Object Oriented Programming

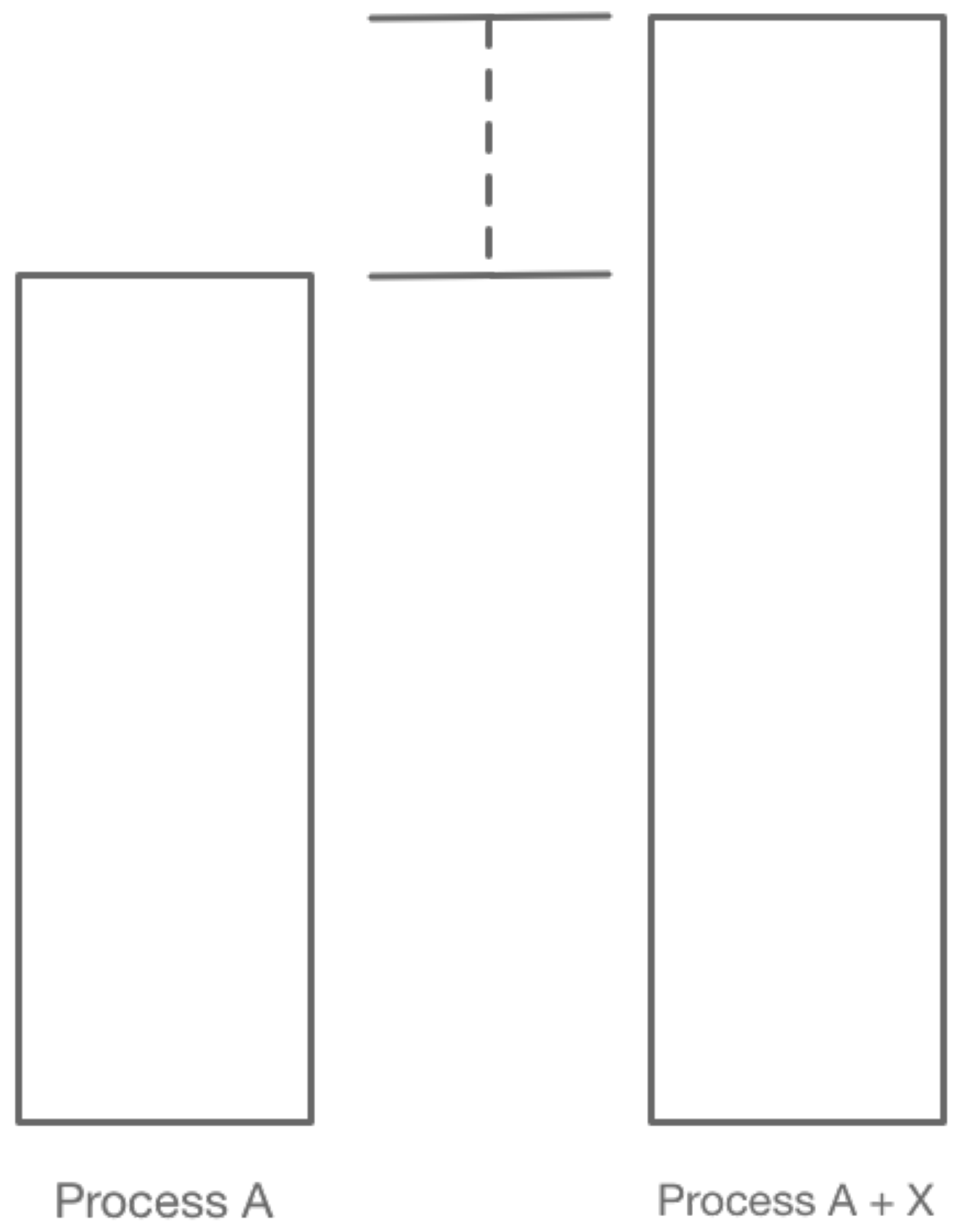
Object Oriented Techniques and their Role in the Iterative Software Development Process

Object Oriented Programming, Why *YOU* suck at it and why *I* rule.

(Official title)

Disclaimer

- No silver bullets
- In search of a better software process



What is OOP?

over-asked uninteresting question

What is OOP?

over-asked uninteresting question

with an interesting answer

Common Lisp

```lisp
(defclass srt-time ()
  ((hr :initarg :hr :initform 0 :accessor hr)
   (mi :initarg :mi :initform 0 :accessor mi)
   (se :initarg :se :initform 0 :accessor se)
   (ms :initarg :ms :initform 0 :accessor ms))
  (:documentation "Time format for srt"))

(defgeneric display (what)
  (:documentation "Returns string that represents the object"))

(defgeneric normalise (time)
  (:documentation "Fix overflow of fields"))

(defmethod normalise ((time srt-time))
  (with-slots (hr mi se ms) time
    (loop until (< ms 1000) do (decf ms 1000) (incf se))
    (loop until (< se 60) do (decf se 60) (incf mi))
    (loop until (< mi 60) do (decf mi 60) (incf hr)))
  time)

(defmethod display ((time srt-time))
  (normalise time)
  (with-slots (hr mi se ms) time
    (format nil "~2,'0d:~2,'0d:~2,'0d,~3,'0d" hr mi se ms)))

(defun make-srt-time (arglist)
  (destructuring-bind (hr mi se ms) arglist
  (make-instance 'srt-time :hr hr :mi mi :se se :ms ms)))
```

