USING HASKELL PROFESSIONALLY

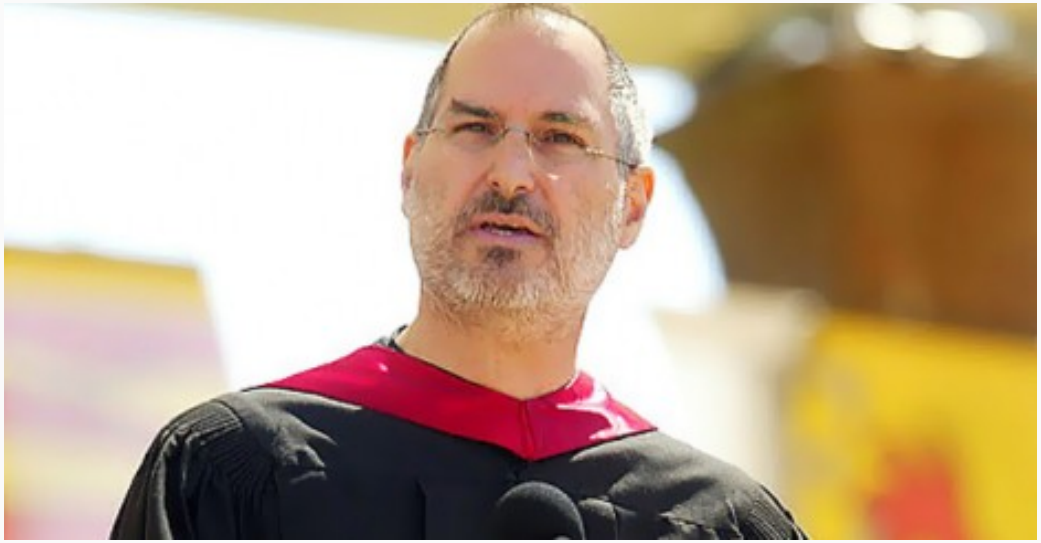
Alfredo Di Napoli

These slides are available at URL.

External links are references are provided at the end.

QUOTING STEVE JOBS..

Today, I'm gonna tell you three stories.



1. My story
2. My company story
3. Your story

THE EARLY DAYS

Back in 2007, I had my first exposure to FP, under the form of a university course. The course taught us OCaml.

The early days (contd.)

Back in 2007, I had my first exposure to FP, under the form of a university course. The course taught us OCaml.

After the initial enthusiasm, I drifted back to Python and other OOP languages.

FAST-FORWARD 2009..

I was still in love with Python. At the time I was a regular attendees of **Python-it.org**, a popular Italian community.

I fell in love with Clojure.

Pushing myself forward in my holy grail search, I was exposed to a huge number of programming language, trying to find the "perfect" one: Scheme, Clojure, Common Lisp, **Haskell**, Io, Ruby, *put-yet-another-language-here*.

So, despite the interest, I did went back to Clojure and lisp-family languages.

fast-forward 2011..

I was writing my masters degree dissertation on *Parallel and Distributed Computing*, using nothing but C++. The topic, together with the exposure to C++ revamped in me the holy flame of fast and compiled languages.

fast-forward 2011.. (contd.)

I was writing my masters degree dissertation on *Parallel and Distributed Computing*, using nothing but C++. The topic, together with the exposure to C++ revamped in me the holy flame of fast and compiled languages.

Being Haskell a compiled language, I wanted to give it another go, so I bought *Learn You Some Haskell for Great Good* and worked my way through it.

I started an internship with a company in the defense field, doing C++ in Rome. To hone my Haskell skills I tried to contribute to an Haskell open source project, the Snap framework.

mid 2012 (contd.)

Being determined in earning a living with functional programming, I decided to concentrates my efforts only on three languages, based on different criteria (commercial users, personal preference, job offers abroad):

- Haskell
- OCaml
- Scala

THE MANCHESTER ERA

Scala programmer during the day, Haskell coder at night.

