

Engaging, Large-Scale Functional Programming Education in Physical and Virtual Space

Kevin Kappelmann, Jonas Rädle, Lukas Stevens

July 28, 2022

Technical University of Munich

lambda
DAAS
28-29 JULY 2022
KRAKÓW | POLAND

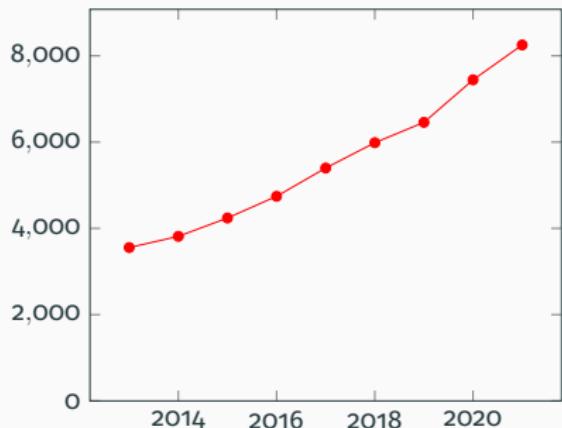
Challenges

Soaring Enrolments

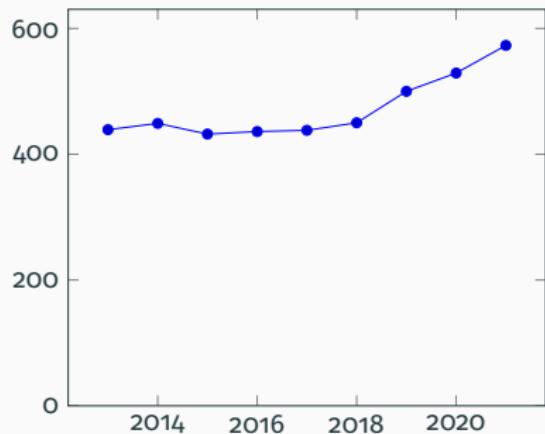
1. Number of Computer Science students exploded

Soaring Enrolments

Example: Computer Science at TU Munich



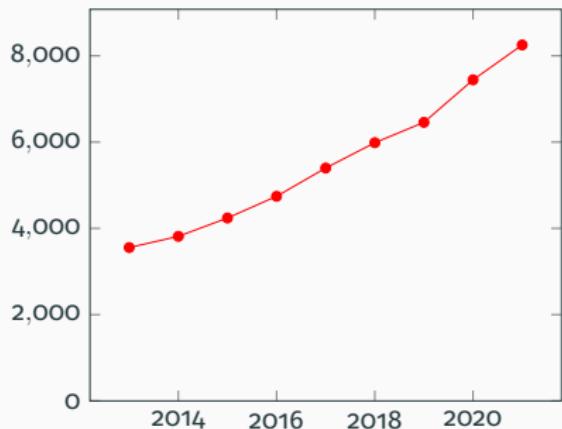
Number of CS students
(132% increase)



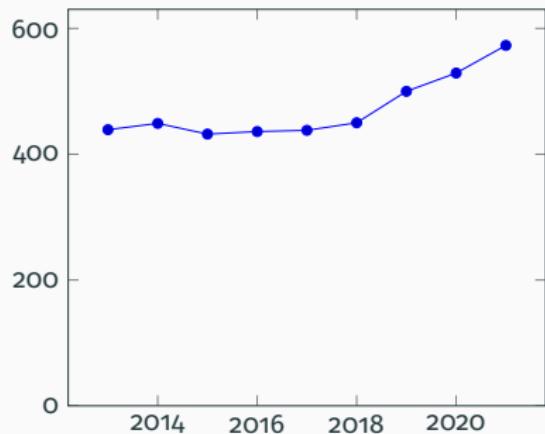
Number of CS academic staff
(31% increase)

Soaring Enrolments

Example: Computer Science at TU Munich



Number of CS students
(132% increase)



Number of CS academic staff
(31% increase)