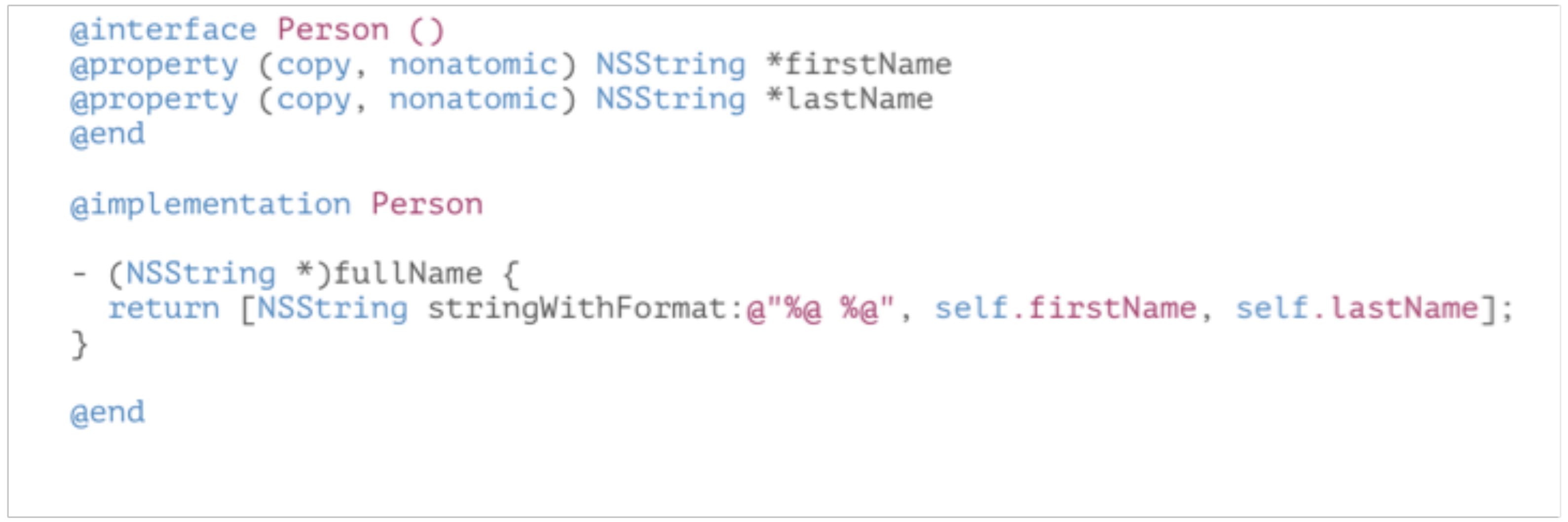
Lua

```lua
Account = {balance = 0}

function Account:new (o)
    o = o or {}
    setmetatable(o, self)
    self.__index = self
    return o
end

function Account:deposit (v)
    self.balance = self.balance + v
end

function Account:withdraw (v)
    if v > self.balance then error"insufficient funds" end
    self.balance = self.balance - v
end

SpecialAccount = Account:new()

function SpecialAccount:withdraw (v)
    if v - self.balance >= self:getLimit() then
        error"insufficient funds"
    end

    self.balance = self.balance - v
end

function SpecialAccount:getLimit ()
    return self.limit or 0
end
```

Java

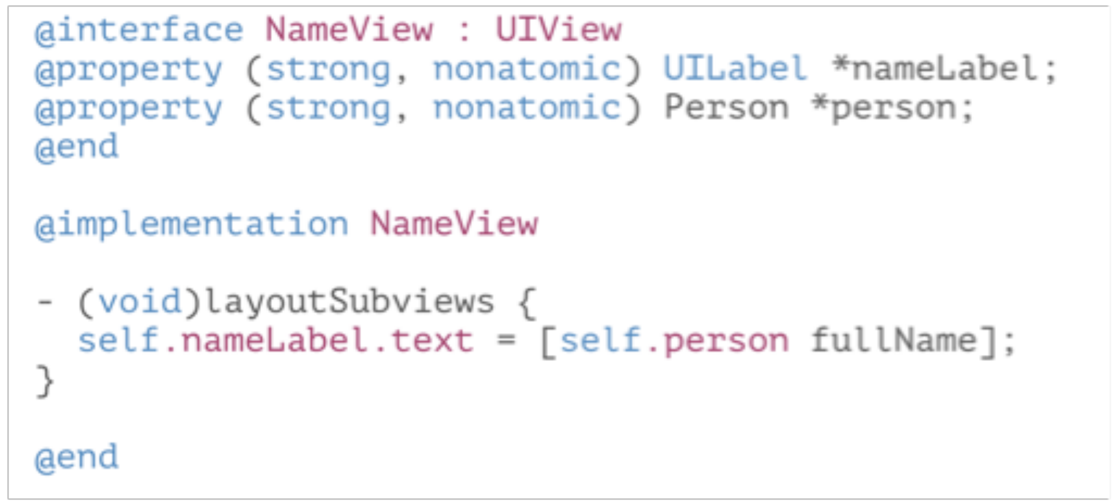
```java
public interface MessageStrategy {
    public void sendMessage();
}

public abstract class AbstractStrategyFactory {
    public abstract MessageStrategy createStrategy(MessageBody mb);
}

public class MessageBody {
    Object payload;

    public Object getPayload() {
        return payload;
    }

    public void configure(Object obj) {
        payload = obj;
    }

    public void send(MessageStrategy ms) {
        ms.sendMessage();
    }
}

public class DefaultFactory extends AbstractStrategyFactory {
    private DefaultFactory() {;}
    static DefaultFactory instance;

    public static AbstractStrategyFactory getInstance() {
        if (instance==null) instance = new DefaultFactory();
        return instance;
    }

    public MessageStrategy createStrategy(final MessageBody mb) {
        return new MessageStrategy() {
            MessageBody body = mb;
            public void sendMessage() {
                Object obj = body.getPayload();
                System.out.println((String)obj);
            }
        };
    }
}

public class HelloWorld {
    public static void main(String[] args) {
        MessageBody mb = new MessageBody();
        mb.configure("Hello World!");
        AbstractStrategyFactory asf = DefaultFactory.getInstance();
        MessageStrategy strategy = asf.createStrategy(mb);
        mb.send(strategy);
    }
}
```

