

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
D A λ S

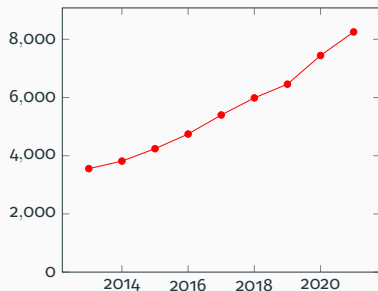
28-29 JULY 2022
KRAKÓW | POLAND

Challenges

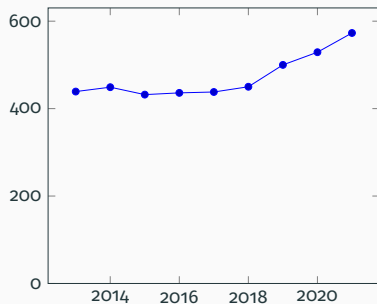
1. Number of Computer Science students exploded

Soaring Enrolments

Example: Computer Science at TU Munich



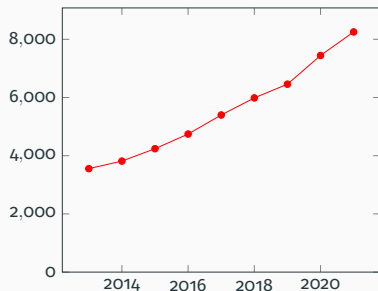
Number of CS students
(132% increase)



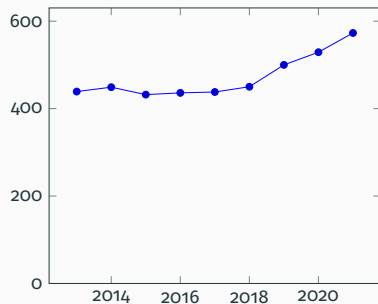
Number of CS academic staff
(31% increase)

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

2. Radical transition to online classes

The Pandemic

How can we go from here...



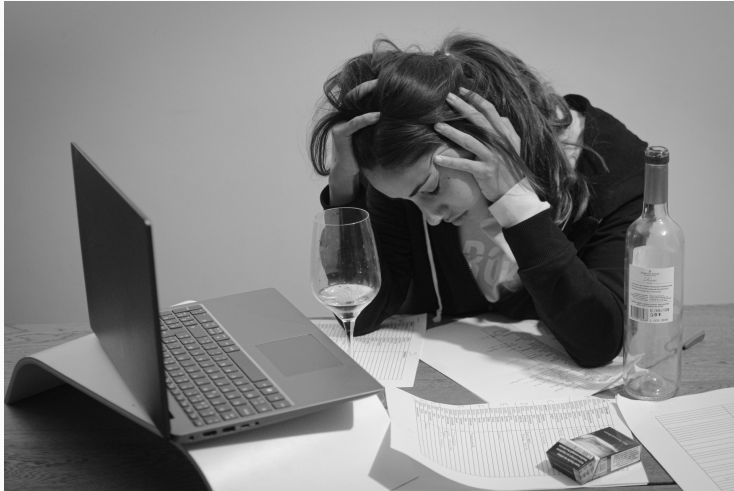
The Pandemic

to here...



The Pandemic

without ending up here?



3. Students question the usefulness of functional languages
beyond academia

Usefulness of Functional Programming



xkcd.com/1312



xkcd.com/1270

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:

github.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:

github.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

Feedback must come fast!

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

Instant Feedback

Lemma: $xs ++ (ys ++ zs) .=. (xs ++ ys) ++ zs$

Proof *by induction on List xs*

Case $[]$

To show: $[] ++ (ys ++ zs) .=. ([] ++ ys) ++ zs$

Proof

$[] ++ (ys ++ zs)$

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

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

QED

Case $x : xs$

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

IH: $xs ++ (ys ++ zs) .=. (xs ++ ys) ++ zs$

Proof

...

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

Functional programming is practical!

Functional programming is practical!

- In 2020, we hosted 3 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

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

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

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

Maybe you want to offer a workshop as well? :)

Offer challenges to go beyond the syllabus!

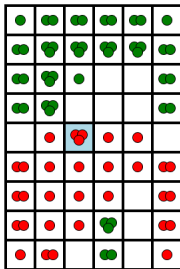
Offer challenges to go beyond the syllabus!

