

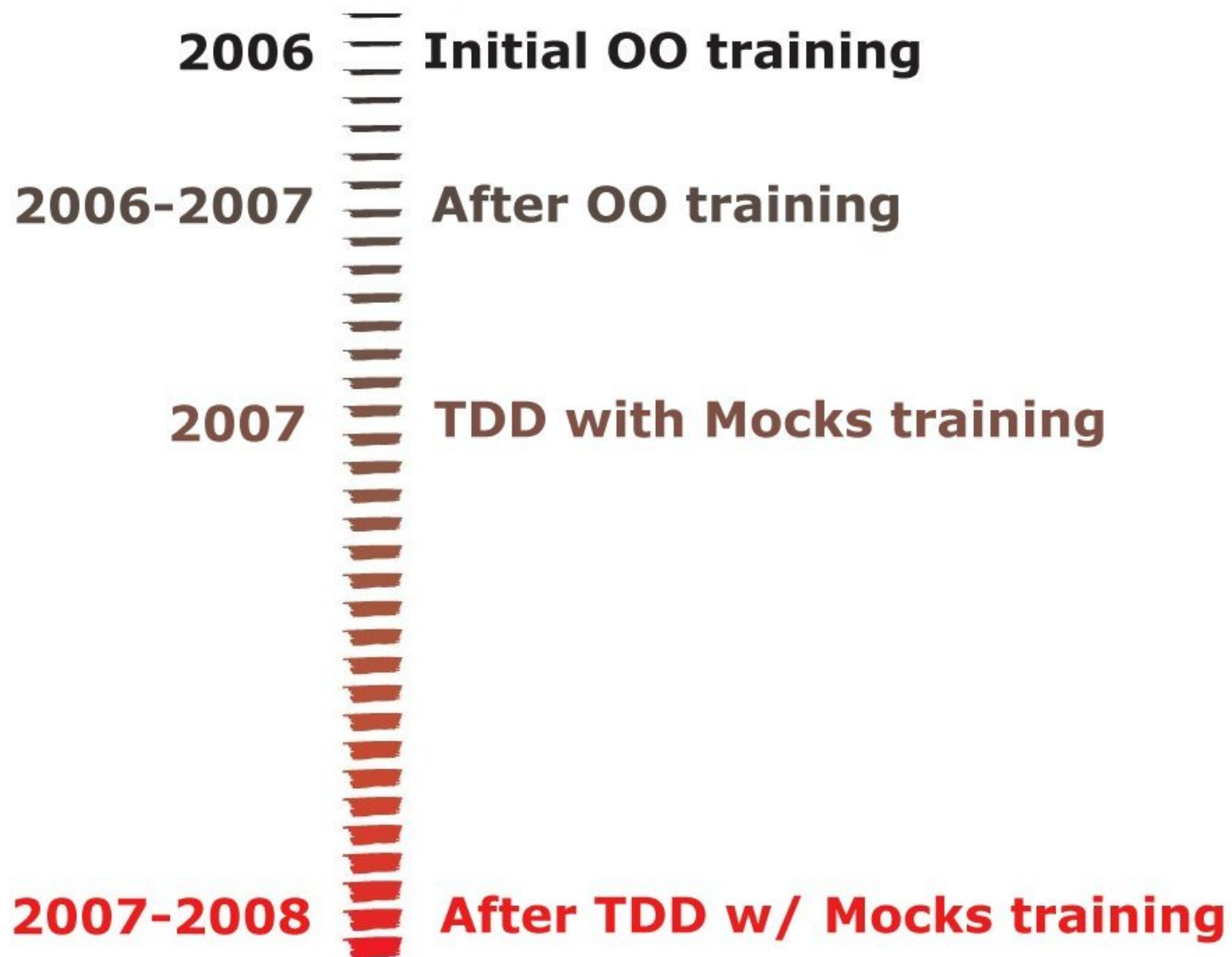


Luca Minudel

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
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 testing structure, and avoid polluting the code under test.

Keywords: Extreme Programming, Unit Testing, Mock Objects

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 testing structure, and avoid polluting the code under test.

We propose a technique called Endo-Testing, which allows us to write code which they test from inside the code they are testing. We write code stubs with two interfaces, one for the code under test, and one for the code we use in our tests.

Our experience is that developing a better structure of both domain code and test code gives the developer a regular format that gives the code a better structure. We should be written to make it easier to achieve this. We have found a technique to achieve this. We have found a technique to achieve this. We have found a technique to achieve this.

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

2 Unit testing with Mock Objects

An essential aspect of unit testing is knowing where you are testing and where any dependencies are. We want to 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

{sfreeeman, 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 code library that is less interesting as a technique for isolating code than is widely thought. This paper describes the use of Mock Objects with an extended version of Test-Driven Development, and worst practices gained from experience. 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 objects play in a system. In [10] we introduced the concept of Mock Objects to support Test-Driven Development. We have better structured tests and, more importantly, we have preserved encapsulation, reduced coupling, and clarified the interactions between classes. We have refined and adjusted the concept since then. In particular, we have found the most important benefit of Mock Objects is called "interface discovery". We have found a framework to support dynamic generation of tests 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 concept of Mock Objects.

