

A stylized fantasy landscape illustration. In the upper left, a black dragon flies against a light beige sky. Several small black birds are scattered across the sky. On the right, a large, bright yellow sun is partially obscured by a black silhouette of a castle with multiple spires. The foreground features dark green and black silhouettes of mountains and trees. The overall color palette is muted, with beige, olive green, and black tones.

Functional Domain Modeling

In TypeScript

told by
Nicolas Carlo

ConFoo 2025



@nicoespeon

Nicolas Carlo

Freelance Web Developer

Special skills:

- ❑ Legacy Code Sorcery
- ❑ Community Building

TS won't catch all type errors



@ts-ignore

Fighting against types...
let's bail out



any

No time to type properly...
put TS to sleep



x as Y

It's not what you think...
please trust me

It takes discipline & experience



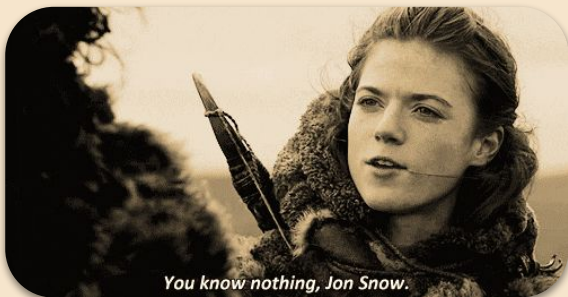
~~@ts-ignore~~

@ts-expect-error



~~any~~

unknown




x as Y

Sometimes necessary



Brace yourself, bugs are coming



As you grow, it will get worse

*aka "we spend more time fixing
regressions than shipping features"*

Let's write automated tests!

*"If you liked it,
you shoulda put a test on it"*

— The Beyoncé Rule 



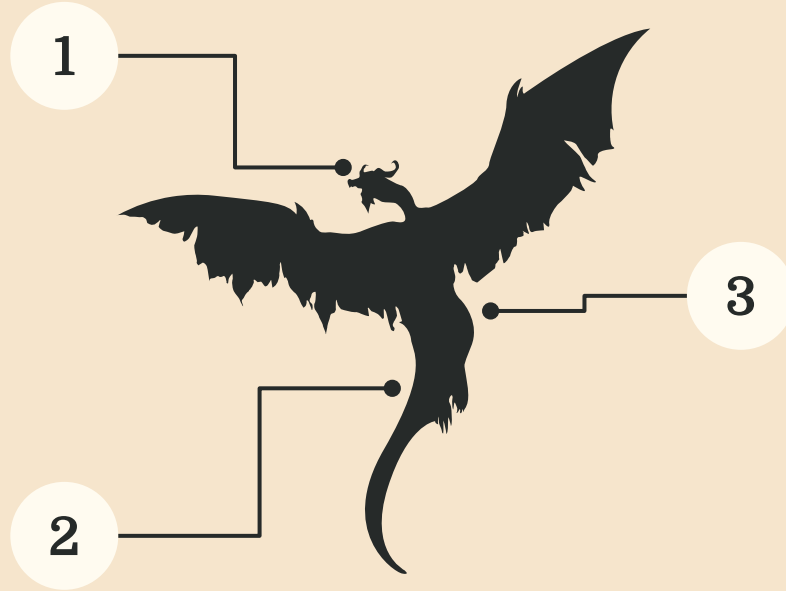
But that won't be enough...

Time

Tedious work
on legacy code

Skills

Approval Testing,
Mikado Method...