1000+ students per course are the new normal

Soaring Enrolments

Faced by

- 2019: 13 student assistants
- 2020: 22 student assistants

The Pandemic

2. Radical transition to online classes

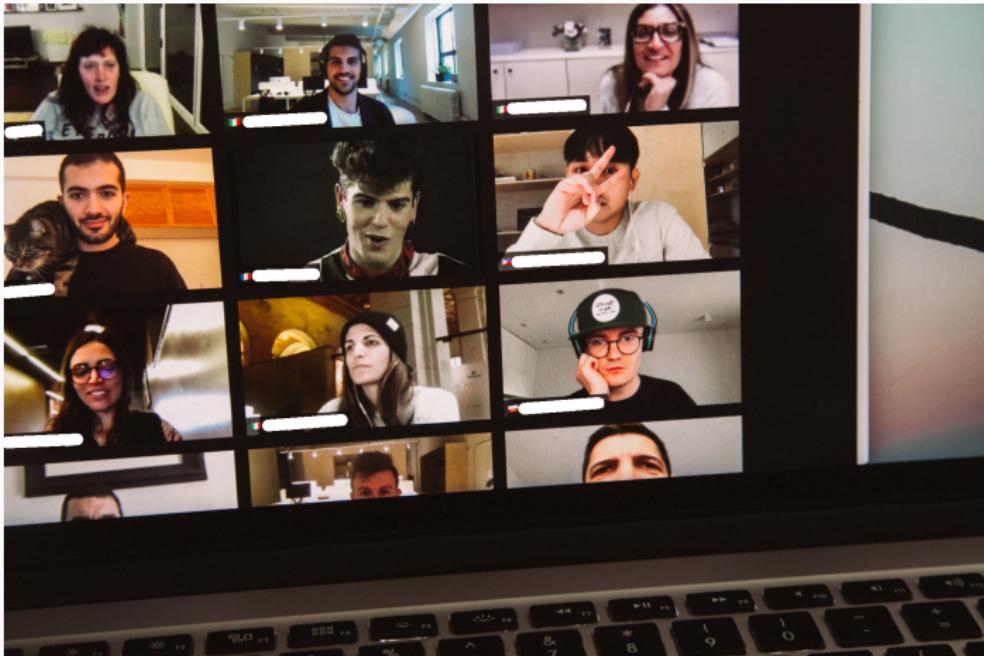
The Pandemic

How can we go from here...



The Pandemic

to here...



The Pandemic

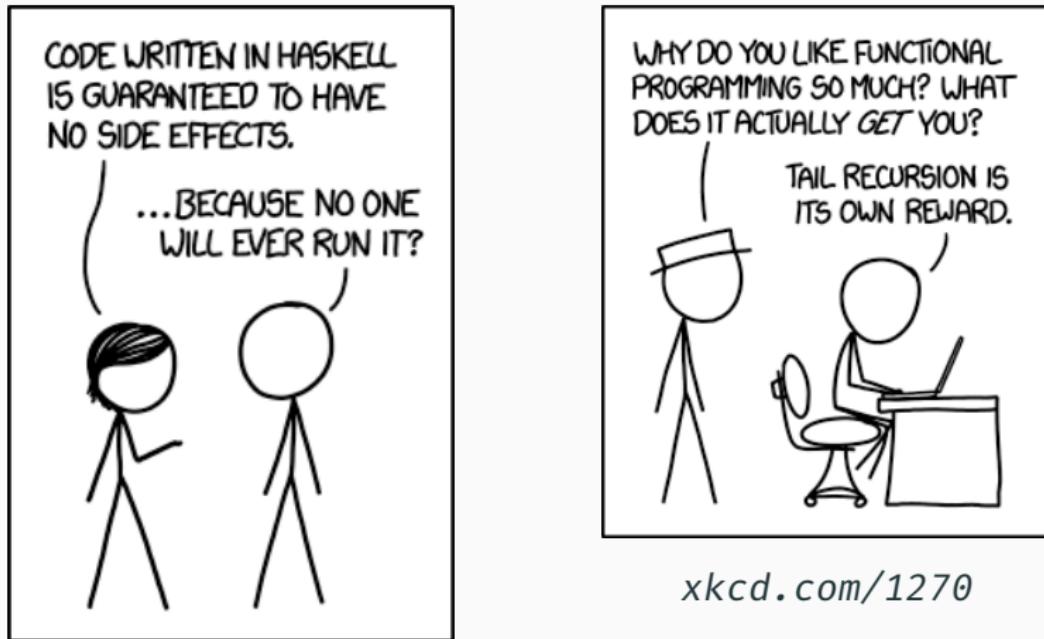
without ending up here?



