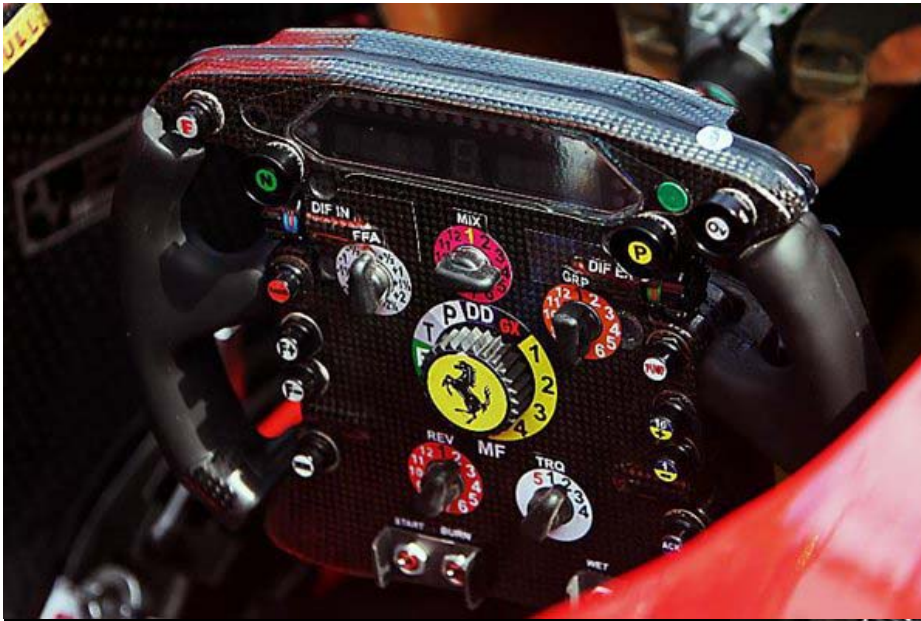
*1st Law of Software Process:*

*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*



# 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***

# 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

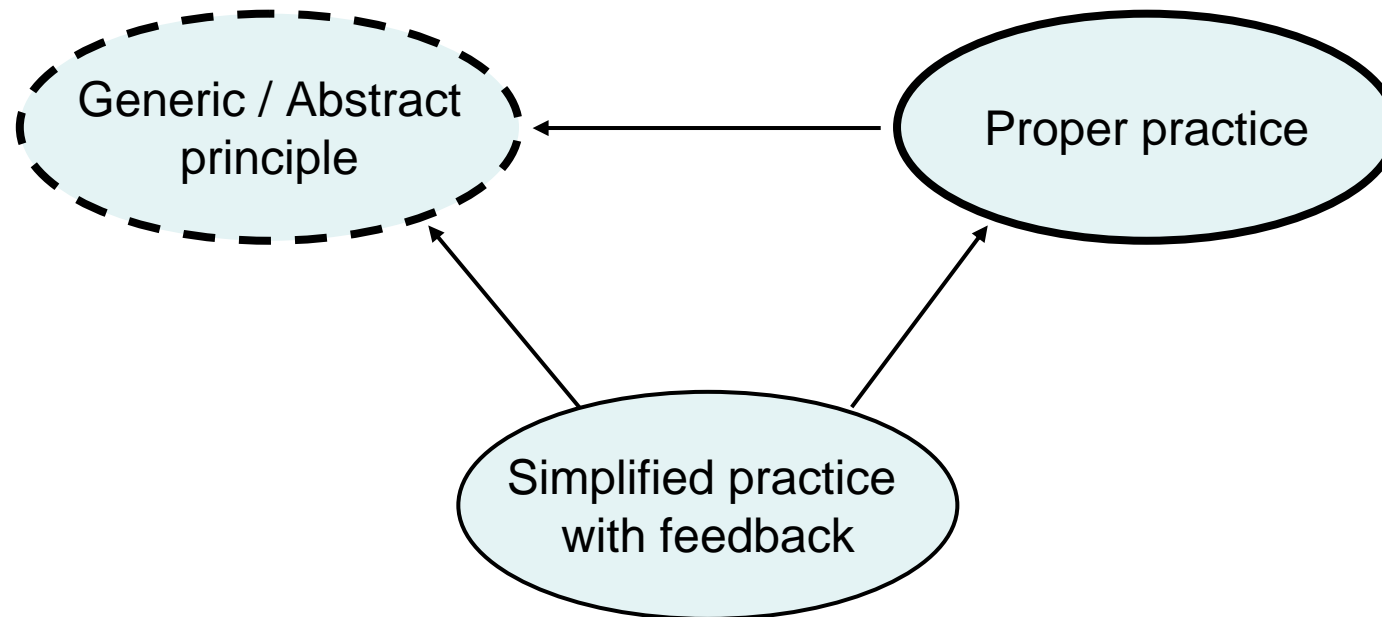
Session 'The Deep Synergy Between  
Testability and Good Design'

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

# What is Cint?

Global technology company

Developing software to help market researchers

Vision: Bring market insight to everyone (through technology)

Industry firsts:

- Marketplace for accessing survey responders globally (Cint Panel Exchange)
  - Tracker technology for market research trackers (repetitive studies)
    - Widget to offer access to respondents within 3rd party software
  - Widget for publishers (blogs, websites, communities) to make money by building respondent panels

Agile teams in Stockholm, Sweden and Atlanta, USA



[WWW.CINT.COM](http://WWW.CINT.COM)

**Code / Paper / Slides :**

**<http://github.com/lucaminudel>**

**Feedback / Comments / Questions :**

**Luca Minudel - [tdd@minudel.it](mailto:tdd@minudel.it)**

