YOW! LambdaJam

Growing a functional discipline

Susan Potter

2019-05-14

Intro

Get in touch

Accounts

GitHub	@mbbx6spp
Keybase	@mbbx6spp
LinkedIn	/in/susanpotter
Twitter	@SusanPotter

- Babysit a bloated Rails app
- Build new services in Haskell
- Test and dev tools in Haskell
- Nix/NixOS system config

I like silly hats!

Discipline

Punishment

discipline (v): to punish or penalize.

Knowledge

discipline (v): to teach by instruction and exercise

Agenda (Lessons learned)

- Learning
- Teaching
- Experiments
- Managing up


TRUST ME
I'M A PANDA

(Re-)Learning how to learn

Distribute over time

- Spread out sessions across sleeps
- Review material from prior sessions to start (recall & restore)

Spaced repetition schedule

Day	Duration	Topic
Mon	30 mins	folds over lists and trees
Tue	30 mins	unfolds over lists and trees
Wed	30 mins	pattern functors & `Fix`
Fri	30 mins	folds (`cata` & F-algebras)
next Mon	30 mins	unfolds (`ana` & Coalgebras)
next Wed	30 mins	refolds (`hylo`)

Distribute over time

- Spread out sessions across sleeps
- Review material from prior sessions to start (recall & restore)

Spaced repetition schedule

Day	Duration	Topic
Mon	30 mins	folds over lists and trees
Tue	30 mins	unfolds over lists and trees
Wed	30 mins	pattern functors & `Fix`
Fri	30 mins	folds (`cata` & F-algebras)
next Mon	30 mins	unfolds (`ana` & Coalgebras)
next Wed	30 mins	refolds (`hylo`)

Test/exercise your knowledge

DO

- Summarize key points after reading
- Devise your own exercises if none exist
- Take notes
- Flash cards, e.g.

```
Q: What is: Functor f => a -> f a
Q: ??? f => f (a -> b) -> b -> a
Q: What are the laws associated with Profunctors?
Q: Types with lawful Comonad instances?
```

DON'T just reread and highlight

Test/exercise your knowledge

DO

- Summarize key points after reading
- Devise your own exercises if none exist
- Take notes
- Flash cards, e.g.

```
Q: What is: Functor f => a -> f a
Q: ??? f => f (a -> b) -> b -> a
Q: What are the laws associated with Profunctors?
Q: Types with lawful Comonad instances?
```

DON'T just reread and highlight

Fighting Biases: Einstellung Effect

Einstellung Effect:

a person's predisposition to solve a given problem in a specific manner even though better or more appropriate methods of solving the problem exist.

- "Just restart the servers; that always fixes it...eventually."
- "I always use <insert your favorite effect or transformer library here>."
- "Just put that in the Rails monolith."

Fighting Biases: Einstellung Effect

Einstellung Effect:

a person's predisposition to solve a given problem in a specific manner even though better or more appropriate methods of solving the problem exist.

- "Just restart the servers; that always fixes it...eventually."
- "I always use <insert your favorite effect or transformer library here>."
- "Just put that in the Rails monolith."

Fighting Biases: Einstellung Effect

Einstellung Effect:

a person's predisposition to solve a given problem in a specific manner even though better or more appropriate methods of solving the problem exist.

- "Just restart the servers; that always fixes it...eventually."
- "I always use <insert your favorite effect or transformer library here>."
- "Just put that in the Rails monolith."

Fighting Biases: Einstellung Effect

Einstellung Effect:

a person's predisposition to solve a given problem in a specific manner even though better or more appropriate methods of solving the problem exist.

- "Just restart the servers; that always fixes it...eventually."
- "I always use <insert your favorite effect or transformer library here>."
- "Just put that in the Rails monolith."

Fighting Biases: Functional Fixedness

Functional Fixedness:

a cognitive bias that limits a person to use an object only in the way it is traditionally used.

- "You can only deploy Node to <insert FaaS/PaaS here>."
- "Haskell is only good for parsing or finance."

