A *Pragmatic* Case For Static Typing

Slides available from github at:

https://github.com/bhurt/presentations/blob/master/statictyping.odp

Please

Hold your

Comments,
Criticisms,
Objections,
Etc.

To the end

I've been paid to work on:

50+ KLoC Clojure code bases

250+ KLoC Ocaml code bases

(plus C, C++, Java, etc.)

What do I mean by Pragmatic?

Programming

is a Means

not an End

Time To Working Code

“Working code” means...

- Written
- Compiling
- Debugged
- Documented
- Maintainable

Static Typing Reduces Time To Working Code

What I don't mean by Static typing:

Java
C++/C
Pascal
C#

What I mean by
Static Typing:

Standard ML
Ocaml
F#
Haskell

How does

Static Typing

Reduce
Time to Working Code
?

I.

The Little Things

I.a.

Static Typing reduces the time to find and fix simple bugs

A type error gives you...

- That an error exists
- The file the error is in
- What line the error is on
- What column the error starts in
- A hint as to what the problem is

… at compile time

Time to fix Static Typing Error:

Average: 10's of seconds

Worst Case: 10's of minutes

An error found by unit testing gives you...

- That a bug exists
- What module/class the bug was detected in
- The manifestation of the bug
- A stack trace (maybe)
- The ability to reproduce the bug

… and that's all

Time to fix Testing Error:

Average: Minutes

Worst Case: Hours

10's of Seconds
To
10's of Minutes

<

Minutes
To
Hours

I.B.

Static Typing eliminates null pointer exceptions

No such thing as a null value.

```
data Maybe a =
       Just a
     | Nothing
```

Example:

```
find :: (a -> bool) -> [a] -> Maybe a
```

Maybe Int ≠ Int

So:

(find f [1..100]) + 3

Is a type error.

```
case (find f [1..100]) of
    Just x -> x + 3
    Nothing -> 77
```

II.

Large scale programming

Large Scale Programming Is Different

What do I mean by “Large Scale”?

- Lots of code (10's of KloC)
- Long lived (years+)
- Many developers (3 or more)

Not all programming is large scale

(and that's OK)

II.a.

“Don't do that!”

“Don't do that!”
doesn't work!

Brian's Observation:

At 3 people on a team, there is a 50% chance that at least one of them is a full-time idiot.

As the teams grow larger, the probability of not having an idiot on the team falls rapidly to zero.

We are all idiots some of the time.

Static Types turn
“Don't do that”
Into
“Can't do that”

```
(defn ex [ b x ]
    (if b
        (+ x 3)
        (x 3)))
```

```
data Either a b =
    Left a
    | Right b

ex :: Either Int (Int -> Int) -> Int
ex (Left x) = x + 3
ex (Right x) = x 3
```

Corollary:

The time you most need static typing is exactly the time it's most tempting to not use it.

II.b.

Documentation

In a large scale project...

Documentation is *even more important!*

In a large scale project...

Documentation is *even harder!*

Types make dandy (machine checked) documentation

Consider:

```
(a -> Bool) -> [a] -> Maybe a
```

(a -> Bool) -> [a] -> Maybe a

I know that:

- It takes some set of elements from the list
- Passes those elements into the function given
- Either returns:
  - Just an element from the list
  - Nothing
- Doesn't do any I/O or other side effects

II.c.

Maintenance

Code
Must
Change

Changing the behavior
of a function
is not necessarily wrong

Example:

… -> Int

Becomes
… -> Maybe Int

Becomes
… -> Either Int String

Becomes
… -> Either Int ErrorMessage

A function changed behavior:

What needs to get fixed now?

Unit testing:
“Something broke”

Static Typing:
“This call right here is wrong”

III.

Multithreading

Multithreading Is Coming

Multithreading

- Moore's law is now:

  2x cores/2 years

- 8 cores now
  256 cores in 10 years
  8,192 cores in 20 years

Multithreaded Code Is Different

The Four Horsemen of The Parallel Apocalypse

- Race Conditions
- Deadlocks
- Livelocks
- Priority Inversions

Every
piece of mutable state needs to come with ironclad guarantees that either:

- accesses to it are properly synchronized

- only one thread accesses it


WAIT FOR IT
.....Wait for it.....
motifake.com

Monads

Monads
represent a computation
(producing a value)
performed in a domain
where some condition is true.