Permission to make digital or hard copies of this work for personal or classroom use is granted without fee, provided that the copies are not made or distributed for profit or commercial advantage, and that the full copyright notice and title are included in the copy. For all other uses, permission should be sought from the ACM Press. Copyright 2004 ACM 1-58113-000-0/000000

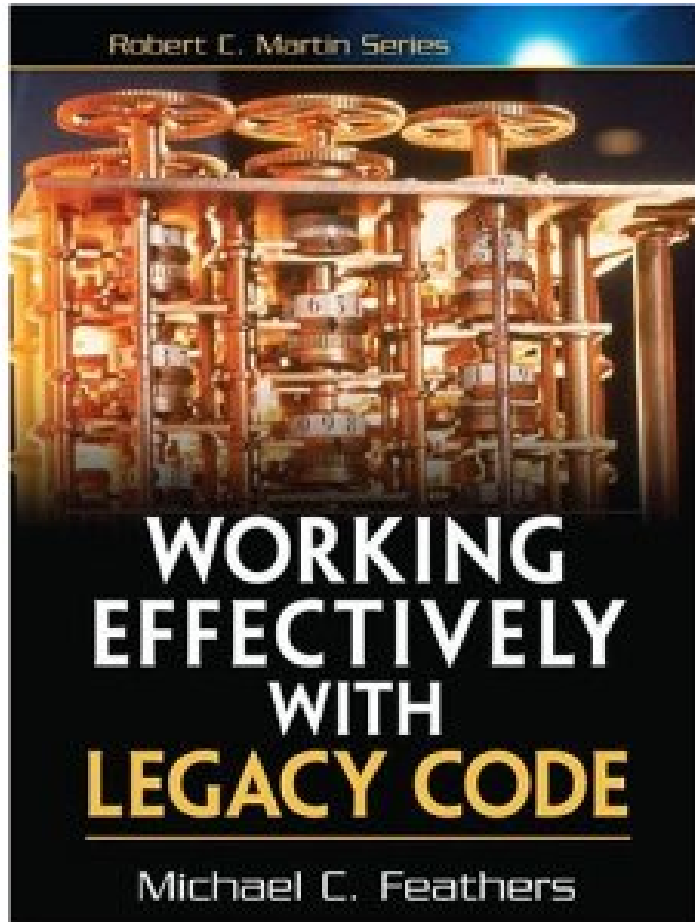
The Addison-Wesley Signature Series

A KENT BECK SIGNATURE BOOK

GROWING
OBJECT-ORIENTED
SOFTWARE,
GUIDED BY TESTS

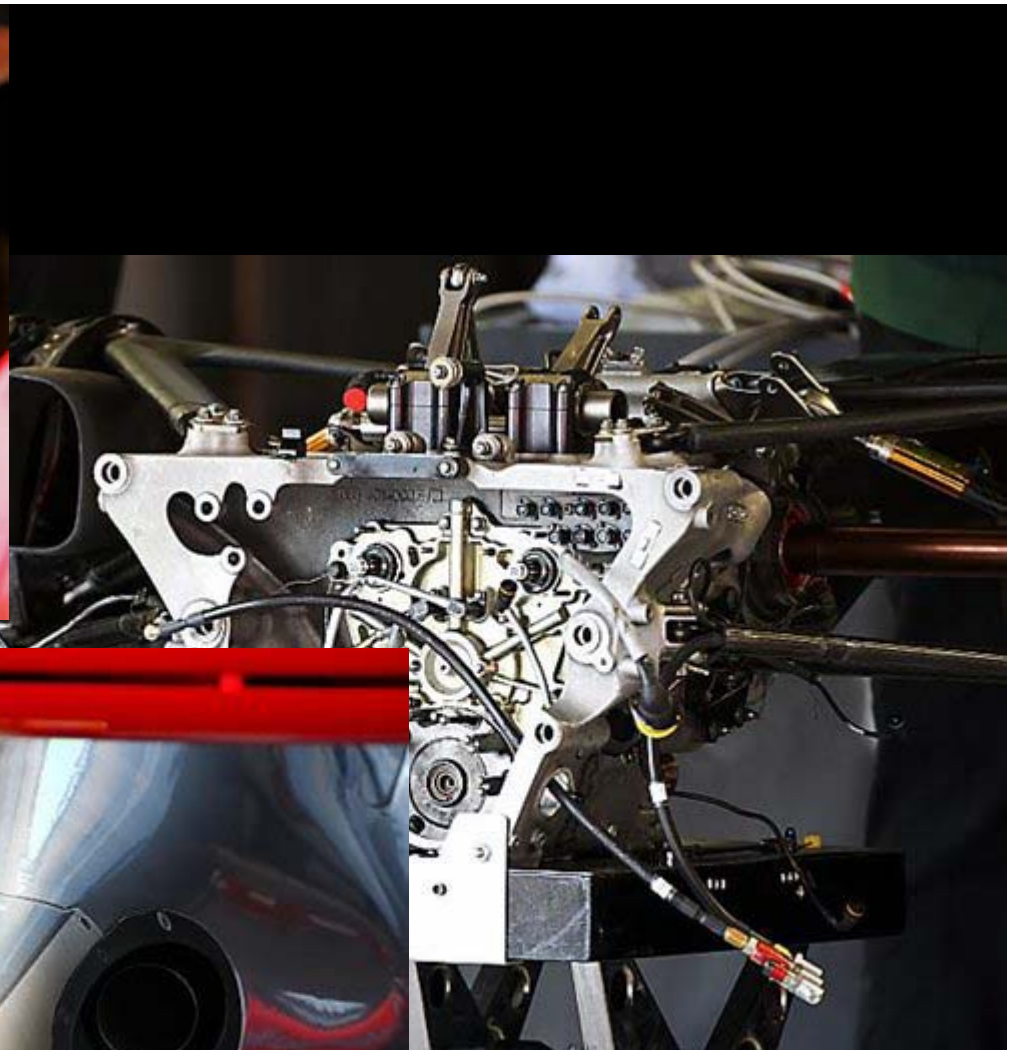
STEVE FREEMAN
NAT PRYCE





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

*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***