Usefulness of Functional Programming

3. Students question the usefulness of functional languages beyond academia

Usefulness of Functional Programming



xkcd.com/1312

There is hope!

- We managed to cope with all these challenges

There is hope!

- We managed to cope with all these challenges
- We share our insights, tools, and exercises for other educators

You can find our resources on:

hub.com/kappelmann/engaging-large-scale-functional-programming

There is hope!

- We managed to cope with all these challenges
- We share our insights, tools, and exercises for other educators

You can find our resources on:

hub.com/kappelmann/engaging-large-scale-functional-programming

Note: We used Haskell, but most ideas apply to any functional programming course

Practical Part

Practical Part

Engagement Mechanisms

Instant Feedback

Feedback must come fast!

Instant Feedback

Feedback must come fast!

- Automated testing and feedback

Feedback must come fast!

- Automated testing and feedback
 - *ArTEMiS* runs tests, manages scores, offers exam mode,...

Feedback must come fast!

- Automated testing and feedback
 - *ArTEMiS* runs tests, manages scores, offers exam mode,...
 - *Tasty* combines QuickCheck, SmallCheck, and HUnit tests

Feedback must come fast!

- Automated testing and feedback
 - *ArTEMiS* runs tests, manages scores, offers exam mode,...
 - *Tasty* combines QuickCheck, SmallCheck, and HUnit tests
 - *Check Your Proof* for automated proof checking

Feedback must come fast!

- Automated testing and feedback
 - *ArTEMiS* runs tests, manages scores, offers exam mode,...
 - *Tasty* combines QuickCheck, SmallCheck, and HUnit tests
 - *Check Your Proof* for automated proof checking
- Manual reviews turned out to be inefficient...

Feedback must come fast!

- Automated testing and feedback
 - *ArTEMiS* runs tests, manages scores, offers exam mode,...
 - *Tasty* combines QuickCheck, SmallCheck, and HUnit tests
 - *Check Your Proof* for automated proof checking
- Manual reviews turned out to be inefficient...
 - *HLint* offers feedback more directly

Workshops With Industry Partners

Functional programming is practical!

Workshops With Industry Partners

Functional programming is practical!

- In 2020, we hosted 3 workshops with industry partners

Workshops With Industry Partners

Functional programming is practical!

- In 2020, we hosted 3 workshops with industry partners
 1. Design patterns for functional programs
 2. Networking and advanced IO
 3. User interfaces and functional reactive programming

Workshops With Industry Partners

Functional programming is practical!

- In 2020, we hosted 3 workshops with industry partners
 1. Design patterns for functional programs
 2. Networking and advanced IO
 3. User interfaces and functional reactive programming
- Great success: 120 registrations for 105 spots

Workshops With Industry Partners

Functional programming is practical!

- In 2020, we hosted 3 workshops with industry partners
 1. Design patterns for functional programs
 2. Networking and advanced IO
 3. User interfaces and functional reactive programming
- Great success: 120 registrations for 105 spots
- In some cases, workshop extended for multiple hours

Competitions

Offer challenges to go beyond the syllabus!

Competitions

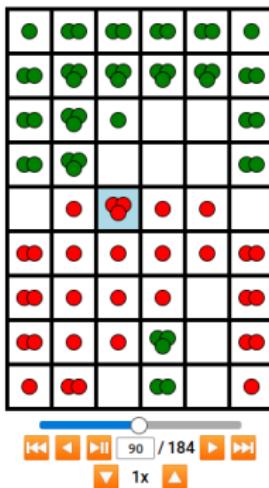
Offer challenges to go beyond the syllabus!