2 DAY COURSE

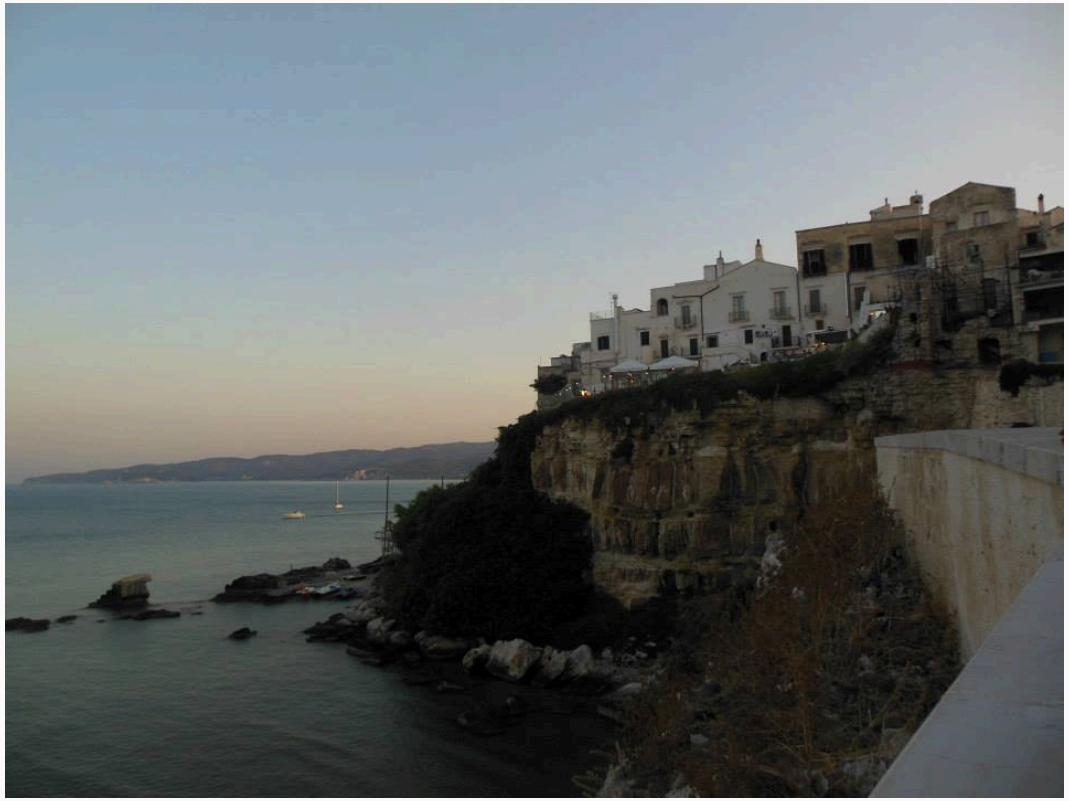
Well-Typed's Fast Track to Haskell

After a couple of months (it was July 2013) Well Typed was hiring. I decided to take pot luck and I applied.

To maximise my chances, I applied to a couple of other positions for Haskell jobs.

Despite the rejections, **I was actually able to face an entire interview doing nothing but Haskell!**

August 2013, Vieste - Italy



Got rejected by WT, but they said "A client of us might be searching soon..."

LANDING THE TECH JOB I LOVED

On the 29th of August, I applied for a Haskell job @ Iris Connect. Took a train to Brigthon, did the interview and was offered the position. **I was officially an Haskeller!**

TAKEHOME LESSONS

1. Don't be afraid to take leaps into the dark
2. Life is about opportunities, seize them
3. Try to contribute to a "famous" Haskell OSS
4. Constantly sharpen your saw
5. Be receptive, do networking

IRIS CONNECT'S STORY

1. A sharing and collaboration platform for teachers via video recording, feedback and introspection.
2. Initially build with RoR, it was rewritten from scratch in Haskell (backend) and RoR + Angular.js (frontend)

1. Effort initially started by my colleage Chris Dornan and Well Typed

WHY HASKELL?

Because software development is a marathon, not a sprint.

"It took me more time writing the specs that implementing the feature itself."

Because we are like Shlemiel the painter.

Shlemiel gets a job as a street painter, painting the dotted lines down the middle of the road.

On the first day he takes a can of paint out to the road and finishes 300 yards of the road. "That's pretty good!" says his boss, "you're a fast worker!".

The next day Shlemiel only gets 150 yards done. "Well, that's not nearly as good as yesterday, but you're still a fast worker."

The next day Shlemiel paints 30 yards of the road. "Only 30!" shouts his boss. "That's unacceptable! On the first day you did ten times that much work! What's going on?"

"I can't help it," says Shlemiel. "Every day I get farther and farther away from the paint can!"