Objective-c



```
@interface Person ()
@property (copy, nonatomic) NSString *firstName
@property (copy, nonatomic) NSString *lastName
@end

@implementation Person

- (NSString *)fullName {
  return [NSString stringWithFormat:@"%@ %@", self.firstName, self.lastName];
}

@end
```

Model

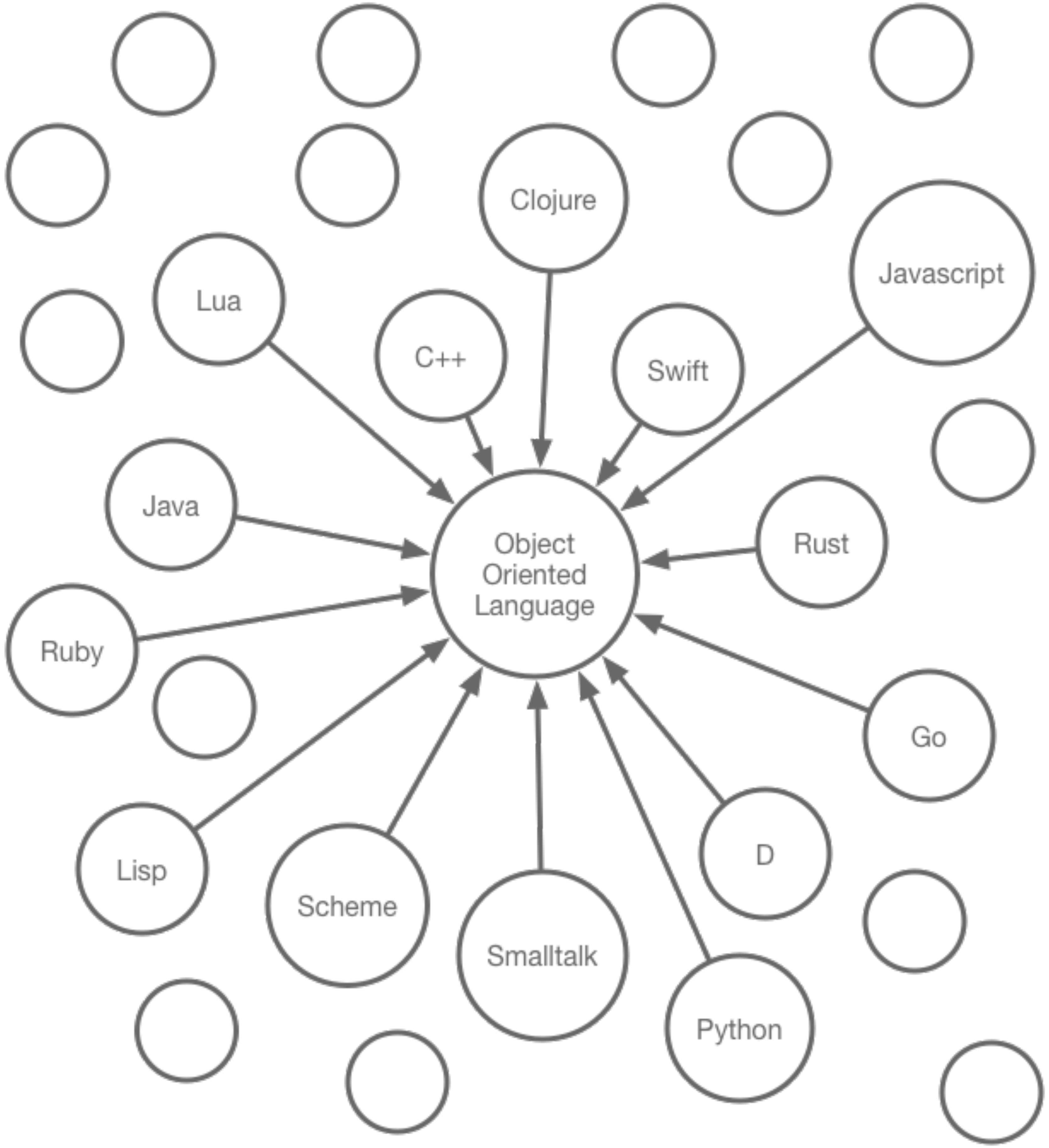


```
@interface NameView : UIView
@property (strong, nonatomic) UILabel *nameLabel;
@property (strong, nonatomic) Person *person;
@end

@implementation NameView

- (void)layoutSubviews {
  self.nameLabel.text = [self.person fullName];
}

@end
```

View

Controller

Object Oriented Langauges



All very different.

What do they all have in common?

