Alexey Golev

# On Types and Typing

## in JavaScript

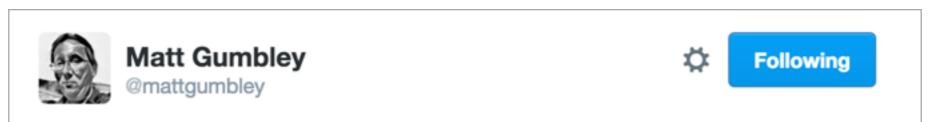
## Kübler-Ross

## Stage 1 Denial

## But it's untyped

- V8: Hidden classes
- SpiderMonkey: One type JS::Value type-tagged values that represent the full range of JavaScript values
- JavaScriptCore: JSValue
- Chakra: `Type`
- Unityped

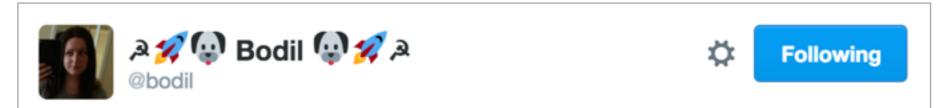
- Your code is unlikely to have almost 100% coverage.
- TDD is optional. Static type system is not
- Refactoring is still a minefield



"TDD replaces a type checker in a dynamically typed language in the same way that a bottle of whisky replaces your daily problems"



Any TDD system with 100% coverage contains an ad hoc, informally-specified, bug-ridden, slow implementation of a type checker.



People in my mentions inevitably "yeah but actually if you've got a comprehensive enough test suite you don't need types."



I don't even understand the argument—why spend such effort writing tests for something a simple type signature could prove without a doubt?

```
it('should return a function', () \Rightarrow {
  expect(someFunction('name'))
  .toBeA('function')
/* aflow */
declare function someFunction(
  x: string
): (props: Array<string>) ⇒ string
```

```
expect(data).to.be.equal('function')
assert.isFunction(result.componentDidMount)
assert.isFunction(result.componentWillUnmount)
assert.isArray(result.someProperty)
```

"Program testing can be used to show the presence of bugs, but never to show their absence"

- Edsger W. Dijkstra, Notes On Structured Programming

"A type system is a tractable syntactic method for proving the absence of certain program behaviors by classifying phrases according to the kinds of values they compute."

-Benjamin C. Pierce, Types and Programming Languages

#### Stage 2

## Anger

"WITHOUT GETTING TOO TECHNICAL ABOUT IT: IT IS JUST A PAIN IN THE ASS TO CODE WITH STATIC TYPING."



"...SINCE YOU HAVE TO BE MAKING ALL THE TYPE CASTING YOURSELF, AND BE VERY CAUTIOUS ABOUT IT. TYPE CASTING ERRORS ARE EVERYWHERE."

, Some 15 years of coding

## "GOOD PROGRAMMERS SHOULD NOT NEED TO BE PROTECTED AGAINST THEIR OWN MISTAKES."

- Good Programmer

"YOUR SOFTWARE CANNOT STAND TO BE QUALITY ASSURED ON STATIC-TYPING ALONE, SO WHY BOTHER PUTTING UP WITH ITS LIMITATIONS?"



#### Stage 3

## Bargaining

## Testing, contracts...

- TypedJS
- Contracts.coffee/Contracts.js (Inspired by Racket)
- rho-contracts.js
- TreatJS
- PropTypes

#### Stage 4

## Depression

## Fatigue

- JSig
- Infernu
- TypeScript
- Flow
- Elm, Purescript, ReasonML, BuckleScript, ghcjs, Idris

## Fatigue?

- JSig
- Infernu
- TypeScript
- Flow
- Elm, Purescript, ReasonML, BuckleScript, ghcjs, Idris

## Fatigue?

- JSig
- Infernu
- TypeScript
- Flow
- Elm, Purescript, ReasonML, BuckleScript, ghcjs, Idris

#### **TypeScript**

- October 1, 2012
- TypeScript
- DefinitelyTyped
- 14,971 stars
- 2,014 forks

#### Flow

- Nov 18, 2014
- OCaml
- flow-typed
- 8,558 stars
- 618 forks

### **TS**

```
interface ActionCreator extends Function {
    (...args: any□): any;
}
interface Reducer extends Function {
    (state: any, action: any): any;
}
```

### **TS**

```
interface Action { type: any; }
interface Dispatch<S> {
  <A extends Action>(action: A): A;
type Reducer<S> =
<A extends Action>(state: S, action: A) \Rightarrow S;
interface Store<S> {
  dispatch: Dispatch<S>;
  getState(): S;
  subscribe(listener: () \Rightarrow void): Unsubscribe;
  replaceReducer(nextReducer: Reducer<S>): void;
}
```

```
function dispatch(action) {
  currentState = currentReducer(currentState, action)
  var listeners = currentListeners = nextListeners
  for (var i = 0; i < listeners.length; i++) {
    var listener = listeners[i]
    listener()
  }
  return action
}</pre>
```

## Flow

```
declare type Dispatch<A: { type: $Subtype<string> }> =
  (action: A) ⇒ A;

declare type Reducer<S, A> = (state: S, action: A) ⇒ S;

declare type Store<S, A> = {
    dispatch: Dispatch<A>;
    getState(): S;
    subscribe(listener: () ⇒ void): () ⇒ void;
    replaceReducer(nextReducer: Reducer<S, A>): void
};
```

#### **TypeScript**

- Structural typing for everything
- Bivariance

#### Flow

- Structural typing for objects, interfaces and functions. Nominal for classes
- Variance of function arguments and type parameters

```
class Admin {}
class SuperAdmin extends Admin {}
function showDashboard(user: Admin): SuperAdmin {}
function showSuperDashboard(user: SuperAdmin): Admin {}
let currentAdmin = new Admin( ... )
let currentSuperAdmin = new SuperAdmin( ... )
showDashboard(currentAdmin) // ok
showDashboard(currentSuperAdmin) //ok
showSuperDashboard(currentAdmin) // error
const h: Admin =
showDashboard(currentSuperAdmin) // ok
```

```
function getAdmin(): Promise<Admin> {}

function rankUp(user: Promise<Admin>):
Promise<SuperAdmin> {}

function rankDown(user: Promise<SuperAdmin>):
Promise<Admin> {}

rankUp(getAdmin())
rankDown(getAdmin()) // error
```

### Flow extras

- (\*) Existential type
- \$Keys<T>
- {| ... |}/\$Exact<T>
- async/await
- Class<T>

## Flow extras

```
const ex: * = 5
ex.map(x \Rightarrow x + 1) //map not found in number
const ex1: any = 5
ex1.map(x \Rightarrow x + 1) //no problem - ex1 is `any`
type Person = {|
 firstName: string,
 lastName: string
|}
const john: Person = { firstName: 'John' } // error
const personKey: $Keys<Person> = 'frstName' // error
```

## Flow extras

```
class X {
    static bar(): string {
       return 'Hi';
    }
}

var a: X = new X();
a.bar(); // Type error

var b: typeof X = X;
b.bar(); // Good
```

#### **TypeScript**

- Apollostack
- Angular
- RxJs
- VSCode

#### Flow

- Relay / Draft / Jest /
   Dataloader / Native
- Radium
- Discord
- Pinterest

#### Stage 5

## Acceptance

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand."

- Martin Fowler

## Thank you