Becoming Friends With Errors

April 26, 2016

Who am I ?

Ashutosh Raina

- http://ashutoshraina.github.io
- **@ashutoshraina**

Objective

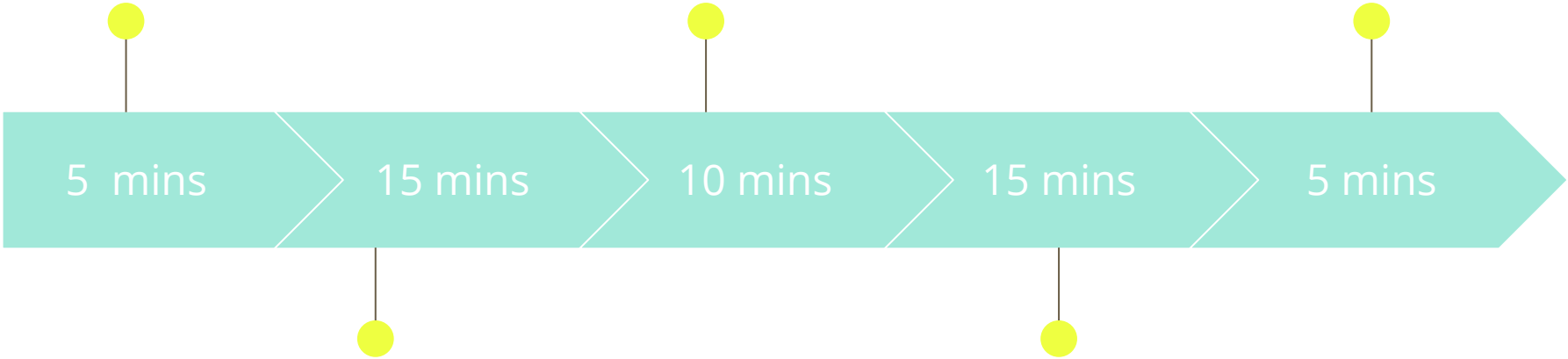
- Embrace errors as first class citizens in the domain
- Design with Errors
- Learn from the language ecosystem
- Errors in an Async world

What is an error ?

Hint : It’s not an exception !!

Learning from other languages

Conclusion



Motivation for designing with Errors in mind

Errors in an Async world

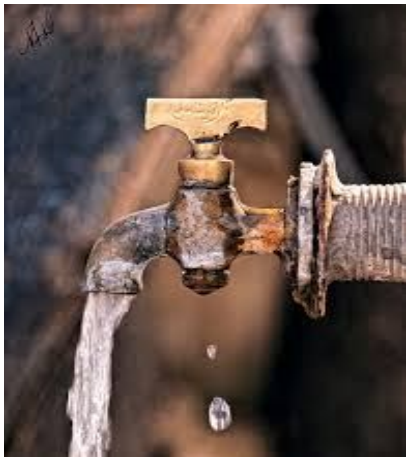
There is no happy path in programming

- Everything can fail, we need to account for it
- Failure/Error is domain information

Error

- Error > Error Code > Exception
- Error is not exception
- Exception is not Error
- Error Codes are least desirable

The Curious Case of Null



- Meaningless, no Type information
- Null is absence of knowledge
- Absence of Signal != noise

Motivational Example

```
public static SqlConnection GetConnection(string connectionName)
{
    var connectionString = ConfigurationManager.AppSettings[connectionName];
    var connection = new SqlConnection(connectionString);
    connection.Open ();
    return connection;
}
```

Problem

Problem

Problem

Motivational Example

- There is no happy path in systems
- Anything that can fail, will fail
- All failures are different, don’t loose the failure context

Motivational Example

- A “rest” api returns error codes instead of Http Status Codes
- Why can’t I connect to my database ?
  - Are my credentials wrong ?
  - Did the database die ?
  - Did the database node die ?
  - Am I talking to wrong database ?

Error handling in other Languages

- RUST
- SWIFT

RUST

```rust
enum Option<T> {
    None,
    Some(T),
}

enum Result<T, E> {
    Ok(T),
    Err(E),
}
```

```rust
use std::num::ParseIntError;
fn double_number(number_str: &str) -> Result<i32, ParseIntError> {
    match number_str.parse::<i32>() {
        Ok(n) => Ok(2 * n),
        Err(err) => Err(err),
    }
}
fn main() {
    match double_number("10") {
        Ok(n) => assert_eq!(n, 20),
        Err(err) => println!("Error: {:?}", err),
    }
}
```

SWIFT

```
enum UserError: ErrorType {
  case InvalidEmail
  case PasswordExpired(daysSinceExpired: Int)
}
```

```
var userService = UserService()
do {
  try getUserByEmail("foo@bar.com")
} catch UserError.InvalidEmail {
  print("Invalid Email.")
} catch UserError.PasswordExpired {
  print("Password Expired")
}
```

Error in the async land : Will you get back to me ?

- Slow processes cause timeouts
- We never got a response back
- Anything that can go wrong, will go wrong

Error in the async land : Will you get back to me ?

- Can I use my earlier abstractions ?
- Yes, we can !!

Conclusion

- We can borrow concepts from all languages and make our own life easier
- We didn’t use the word Monad or FP or Haskell
- Errors are friendly dragons with no fire
- Let’s be friends with errors

Thank you

Ashutosh Raina

- http://ashutoshraina.github.io
- **@ashutoshraina**