The Interface

The Interface
a mechanism for enabling dynamic behavior with static code

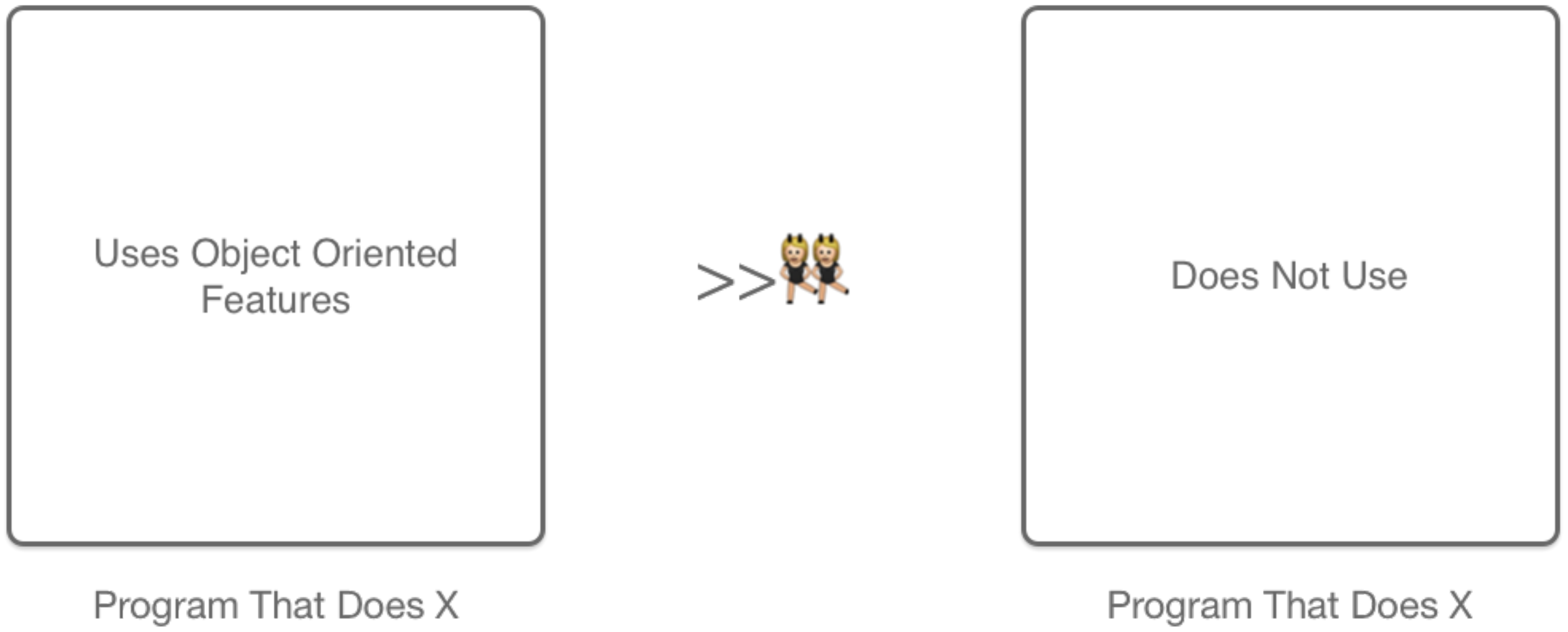
Interface Example

```
public class Person {
    public String name;

    public void printName() {
        System.out.println(name);
    }
}

public class VipPerson extends Person {
    public void printName() {
        System.out.println("Mr. " + name);
    }
}

public static void displayPerson(Person p) {
    p.printName();
}
```

Why Object Oriented Programming?

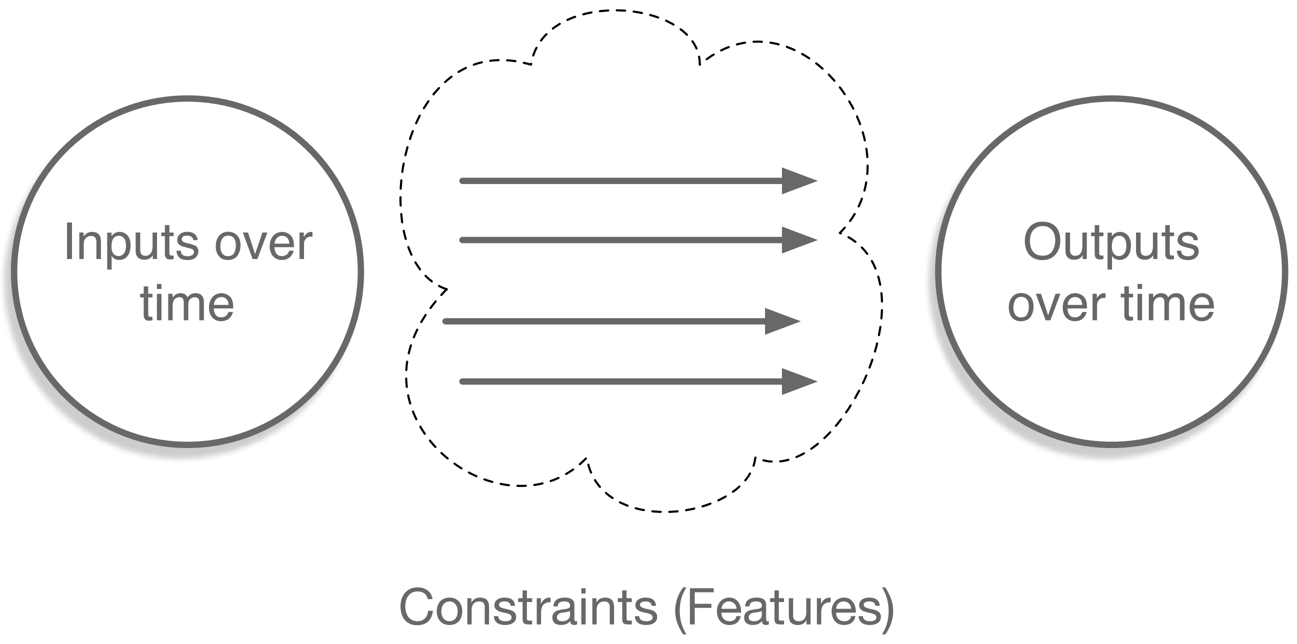
more interesting question

If OOP is useful at all...



What is a program?

A program is a machine executable definition of
a process that maps a set of inputs over time to a set of outputs over time meeting a set of given constraints.



Program A
>>👯?
Program B

What is More Awesome?

— two programs

— meet same constraints

— is there an A more awesome than B?

What is More Awesome?

What if we did one of these to B?

- Rot13 all variable and function names
- Minify

Why is A more awesome than B?

Change

Uncle Bob

Robert C. Martin

The secondary value of software is to meet the user's needs.
The primary value is to change.