1. The more time it pass, the farther we get from our “paint can”, the mental model we built of the system.
2. In large scale systems, you can have parts that won't be touched for *years*!
   2.1 How do you defend yourself when the refactoring or feature time comes?
3. A rich, strong and expressive type system can be your ultimate ally against complexity
   3.1 Things like `newtype`s and `ADT`s can help you cure common “diseases” like *Boolean Blindness*

As universe expands, so does the entropy in your software: use types to keep it at bay!

"Pros" of working in Haskell

1. Refactoring is a dream
2. EDSLs are a piece of cake
3. Makes impossible states unrepresentable
4. High quality libraries

REFACTORING IS A DREAM

1. The type system naturally guides you
2. In Haskell we tend to write small and generic functions
   - 2.1 Cfr. Bob Martin's "Clean Code"
   - 2.2 Most of the time they don't even break as they are written to work on polimorphic types
   - 2.3 Code reuse = profit!

So ultimately is not just about the strong type system, is about Haskell's (and Haskellers) natural tendency towards **composition** and **parametricity**.

EDSLS ARE A PIECE OF CAKE

```
fromPreset :: MediaFile -> MediaFile
           -> Maybe Atlas.VideoFilter
           -> VideoPreset -> Maybe VideoRotation
           -> LogLevel -> [T.Text]
fromPreset filename outFilePath flt vpres vi ll =
  let cli = ffmpegCLI $ mconcat [
              i $ toTextIgnore filename
            , loglevel ll
            , fromVideoPreset vpres
            , isVideoRotated vi <?> resetRotateMetadata
            , yuv420p
            , vf [rotateMb vi]
            , isJust flt <?> vf_technicolor
            , o_y_ext (toTextIgnore outFilePath) (Left vpres)
            ]
  in T.words cli
```

MAKES IMPOSSIBLE STATES UNREPRESENTABLE

Real world scenario:

```haskell
-- | Creates a new Supervisor.
-- Maintains a map <ThreadId, ChildSpec>
newSupervisor :: IO Supervisor

-- | Start an async thread to supervise its children
supervise :: Supervisor -> IO ()

-- | forkIO-inspired function
forkSupervised :: Supervisor
               -> RestartStrategy
               -> IO ()
               -> IO ThreadId
```

Example usage:

```
main = do
  sup <- newSupervisor
  supervise sup
  _ <- forkSupervised sup OneForOne $ do
         threadDelay 1000000
         print "Done"
```

Can you spot a potential bug?

Nothing in the types is forcing us to call `supervise` before actually supervise some thread!

```
main = do
  sup <- newSupervisor
  -- Wrong! We forgot to start the supervisor...
  _ <- forkSupervised sup OneForOne $ do
         threadDelay 1000000
         print "Done"
```

As Haskellers, we can certainly do better!

PHANTOM TYPES TO THE RESCUE!

Phantom Types allow us to “embed” constrain on our types, together with smart constructors.

```
data Uninitialised
data Initialised

data Supervisor_ a = Supervisor_ {
      -- record fields (omitted)
      }

type SupervisorSpec = Supervisor_ Uninitialised
type Supervisor = Supervisor_ Initialised
```

Let's now slightly change our API to be this:

```haskell
-- | Creates a new Supervisor.
newSupervisor :: IO SupervisorSpec

-- | Start an async thread to supervise its children
supervise :: SupervisorSpec -> IO Supervisor
```

What did we get? Let's try to run the "wrong" snippet again...

```
main = do
  sup <- newSupervisor
  _ <- forkSupervised sup OneForOne $ do
         threadDelay 1000000
         print "Done"
```

What did we get? Let's try to run the "wrong" snippet again...

```
main = do
  sup <- newSupervisor
  _ <- forkSupervised sup OneForOne $ do
         threadDelay 1000000
         print "Done"
```

GHC will complain:

```
Couldn't match type Control.Concurrent.Supervisor.Uninitialised
         with Control.Concurrent.Supervisor.Initialised
Expected type: Supervisor
Actual type: Control.Concurrent.Supervisor.SupervisorSpec
```

1. This is because now we require a `Supervisor` to be initialised first
2. The type system prevented us making a silly mistake
   2.1 Failed with a very useful error message
3. Profit!

This is just a small example (this is only one of the possible solutions), but the benefits are real.

