zalando

Learnings from 4 Years of TypeScript

Zalando SE

Bastian Sieker

04.12.2019

Foreword

Table of Contents

Craftsmanship

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

Martin Fowler

Good Code: Characteristics

- Functionality
- Readability
- Maintainability / Extensibility
- Testability
- Documentation
- Efficiency

Good Code: Known Principles

- Single Responsibility (SOLID)
- Reuse code (DRY)
- Keep it stupid simple (KISS)

The Last Programming Language

A Case Against the Go To Statement

Less is More: Language Features

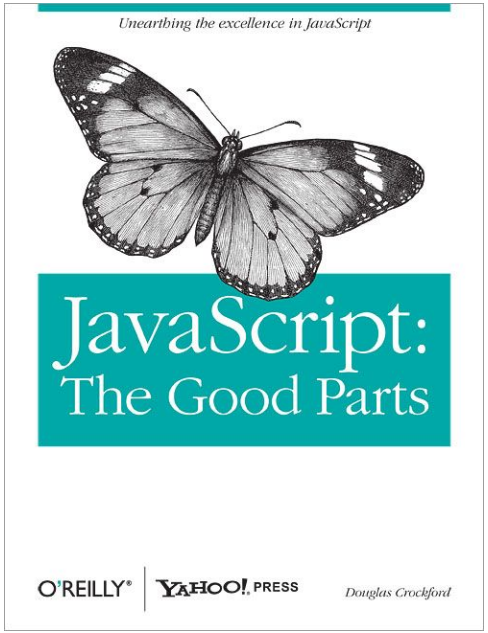
Avoiding (the right) feature can improve code:

- Reduce complexity
- Increase consistency

If you are interested in more:

- https://blog.ploeh.dk/2015/04/13/less-is-more-language-features/
- https://www.youtube.com/watch?v=ecIWPzGEbFc

Less is More: Language Features



https://covers.oreillystatic.com/images/9780596517748/lrg.jpg

zalando

Do Not Use `null`

```
// hidden complexity
typeof null === 'object';

// not so kiss
type validateEan =
    (ean: string | undefined | null) => boolean;
```

Functions

How to Define Functions #1

```
const isProduction = (env: string): boolean =>
   env === 'production';
```

- Reduce syntactic noise
- Consistency
- Return an expression

How to Define Functions #2

```
type isProduction = (env: string) => boolean;

const isProduction: isProduction = (env) =>
   env === 'production';
```

- Improve readability
- Start with type signature
- Reusable type definitions

How to Define Functions #3

```
type EnvPredicate = (env: string) => boolean;

const isDev: EnvPredicate = (env) =>
   env === 'development';

const isTest: EnvPredicate = (env) =>
   env === 'test';

const isValidNodeEnv: EnvPredicate = (env) =>
   isDev(env) || isProduction(env) || isTest(env);
```

Use `type` over `interface`

```
// union and literal types
type Shape = Square | Rectangle | Circle;
type Answer = 'yes' | 'no';

// nicer and more consistent syntax for functions
type noopType = () => undefined;

interface noopInterface {
  (): undefined;
};
```

Classes

Do Not Use `class` (and Prototype-Based Inheritance)

- Hidden complexity
- Encourage to use mutable state (not simple)
- We do not need to

Handling State with Closures

```
type productsCacheFactory = () => (productId: string) => Product;

const productsCacheFactory: productsCacheFactory = () => {

  const state: ProductMap = {};  // initial state

  // ... cache implementation code

  return (productId: string) => state[productId];
}
```

Discussing Code

Discussing Code

Ask yourself (or the team):

- Why to solve a certain problem the way you do?
- What do you like and dislike about the code you write, and why?

Discussing Code

Align in the team:

- Indent: 2 tabs or 4 spaces?
  - You ALWAYS want [whatever you want]
- Do the same on a higher level, ideas:
  - `Promise` vs `async/await`
  - Imperative vs declarative code
  - TypeScript vs JavaScript

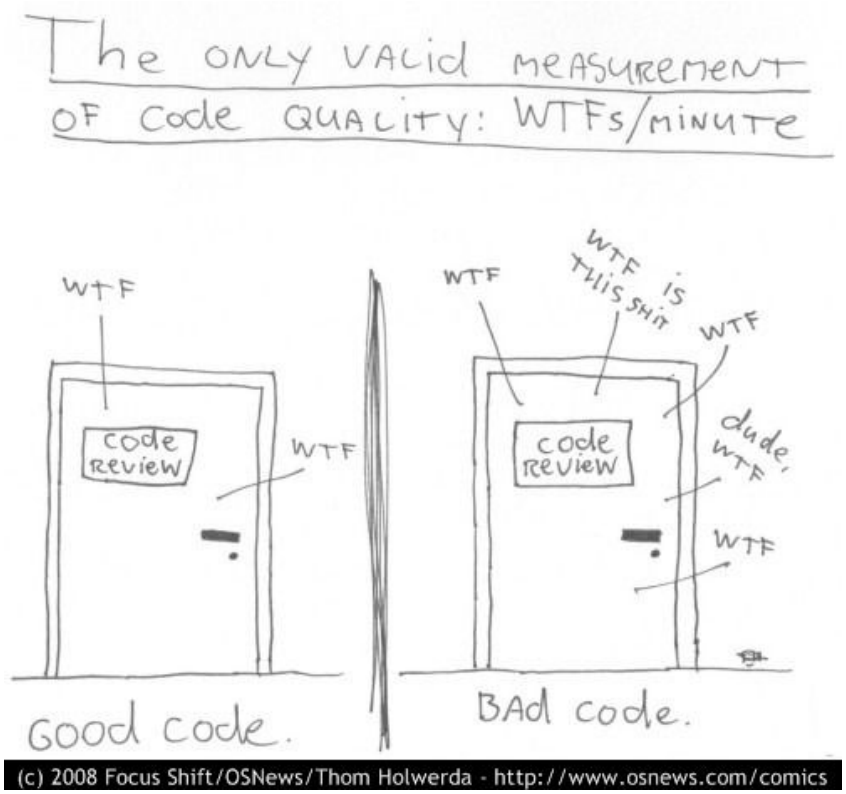
Discussing Code

Why is that a good idea?

- Consistency
- Helps onboarding new team members
- Helps to understand what you are doing

Putting it in Perspective

- My examples are heavily simplified
  - There are downsides
  - There are exceptions
- Automation may be difficult
- Whole team must be onboard
- Probably hard to maintain
- https://github.com/labs42io/clean-code-typescript



The End

- Questions?
- mail@bzums.de
- github.com/bzums

zalando