New Code

```
public static void main(String[] args) {

}
```

Changing Old Code

```java
public static void main(String[] args) {
    System.out.println("Nice to meet you, Mr. Bowie");
}
```

Changing Old Code

```java
public static void main(String[] args) {
    if (args.length > 0) {
        System.out.println("Nice to meet you, " + args[0]);
    } else {
        System.out.println("Nice to meet you, Mr. Bowie");
    }
}
```

Changing Old Code

```java
public static void main(String[] args) {
    String greeting = "Nice to meet you, ";
    if (args.length > 0) {
        if (args[1].equals("Iman")) {
            System.out.println(greeting + "Mrs. Bowie");
        } else {
            System.out.println(greeting + args[0]);
        }
    } else {
        System.out.println(greeting + "Mr. Bowie");
    }
}
```

Get it?

What's wrong with old code?
Why is change a problem?

Answer: Internal Constraints

Internal Constraints are a burden on Change

Duality of Syntax

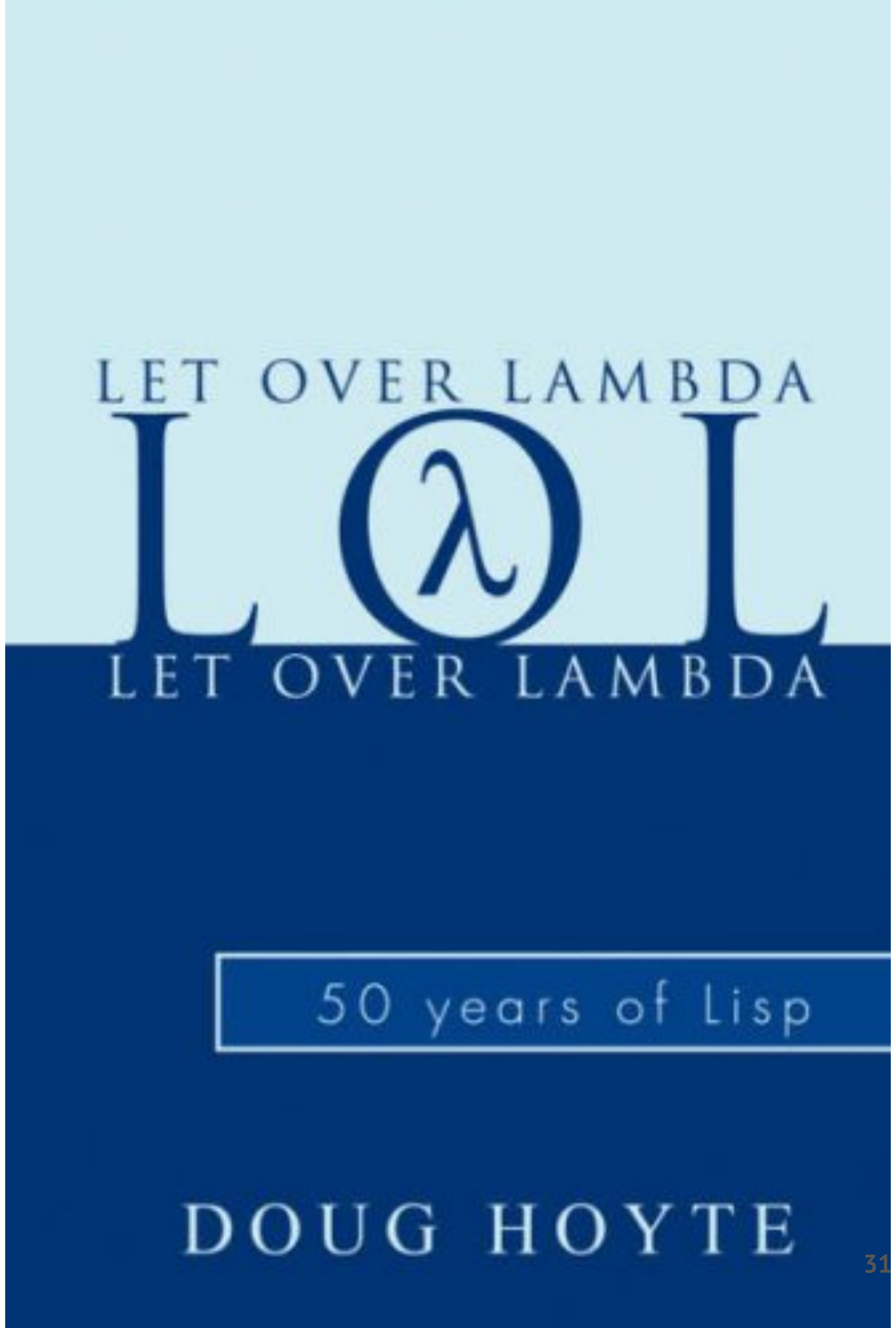
Let Over Lambda

Doug Hoyte

Duality of Syntax

Same syntax multiple behaviors

Richard Gabriel: "Compression"



The Interface

The OOP mechanism for duality of syntax

```
public class Person {
    public String name;

    public void printName() {
        System.out.println(name);
    }
}

public class VipPerson extends Person {
    public void printName() {
        System.out.println("Mr. " + name);
    }
}

public static void displayPerson(Person p) {
    p.printName();
}
```

The Interface

```
public class Person {
    private String name;

    public String getName() {
        return name;
    }

    public setName(String newName) {
        name = newName;
    }
}
```

The Interface

- Enables the same functionality with *fewer internal constraints*
- Fewer internal constraints, means *easier to change*
- Easier to change means *more awesome!* 👯

"I'm already using an object oriented language so I'm already doing this."