The Monad...	Means the computation...
IO a	Can do I/O, etc.
STM a	Must be in a transaction
ST s a	Must execute in a single thread s
Maybe a	May not produce a value

Functions declare what domain they are in by the monad of their return type.

e.g.

```
hGetLine :: Handle -> IO String
```

```
readTVar :: TVar a -> STM a
```

```
writeTVar :: TVar a -> a -> STM ()
```

Monads
Allow us to prove
We don't have
“Four Horsemen” bugs

Software Transactional Memory (STM)

- “Like database transactions for mutable variables”
- Gives 'A', 'C', and 'I' of ACID
- Solves all four horsemen

A tale of two STMs

- C#:
  - Many developers
  - 2 years
  - FAILURE
- Haskell:
  - Few developers (SPJ & a grad student?)
  - “a long weekend”
  - SUCCESS

The problems with STM

- Performing non-transactional side effects while in a transaction
- Accessing transactional variables not in a transaction

Monads prevent side effects within a transaction.

atomically :: STM a -> IO a
“Performs the transaction (causing side effects)”

??? :: IO a -> STM a
No such function!

Can't do that!

Monads enforce being in transaction to access mutable cells.

```
readTVar :: TVar a -> STM a

writeTVar :: TVar a -> a -> STM ()
```

A similar trick for Single Threaded Access

```
newSTRef :: a -> ST s (STRef s a)

readSTRef :: STRef s a -> ST s a

writeSTRef :: STRef s a -> a -> ST s ()

runST :: (forall s. ST s a) -> a
```

What you should think...

Or maybe...

IV.

Static Types

vs.

(Lisp) Macros

Statically typed languages don't use macros

(to a first approximation)

C++, Java, etc.:
“Code and data are different things.”

Lisp, Haskell:
“Code and data are the same thing”

Lisp: “All code is data”

Leads to:

- Homoiconic representation (s-expressions)
- Macros (code that manipulates other code)

Haskell: “All Data is Code”

Leads to:

- Lazy Evaluation
- Combinators (including Monads)

“All code is data”

≈

“All data is code”

In Review...

- Reduced bug fix time
- No more NPEs
- Machine checked documentation
- (Some) protection against idiots (including you)
- Maintainable code
- Protection against the four horsemen
- Powerful abstractions
- But most importantly...

Static Typing means

Working Code Sooner

fini

Addendums

A Brief

And Needlessly Provocative

History of Functional Programming

In 1936, Alonzo Church Invents Lisp

In 1936, Alonzo Church
Invents Lisp

Except...

He called it “the Lambda Calculus”

He thought he was doing mathematics

In 1936, Alonzo Church
Invents Lisp

Except...

He called it “the Lambda Calculus”

He thought he was doing mathematics

Problem:

Lambda Calculus still allows for paradox and invalid proofs.

So, In 1940, Alonzo Church
Invents Haskell

Except...

He called it
“the Simply Typed Lambda Calculus”

Still thought he was doing mathematics

Kurt Gödel's response:

In 1958, John McCarthy realizes that Alonzo Church was doing

Programming

NOT

Mathematics

Problem:

Lisp still allows for buggy and wrong programs.

So, in 1973, Robin Milner steals the
Simply Typed Lambda Calculus
And renames it “Meta-Language”
(aka ML)

Alan Turing's Response:

John Carmack quotes are from:

http://www.altdevblogaday.com/2011/12/24/static-code-analysis/

“if you have a large enough codebase, any class of error that is syntactically legal probably exists there.”

- John Carmack

“Anything that isn’t crystal clear to a static analysis tool probably isn’t clear to your fellow programmers, either. “

- John Carmack

“A lot of the serious reported errors are due to modifications of code long after it was written.”

- John Carmack

“The classic hacker disdain for “bondage and discipline languages” is short sighted – the needs of large, long-lived, multi-programmer projects are just different than the quick work you do for yourself.”

- John Carmack

“If you aren’t deeply frightened about all the additional issues raised by concurrency, you aren’t thinking about it hard enough.”

- John Carmack

“The first step is fully admitting that the code you write is riddled with errors. That is a bitter pill to swallow for a lot of people, but without it, most suggestions for change will be viewed with irritation or outright hostility. You have to *want* criticism of your code.”

- John Carmack

Real Statically Typed Languages have...

- Type inference
  - No need to explicitly annotate most types
- REPLs
- A succinct type notation
- Powerful types
  - More than just type variables (templates/generics)

Important Note:

Static Typing

vs.

Unit Testing
Agile Development
Etc.

Important Note:

Static Typing

AND

Unit Testing
Agile Development
Etc.