FUNCTIONAL ALGEBRA BY EXAMPLE

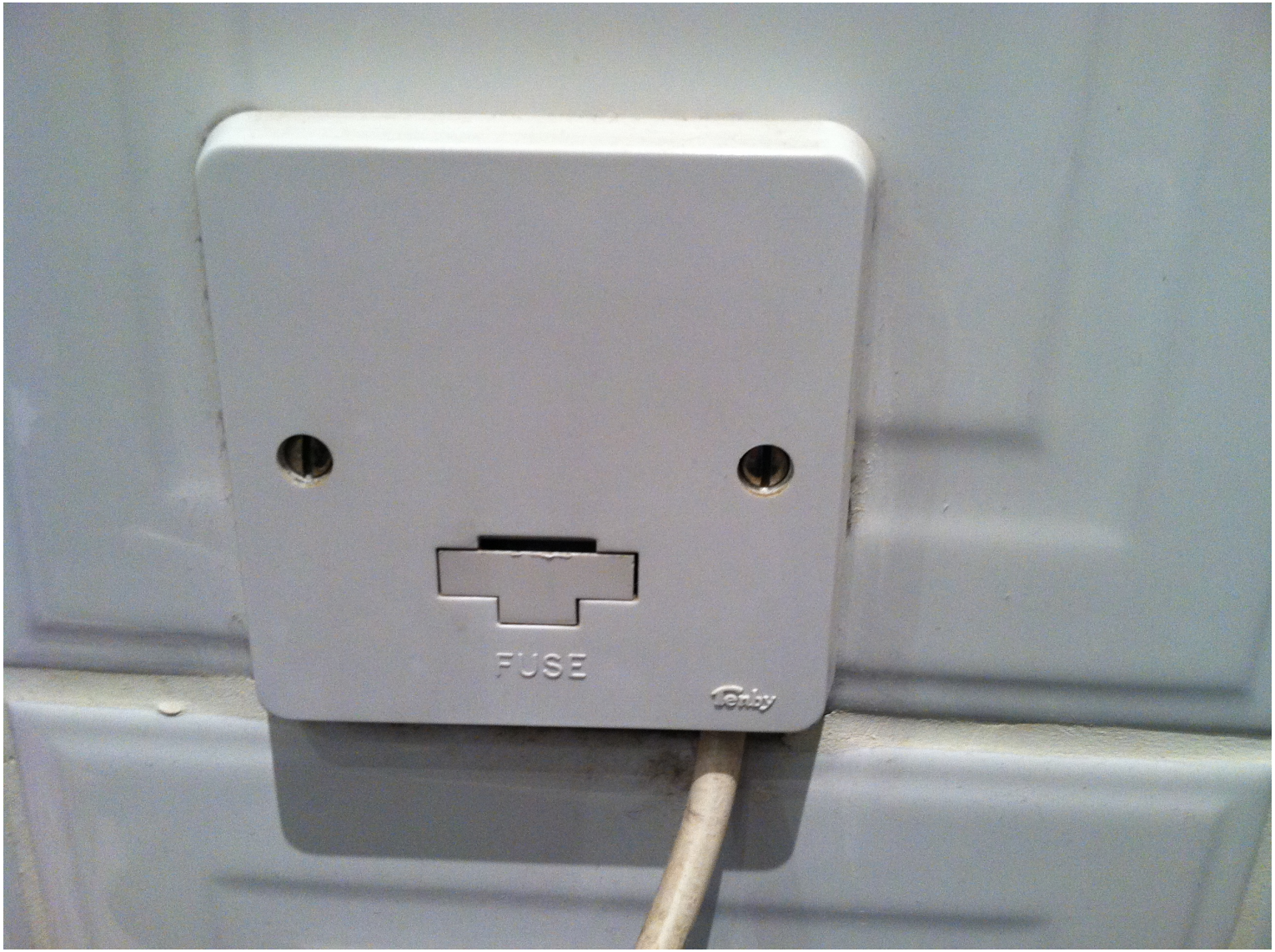
Figure: Credit http://www.flickr.com/photos/slambo_42