Maybe…

Building with Abstractions vs Abstracting

- UIViewController
- UIView
- UITableViewDelegate
- Hollywood Pattern and UIApplicationDelegate

Origin of the 5000 line class

Step 1: Take an existing abstraction

Step 2: Give it a name that matches our problem domain

Step 3: Add code to do stuff

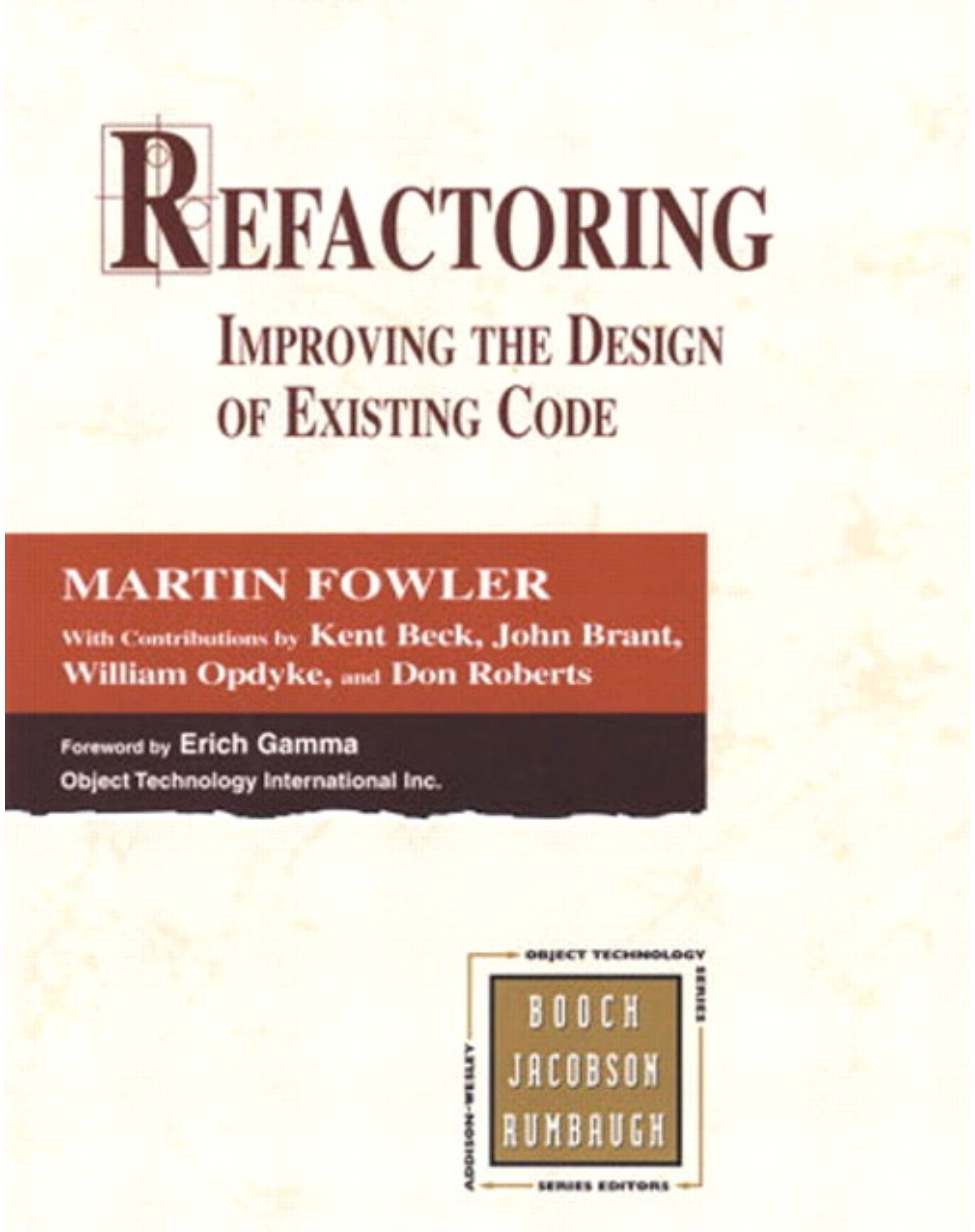
Step 4: Refactor

Writing is Rewriting

“By the time I am nearing the end of a story, the first part will have been reread and altered and corrected at least one hundred and fifty times. I am suspicious of both facility and speed. Good writing is essentially rewriting.” *-Roald Dahl*

“I have rewritten — often several times — every word I have ever published. My pencils outlast their erasers.” -*Vladimir Nabokov*

It's too hard to get code right the first time.


REFACTORING
IMPROVING THE DESIGN OF EXISTING CODE
MARTIN FOWLER
With Contributions by Kent Beck, John Brant, William Opdyke, and Don Roberts
Foreword by Erich Gamma
Object Technology International Inc.
OBJECT TECHNOLOGY SERIES
BOOCH JACOBSON RUMBAUGH
SERIES EDITORS
ADDISON-WESLEY

Refactoring

Martin Fowler

"Elements of Style"

Code Smells

— Duplicated Code

— Shotgun Surgery

— Long Method

— Speculative Generality

— *etc.*

Continuous Refactoring

- A perspective change: messes become problems with solutions
- Object oriented programming becomes a tool for taking things apart as well as putting things together
- Abstractions are fun!