D. H. Ackley et al. 2002

*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

OCP – DIP

SRP – ISP

LSP

LoD

Downsides observed in the use of mocks ?

Difficulties experienced with TDD and 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**

Co-Evolution / Attractors / Barriers

Prof. Sugata Mitra speculation

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

**High adherence to
Design Principles**



<- After TDD w/ Mocks

<- TDD with Mocks

<- Average (*)

**Low adherence to
Design Principles**

Norwegian Developer Conference 2010

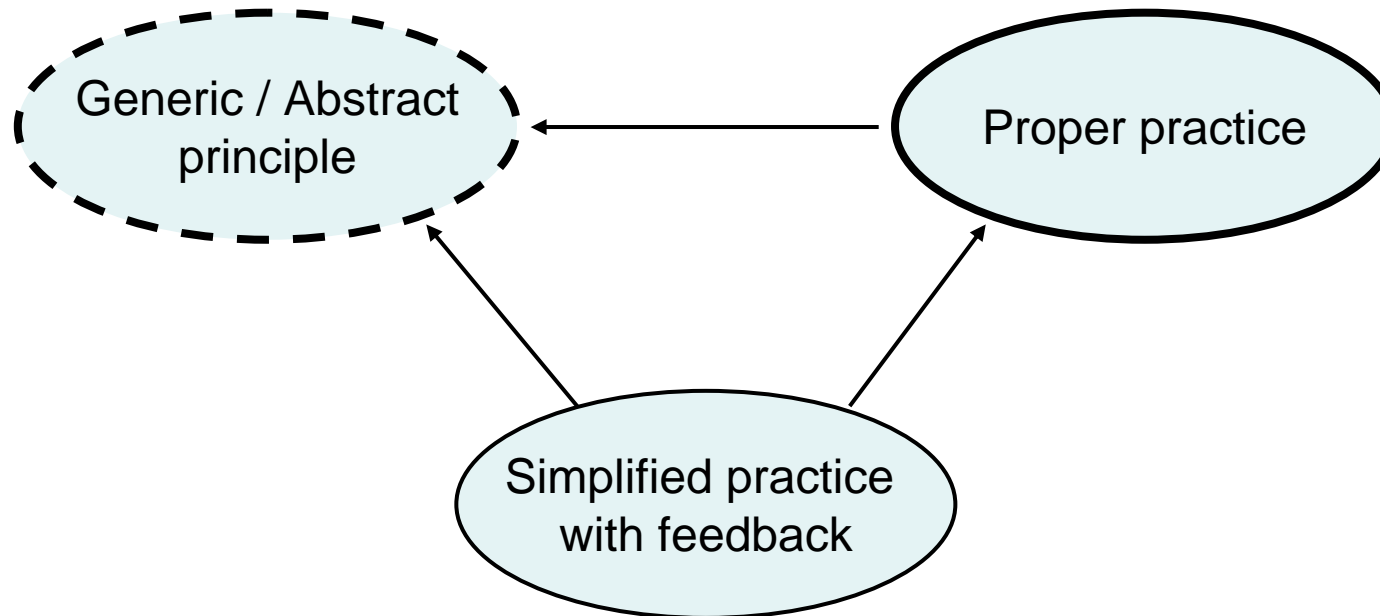
Session 'The Deep Synergy Between Testability and Good Design'

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