- Diverse, weekly competition exercises

Competitions

Tobias Markus vs. Severin Schmidmeier

Winner: ● Severin Schmidmeier



Stats

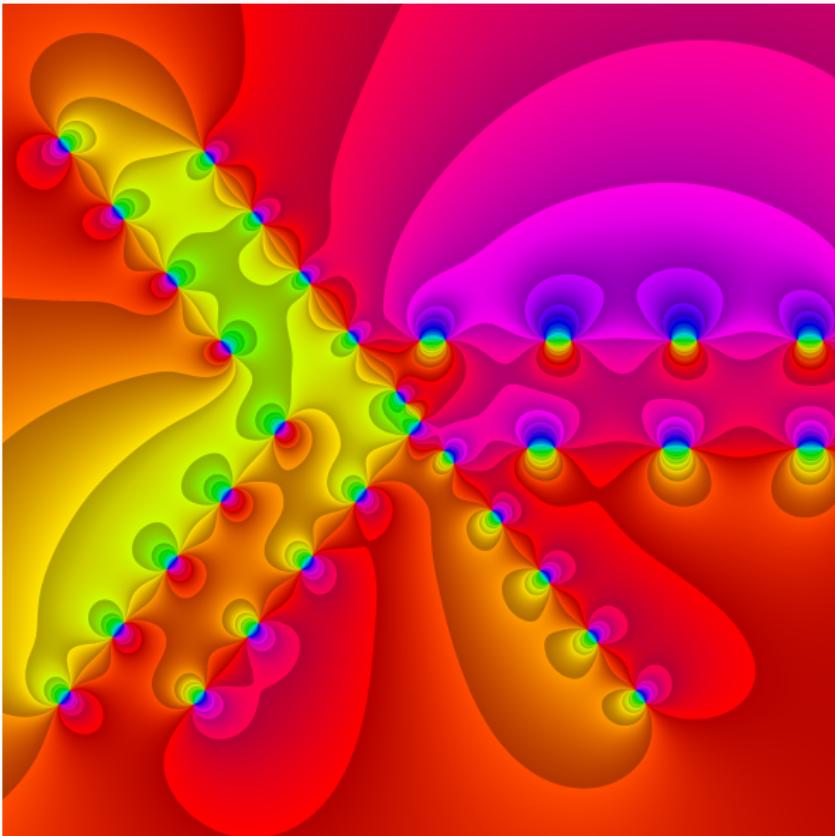
Statistic	Tobias Markus	Severin Schmidmeier
Moves made	49	49
Orbs captured	40	89
Capture/loss ratio	0.4494	2.2250

Competitions

Scoreboard (FROZEN)



Competitions



Competitions

```
module Exercise_13 where

import Data.Bool (bool)
import Data.Maybe (fromMaybe)
import Data.List (stripPrefix, isPrefixOf, findIndex, genericIndex)
import Data.Char (ord)
import Data.Word (Word8)
import qualified Data.ByteString as B
import Transform

animate :: [(String, Transform -> Transform)] -> String -> [String]
animate a s = map svg $ scanl (flip applyAnim) (parseInput s) $ map (:) [] a

paint :: String -> String
paint = svg . parseInput
```

Competitions

Offer challenges to go beyond the syllabus!

- Diverse, weekly competition exercises
- Awards for top 30 students

Competitions

Offer challenges to go beyond the syllabus!

- Diverse, weekly competition exercises
- Awards for top 30 students
- Works extremely well to motivate talented students.

I/O Mocking

Motivation

- Submissions (primarily) tested with QuickCheck

Motivation

- Submissions (primarily) tested with QuickCheck
- I/O is an important part of the syllabus

Motivation

- Submissions (primarily) tested with QuickCheck
- I/O is an important part of the syllabus

So how do we test I/O in Haskell?

The Standard Way

```
copyFile :: FilePath -> FilePath -> IO ()  
copyFile = _
```

The Standard Way

```
copyFile :: MonadFileSystem#(m) =>
            FilePath -> FilePath -> m()
copyFile = _
```

The Standard Way

```
import qualified Prelude
import Prelude hiding (readFile, writeFile)

class Monad m => MonadFileSystem m where
    readFile :: FilePath -> m String
    writeFile :: FilePath -> String -> m ()

copyFile :: MonadFileSystem m =>
            FilePath -> FilePath -> m ()
copyFile = _
```

The Standard Way

```
import qualified Prelude
import Prelude hiding (readFile, writeFile)

class Monad m => MonadFileSystem m where
    readFile :: FilePath -> m String
    writeFile :: FilePath -> String -> m ()

copyFile :: MonadFileSystem m =>
            FilePath -> FilePath -> m ()
copyFile source target = do
    content <- readFile source
    writeFile target content
```

Multiple Instantiations

```
instance MonadFileSystem IO where
    readFile = Prelude.readFile
    writeFile = Prelude.writeFile
```

Multiple Instantiations

```
instance MonadFileSystem IO where
    readFile = Prelude.readFile
    writeFile = Prelude.writeFile

data MockFileSystem =
    MockFileSystem (Map FilePath String)
instance MonadFileSystem (State MockFileSystem) where
    readFile = _
    writeFile = _
```

The Problem

What is the problem with

```
copyFile :: MonadFileSystem m =>
            FilePath -> FilePath -> m ()
copyFile = _
```

The Problem

What is the problem with

```
copyFile :: MonadFileSystem m =>  
          FilePath -> FilePath -> m ()  
copyFile = _
```

Lack of transparency!