Rich Hickey

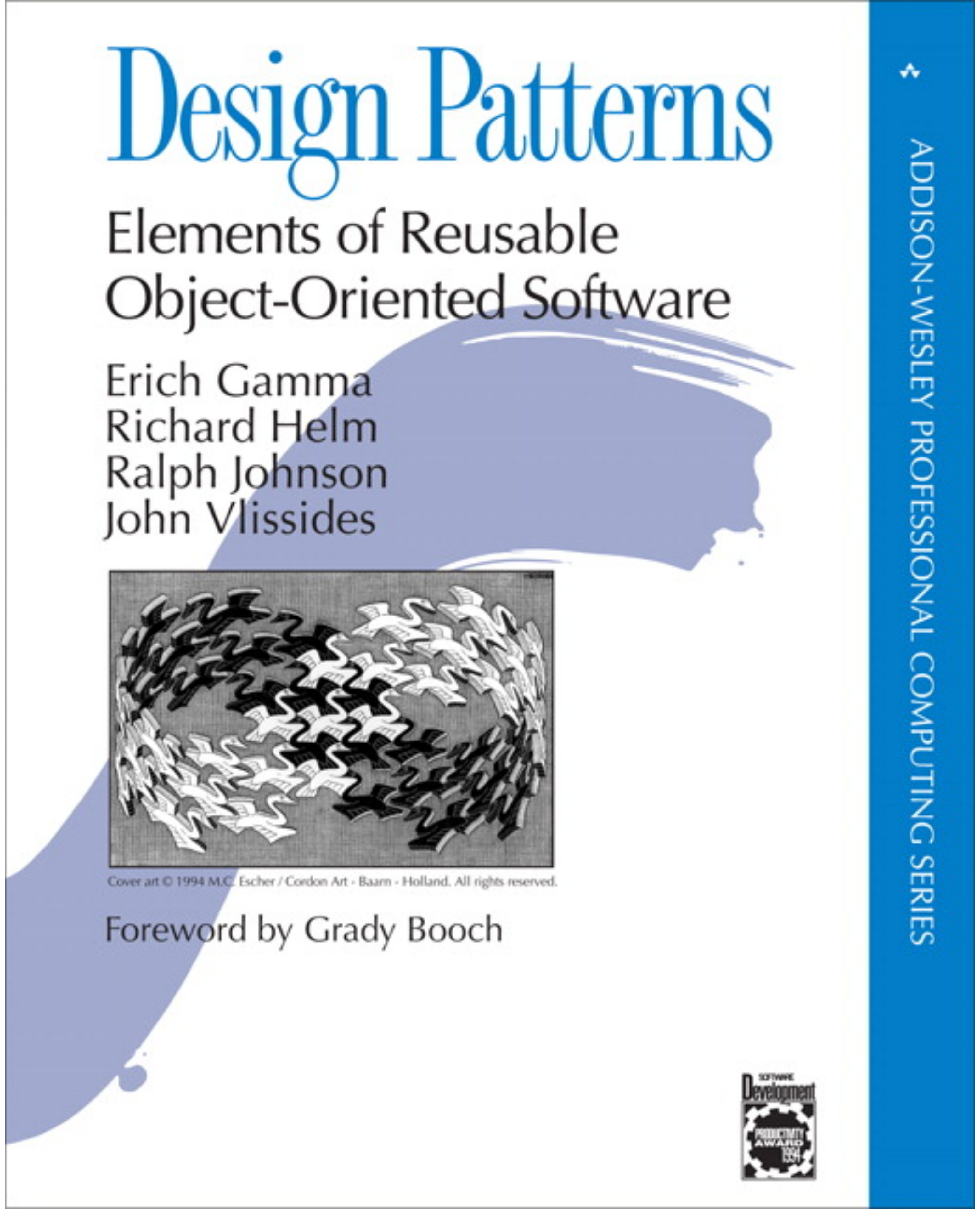
Creator of Cojure

Simple Made Easy

Easy: nearby, subjective

Simple: not intertwined, objective (not easy)

Simplicity comes after complexity, not before.


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Cover art © 1994 M.C. Escher / Cordon Art - Baarn - Holland. All rights reserved.
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Design Patterns

Abstractions for managing dependencies in ways not directly supported by the language.

Build with: *bad*

Refactor to: *super sweet*

Indirecton without Abstraction

"Where is anything getting done?!"

Abstractions mean more smaller pieces

Trading in physical locality for conceptual locality

Bad abstractions and bad names are still bad code

Abstractions aren't the problem

Average adult has a 15,000 word vocabulary

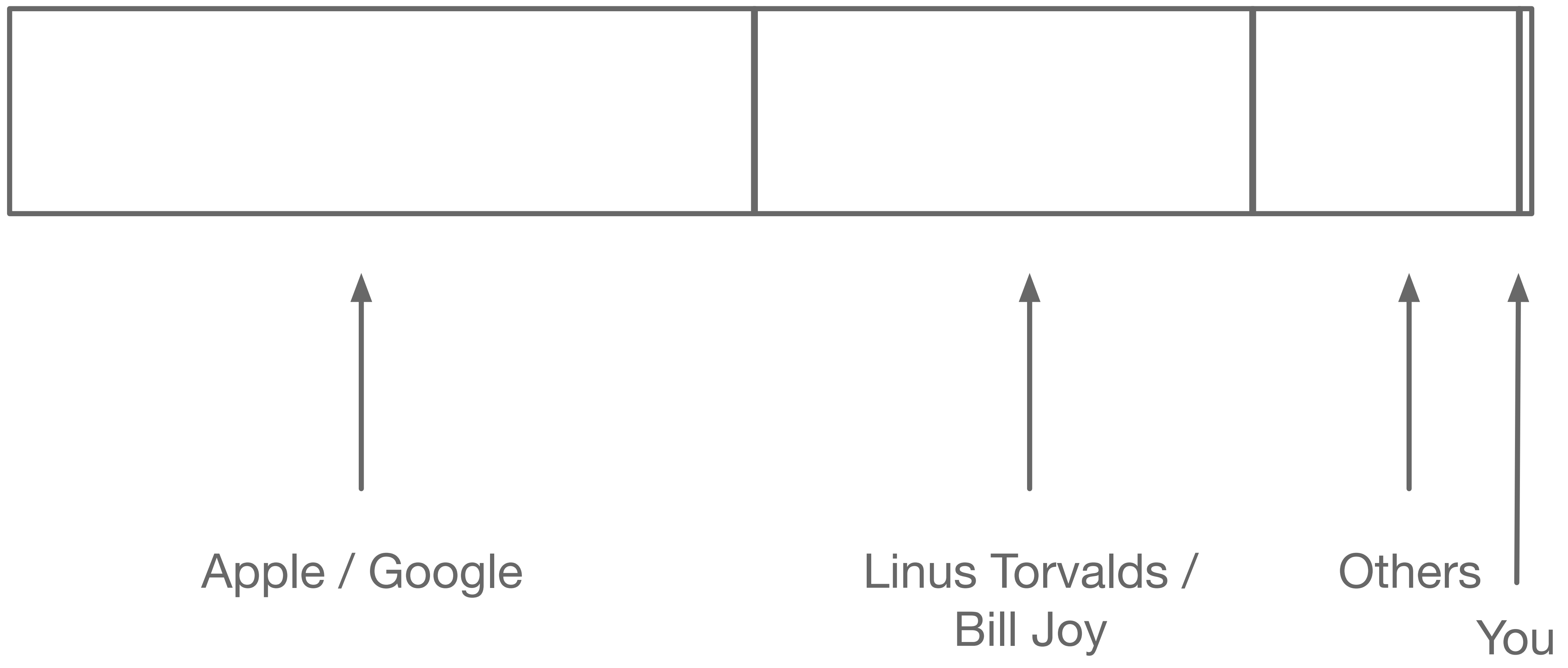
We don't understand every line of external libraries