SNAGS OF WORKING IN HASKELL

1. Slow(ish) Compilation
2. Cabal Hell
3. ???

slow(ish) compilation

SLOW COMPILATION: THE CAVEAT

1. It's a problem all non-interpreted languages have to deal with
2. GHC indeed does incremental compilation, building only what's changed
3. It's even slower if..
   - 3.1 You have TH (Template Haskell) in your code
   - 3.2 You are building with profiling enabled

If you want faster feedback loop, consider using ghci

What can **you** do (as a community) to embrace, support and spread functional programming?

Be pragmatic.

Don't be condescending.

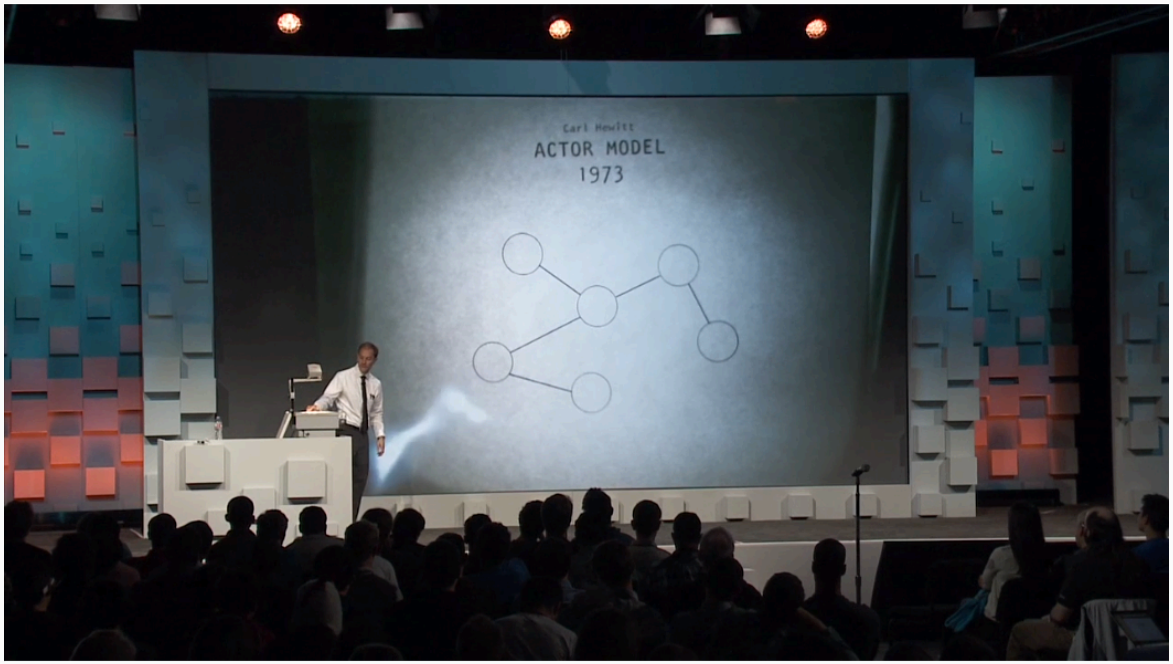
Quoting Bazerman's "Letter to a young Haskell enthusiast"

[..]Haskell wasn't built on great ideas, although it has those. It was built on a culture of how ideas are treated.[..]

[..]In functional programming, our proofs are not by contradiction, but by construction. If you want to teach functional programming, or preach functional programming, or just to even have productive discussions[..], it will serve you well to learn that ethic.[..]

Keep an open mind.

QUOTING BRET VICTOR'S "THE FUTURE OF PROGRAMMING"



[..]So the most dangerous thought that you can have as a creative person is to think that you know what you're doing. Because once you think you know what you're doing, **you stop looking around for other ways of doing things.**[..]

[..]If you want to be open or receptive to new ways of thinking, to invent new ways of thinking, I think the first step is you have to say to yourself, **"I don't know what I'm doing."**[..] I think you have to say, "we don't know what programming is".[..]

And once you truly understand that - and once you truly believe that - then you're free.

Thank you.

Questions?

My road to Haskell

http://www.alfredodinapoli.com/posts/2014-04-27-my-road-to-haskell.html

Don Stewart - Haskell in the large

http://code.haskell.org/~dons/talks/dons-google-2015-01-27.pdf

Bret Victor - The Future of Programming

http://vimeo.com/71278954

Joel Spolsky - Back to Basics

http://www.joelonsoftware.com/articles/fog0000000319.html

Gershom Bazerman - Letter to a Young Haskell Enthusiast

http://comonad.com/reader/2014/letter-to-a-young-haskell-enthusiast/