The Solution

Delay mocking to the compilation stage

The Solution

Delay mocking to the compilation stage

by replacing the *IO* module with a mixin.

The Mixin

```
data RealWord = RealWord {  
    workDir :: FilePath,  
    files :: Map File Text,  
    handles :: Map Handle HandleData,  
    user :: IO (),  
    ...  
}
```

The Mixin

```
data RealWord = RealWord {
    workDir :: FilePath,
    files :: Map File Text,
    handles :: Map Handle HandleData,
    user :: IO (),
    ...
}

newtype IO a = IO { unwrapIO ::  
    ExceptT IOException (PauseT (State RealWorld)) a }
```

The Pause Monad

```
class Monad m => MonadPause m where
    pause :: m ()
    stepPauseT :: m a -> m (Either (m a) a)
```

An Example Interaction

```
main = do
  x <- getLine
  putStrLn $ "Hello " ++ x
```

```
user s = do
  hPutStrLn stdin s
  output <- hGetLine stdout
  when (_)
```

Check Your Proof

CYP In A Nutshell

- Operates on a strict, untyped subset of Haskell

CYP In A Nutshell

- Operates on a strict, untyped subset of Haskell
- Automatically checks simple inductive proofs

CYP In A Nutshell

- Operates on a strict, untyped subset of Haskell
- Automatically checks simple inductive proofs
- Integrates with Tasty

Background Theory

```
data List a = [] | a : List a
```

```
[] ++ ys = ys
```

```
(x : xs) ++ ys = x : (xs ++ ys)
```

```
goal xs ++ (ys ++ zs) .=. (xs ++ ys) ++ zs
```

The [] Case

Lemma: $xs \text{ ++ } (ys \text{ ++ } zs) \text{ .=} (xs \text{ ++ } ys) \text{ ++ } zs$

Proof by induction on List xs

Case []

To show: $[] \text{ ++ } (ys \text{ ++ } zs) \text{ .=} ([] \text{ ++ } ys) \text{ ++ } zs$

Proof

$$[] \text{ ++ } (ys \text{ ++ } zs)$$

(by def ++) $\text{.=} ys \text{ ++ } zs$

(by def ++) $\text{.=} ([] \text{ ++ } ys) \text{ ++ } zs$

QED

The Cons Case

Case $x : xs$

To show: $(x : xs) ++ (ys ++ zs)$

$$\therefore= ((x : xs) ++ ys) ++ zs$$

IH: $xs ++ (ys ++ zs) \therefore= (xs ++ ys) ++ zs$

Proof

$$\begin{aligned} & (x : xs) ++ (ys ++ zs) \\ (\text{by def } ++) \quad & \therefore= x : (xs ++ (ys ++ zs)) \\ (\text{by IH}) \quad & \therefore= x : ((xs ++ ys) ++ zs) \end{aligned}$$

$$\begin{aligned} & ((x : xs) ++ ys) ++ zs \\ (\text{by def } ++) \quad & \therefore= (x : (xs ++ ys)) ++ zs \\ (\text{by def } ++) \quad & \therefore= x : ((xs ++ ys) ++ zs) \end{aligned}$$

QED

QED

Our Experience With CYP

- Student feedback 18 positive, 3 negative

Our Experience With CYP

- Student feedback 18 positive, 3 negative
- Main criticism: lack of documentation

Our Experience With CYP

- Student feedback 18 positive, 3 negative
- Main criticism: lack of documentation
- Mostly well-structured inductive proofs in the exam

Find more in our repository!

- ACM-ICPC-like programming contest framework
- A music synthesiser
- More engagement mechanisms and insights, our technical setup,...

[thub.com/kappelmann/engaging-large-scale-functional-programming](#)

Future Work

Future Work

Preventing collaboration/cheating



Any questions?

Thanks to Tobias Nipkow, Manuel Eberl, our student assistants, our industry partners (Active Group, QAware, TNG Technology Consulting, and Well-Typed), and our 2000 Haskell students