- 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

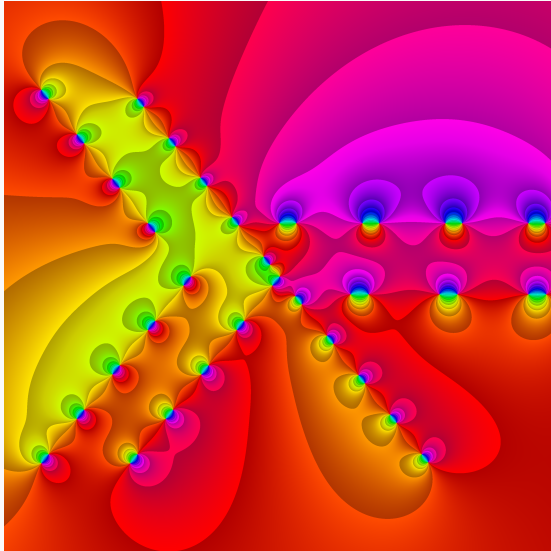
Competitions

Scoreboard (FROZEN)



Place	Team						
1.	haskellhackers						
	0 17:23:27	5 17:42:00	0 17:16:05	5 18:29:07	0 19:35:35	0 17:47:02	5 19:31:05
2.	ghzi						
	0 17:15:27	1 18:36:54	13 20:27:54	1 18:17:40	0 19:52:40	1 17:29:31	4 20:42:05
3.	maol						
	0 17:11:54	7 17:59:03	0 17:10:48	7 19:27:56	2 18:46:35	3 18:30:55	2 21:00:30
4.	gbs2021						
	6 18:33:43	1 17:28:10	3 18:19:06	7 19:06:18	2 19:51:59	1 17:52:30	5 20:52:36
5.	piniaturtles						

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
```

Offer challenges to go beyond the syllabus!

- Diverse, weekly competition exercises
- Works extremely well to motivate talented students.

Offer challenges to go beyond the syllabus!

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

Offer challenges to go beyond the syllabus!

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

Maybe you want to offer awards or competitions as well? :)

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.readFile
```

Multiple Instantiations

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

```
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 RealWorld = RealWorld {  
  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

Student submission

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

Mock user

```
user s = do
  hPutStrLn stdin s
  out <- hGetLine stdout
  when (out /= _)
    (fail $ _)
```

An Example Interaction

Student submission

```
main = do
```

```
  x <- getLine
```

```
  putStrLn $ "Hi " ++ x
```

Mock user

```
user s = do
```

```
  hPutStrLn stdin s
```

```
  out <- hGetLine stdout
```

```
  when (out /= _)
```

```
    (fail $ _)
```

An Example Interaction

Student submission

```
main = do
```

```
  x <- getLine
```

```
  putStrLn $ "Hi " ++ x
```

Mock user

```
user s = do
```

```
  hPutStrLn stdin s
```

```
  out <- hGetLine stdout
```

```
  when (out /= _)
```

```
    (fail $ _)
```

An Example Interaction

Student submission

```
main = do
```

```
  x <- getLine
```

```
  putStrLn $ "Hi " ++ x
```

Mock user

```
user s = do
```

```
  hPutStrLn stdin s
```

```
  out <- hGetLine stdout
```

```
  when (out /= _)
```

```
    (fail $ _)
```

An Example Interaction

Student submission

```
main = do
```

```
  x <- getLine
```

```
  putStrLn $ "Hi " ++ x
```

Mock user

```
user s = do
```

```
  hPutStrLn stdin s
```

```
  out <- hGetLine stdout
```

```
  when (out /= _)
```

```
    (fail $ _)
```

An Example Interaction

Student submission

```
main = do
```

```
  x <- getLine
```

```
  putStrLn $ "Hi " ++ x
```

Mock user

```
user s = do
```

```
  hPutStrLn stdin s
```

```
  out <- hGetLine stdout
```

```
  when (out /= _)
```

```
    (fail $ _)
```

An Example Interaction

Student submission

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

Mock user

```
user s = do
  hPutStrLn stdin s
  out <- hGetLine stdout
  when (out /= _)
    (fail $ _)
```


An Example Interaction

Student submission

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

Mock user

```
user s = do
  hPutStrLn stdin s
  out <- hGetLine stdout
  when (out /= _)
    (fail $ _)
```

An Example Interaction

Student submission

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

Mock user

```
user s = do
  hPutStrLn stdin s
  out <- hGetLine stdout
  when (out /= _)
    (fail $ _)
```

An Example Interaction

Student submission

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

Mock user

```
user s = do
  hPutStrLn stdin s
  out <- hGetLine stdout
  when (out /= _)
    (fail $ _)
```

Find more in our repository!

- A music synthesiser, UNO framework, turtle graphics,...
- Proof checker for inductive and equational reasoning
- More engagement mechanisms and insights, our technical setup,...

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

The background is a stylized, colorful illustration of a bridge and a city skyline. The bridge is a large, arched structure with a complex truss system, rendered in shades of blue, yellow, and red. Below the bridge, there are silhouettes of city buildings in various colors. The overall style is graphic and artistic, with a soft, painterly texture.

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