We certainly don't know what's behind Apple's abstractions

Even worse...

We don't even write our own programs

Contribution to your app breakdown:



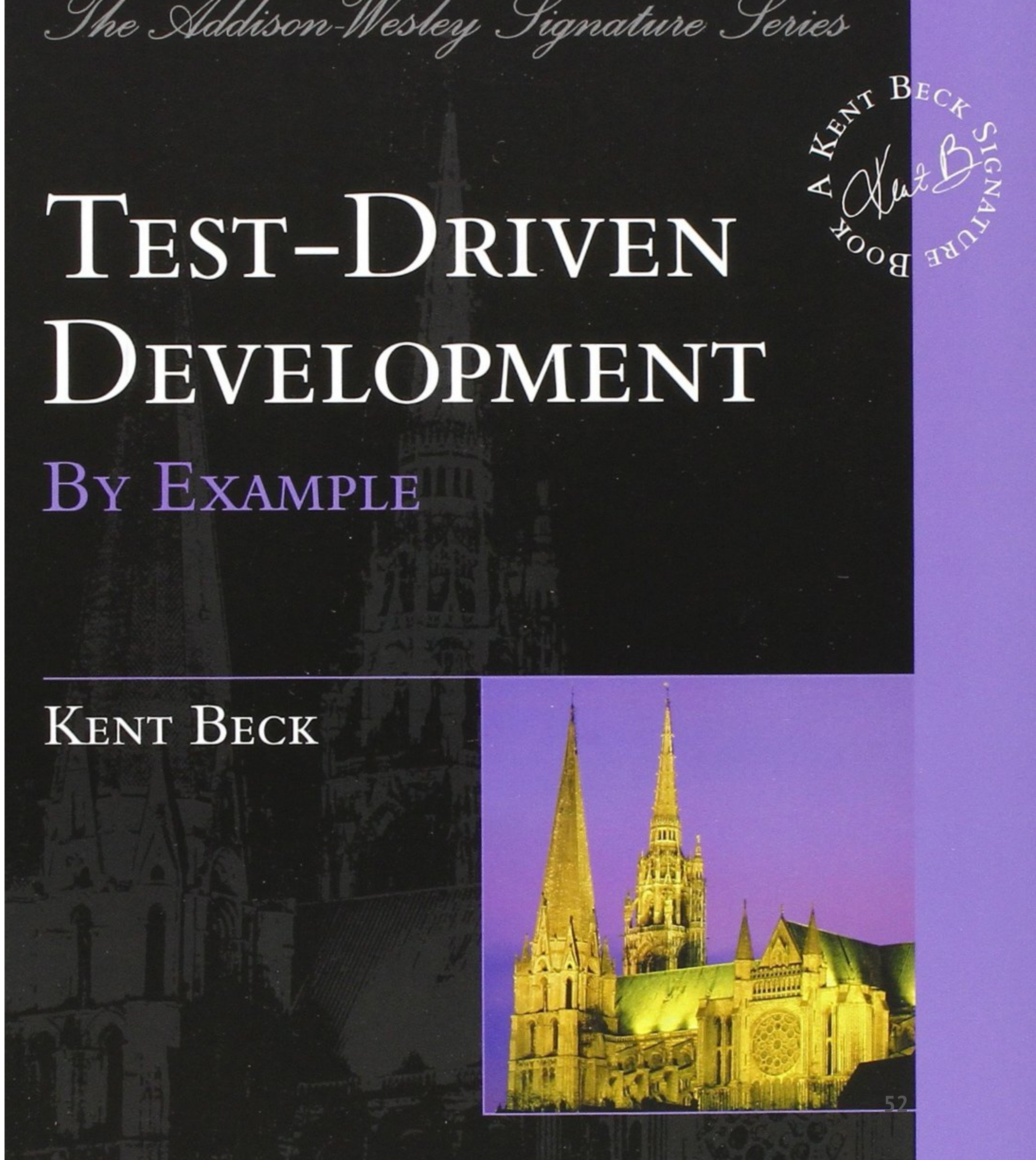
Its hard to refactor without any tests

Test Driven Development

Kent Beck

Tests are developed at the same time as the code.

Seems to elicit rather strong emotions.


The Addison-Wesley Signature Series
A Kent Beck Signature Book
Test-Driven Development
By Example
Kent Beck

In Conclusion…

Iterative Development

Its too hard to get it right the first time

Don't just stop at "It Works"

Step 4: *Refactor*

Make it work first,
then make it >>👯

fin

(p.s. I still rule)