Fighting Biases: Functional Fixedness

Functional Fixedness:

a cognitive bias that limits a person to use an object only in the way it is traditionally used.

- "You can only deploy Node to <insert FaaS/PaaS here>."
- "Haskell is only good for parsing or finance."

Fighting Biases: Functional Fixedness

Functional Fixedness:

a cognitive bias that limits a person to use an object only in the way it is traditionally used.

- "You can only deploy Node to <insert FaaS/PaaS here>."
- "Haskell is only good for parsing or finance."

Fighting Biases: Strategies

- Take regular breaks
- Step back, re-evaluate
- Ask pertinent people different questions
- Listen

Learning how to teach

Examples »= Definition »= Counterexamples

```
-- Examples of a pattern
-- >>> assoc (++) [1] [2] [3]
-- True
-- >>> assoc (+) 1 2 3
-- True
-- >>> assoc (*) 1 2 3
-- True
-- >>> neAppend (a :| as) (b :| bs) = a :| (as ++ b : bs)
-- >>> assoc neAppend (1 :| []) (2 :| []) (3 :| [])
-- True
assoc :: (a -> a -> a) -> a -> a -> a -> Bool
assoc f x y z = f (f x y) z == (f x (f y z))
```

Examples »= Definition »= Counterexamples

```
-- Introduce the definition
-- class Semigroup a where
--   (<>) :: a -> a -> a
instance Semigroup a => Semigroup (NonEmpty a) where
  (a :| as) <> (b :| bs) = a :| (as ++ b : bs)

-- >>> plusConcat = mkConcat (+) 0
-- >>> multConcat = mkConcat (*) 1
-- >>> appConcat = mkConcat (++) []
mkConcat :: (a -> a -> a) -> a -> ([a] -> a)
mkConcat f neutral xs = foldr f neutral xs
```

Examples »= Definition »= Counterexamples

```
class Semigroup a => Monoid a where
  mempty :: a

-- TODO: Implement Monoid instance for NonEmpty :)
```

Exercises

- Classes two times a week
- Exercises 20+ minutes a day between
- Exercises test current and prior lessons
- A couple of live demo exercises in class

Pre-test

- Class participants are at different levels
- Guage for progress (before and after comparison)
- Unsuccessful retrieval may enhance learning as per Richland, L.E., Kornell, N., & Kao, S.L. (2009). The pretesting effect: Do unsuccessful retrieval attempts enhance learning? Journal of Experimental Psychology: Applied, 15(3), 243-257.

Post-test

- Testing prompts retrieval practice
- Can be fun, low stakes, no shame
- Helps measure progress
- Immediate feedback helps avoids *illusions of competence*

Team-level learning (safe-to-fail experiments)

- Define hypothesis
- Design experiment
- Document results
- Share recommendations back to team
- Team Discusses
- Ticket cleanup for failures and successes

Managing upward

- Learn lessons from past failures
- Propose solution to the core problems
- Offer PoC evidence that solution satisfies a core requirement
- Frame results to your audience (technical vs non-technical)
- Incremental rollout, show results early, increase trust

In Closing

Recap: Growing a functional discipline involves

- learning how to learn effectively
- learning how to teach better
- allowing experimentation without incurring the debt
- setting expectations above to promote trust

Recap: Growing a functional discipline involves

- learning how to learn effectively
- learning how to teach better
- allowing experimentation without incurring the debt
- setting expectations above to promote trust

Recap: Growing a functional discipline involves

- learning how to learn effectively
- learning how to teach better
- allowing experimentation without incurring the debt
- setting expectations above to promote trust

Recap: Growing a functional discipline involves

- learning how to learn effectively
- learning how to teach better
- allowing experimentation without incurring the debt
- setting expectations above to promote trust

Questions?

Accounts

GitHub	@mbbx6spp
Keybase	@mbbx6spp
LinkedIn	/in/susanpotter
Twitter	@SusanPotter

- functionalprogramming.rocks

Hats!