Susan Potter @SusanPotter

CodeMesh December 2013

FACES EVERYWHERE

OBVIOUS FACE



LESS OBVIOUS FACE

MANUFACTURED FACE

SQUINT FACE

MAYBE FACE

OO PATTERNS VS FP ABSTRACTIONS

→ **More (Subjective -> Objective)**

→ More (Ambiguous -> Precise)

→ "Fluffy" Interfaces -> Generic Functions

OO PATTERNS VS FP ABSTRACTIONS

→ More (Subjective -> Objective)

→ **More (Ambiguous -> Precise)**

→ "Fluffy" Interfaces -> Generic Functions

OO PATTERNS VS FP ABSTRACTIONS

→ More (Subjective -> Objective)

→ More (Ambiguous -> Precise)

→ **"Fluffy" Interfaces -> Generic Functions**

LAYING BRICKS

WARNING

→ **Abstract Algebra -> (Continuous, Infinite)**

→ Real World -> usually (Discrete, Finite)

WARNING

→ Abstract Algebra -> (Continuous, Infinite)

→ **Real World -> usually (Discrete, Finite)**


PANDA
GARDEN

CODE

EXAMPLE UNIX PIPE

```
1 find . -name "*.rb" \
2   | xargs egrep "#.*?TODO:" \
3   | wc -l
```

Character-based, through file descriptors

EXAMPLE FUNCTION COMPOSITION

```
1 (length . mapToUpper . sanitize) input
```

Value based, through functions

VALUING VALUES IN REAL WORLD

```
1  sealed trait PossiblyMaybe[+A]
2  final case class Somefink[A](a: A) extends
       PossiblyMaybe[A]
3  final case object Nowt extends PossiblyMaybe[
       Nothing]
4
5  object PossiblyMaybeOps {
6    def noneDefault[A](pm: PossiblyMaybe[A])(a:
         A): A = pm match {
7      case Somefink(x) => x
8      case _ => a
9    }
10 }
```

Note `_` in second match, caters for nulls

FUNCTOR

```
1 class Functor f where
2     fmap  (a → b) → f a → f b
```

SBT CONSOLE

CONTRAVARIANT FUNCTOR

```
1 class Contravariant f where
2     contramap  (b → a) → f a → f b
```

BI FUNCTOR

```
1 class Bifunctor f where
2     bimap  (a → c) → (b → d) → f a b → f c d
```

PRO FUNCTOR

```
1 class Profunctor f where
2     dimap  (c → a) → (b → d) → f a b → f c d
```

SAME TYPE, MANY INTERFACES

A type defined as a Monad can also be

→ **An Applicative**

→ A Functor

→ And possibly many others :)

SAME TYPE, MANY INTERFACES

A type defined as a Monad can also be

→ An Applicative

→ **A Functor**

→ And possibly many others :)

SAME TYPE, MANY INTERFACES

A type defined as a Monad can also be

→ An Applicative

→ A Functor

→ **And possibly many others :)**

APPLICATIONS

KNOWN USES

→ Monoids: Accumulators are everywhere, almost

→ Functors: Lots of places (endo, contravariant, bi, pro)

→ Monads: Effects, "Linear Happy Path", and more

→ Applicatives: "validations", and more

→ More … e.g. Arrows, Zippers, Lenses, etc.

THINKING ALGEBRAICALLY

→ Properties: property based testing: quickcheck, scalacheck

→ Data Types: start closed, extend using "type classes", dependent types, etc when relevant

→ Abstractions: build small building blocks, use motar to build solid walls

→ Dist Systems: using algebraic abstractions, properties to build more useful distributed systems

ROYAL FAIL

http://www.flickr.com/photos/dadavidov/

QUESTIONS

Questions?