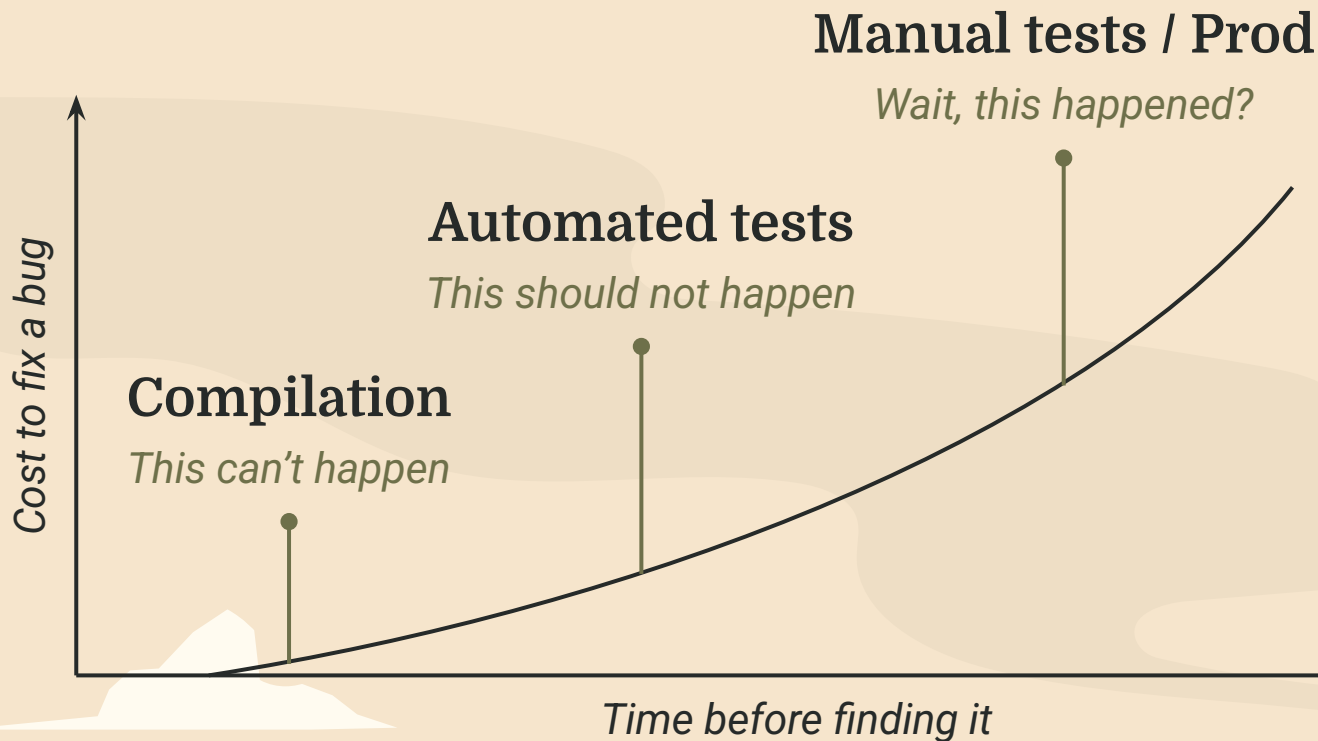
Refactoring

Easy to write tests
that fail when you
refactor the code

Hope
is not a
strategy



The faster you catch it, the better



Our battle plan: 2 spells

Discriminated Unions

Make impossible
states unrepresentable

Brand Types


Fight against
Primitive Obsession




A stylized illustration with a light beige background. In the center, the text '01' is displayed in a large, bold, black serif font. Below it, the text 'Too many booleans' is written in a smaller, black serif font. The background features two large black dragon silhouettes with spread wings, one on the left and one on the right. Several small black bird silhouettes are scattered in the sky. In the bottom right corner, there is a black silhouette of a castle with multiple towers and spires. The bottom left corner shows dark green, jagged shapes representing trees or hills.

01

Too many
booleans



```
type UserSubscription = {  
  isSubscribed: boolean  
  hasCanceledSubscription: boolean  
  isTrialPeriod: boolean  
  subscriptionEndDate: Date | null  
  trialEndDate?: Date  
}
```



5 booleans (ON or OFF)

$2^5 = 32$ representable states

How many should be *impossible*?





**You won't test
them all**

Hope
is not a
strategy



Enums are not enough in TypeScript

```
enum UserSubscription {  
    NewCustomer = "new customer",  
    InTrialPeriod = "in trial period",  
    Subscribed = "subscribed",  
    SubscriptionCanceled = "canceled",  
    SubscriptionExpired = "expired",  
}
```

TypeScript Enums
can only be
string or number

The TS equivalent of
functional Enums are
Discriminated Unions

Introduce Discriminated Unions

```
type UserSubscription =  
  | { status: "new customer" }  
  | { status: "in trial period"; trialEndDate: Date }  
  | { status: "subscribed"; subscriptionEndDate: Date }  
  | { status: "subscription canceled"; subscriptionEndDate: Date }  
  | { status: "subscription expired"; subscriptionEndDate: Date }
```



Only list the possible states of your Domain

Can't represent impossible ones = fewer tests to write

What client code looks like

```
function getExpirationWarnings(subscription: UserSubscription): string[] {  
    if (  
        subscription.status === "in trial period"  
        && isWithinAWeek(subscription.trialEndDate)  
    ) {  
        return ["Your trial period is ending soon"];  
    }  
  
    // ...  
    return [];  
}
```

Move defensive logic to TypeScript

```
type UserSubscription =
```

```
  | NewCustomer
```

```
  | UserInTrial
```

```
  | UserSubscribed
```

```
  | UserSubscriptionCanceled
```

```
  | UserSubscriptionExpired;
```

```
declare function cancel(subscription: UserSubscribed): UserSubscriptionCanceled
```

Pros & cons of Discriminated Unions

Reduce states count

Fewer possible scenarios is
easier to manage at scale

Native TypeScript

You don't need a type library
to introduce the pattern

Express your Domain

It helps you pause and think
about business logic


Need to teach it

You need to change the way
your team do things



The background is a stylized illustration with a warm, orange-brown color palette. It features a large, dark silhouette of a castle with multiple towers and spires on the left side. In the upper right, a large dragon with spread wings is flying, accompanied by several smaller birds. The sky is filled with soft, white, cloud-like shapes. The bottom of the image shows a dark, silhouetted landscape with rolling hills and trees.

02


Primitive Obsession



```
export type Employee = {  
  id: string  
  fullName: string  
  skills: string[]  
  privateInfo: {  
    referrals: number  
    phone: string  
    dateOfBirth: string  
    monthlySalary: number  
  }  
}
```



Can you see more
impossible states?



The image features a central rectangular piece of light beige, textured paper with irregular, torn edges. It is set against a solid dark grey or black background. Scattered around the paper are several small, black, stylized bird silhouettes in flight. There are also four white, irregular, cloud-like or smoke-like shapes: one in the top-left corner, one in the top-right corner, one in the bottom-left corner, and one in the bottom-right corner. The text is centered on the paper in a dark, bold, sans-serif font.

**string and number
are often too loose**

3 pernicious problems with Primitives



Confusion

TypeScript won't prevent John to pass *fullName* instead of *id* by mistake



Security

The surface area of potential injections or bad data is huge



Duplication

You need to check for the same conditions everywhere in the code
(and it's not consistent)

Type Aliases can only document

```
type Phone = string
type Money = number

type EmployeeInfo = {
  age: number
  phone: Phone
  dateOfBirth: string
  monthlySalary: Money
}
```



It can improve readability,
but it won't be effective

Introduce Brand Types

```
type Phone = string & { readonly type: 'Phone' }
```



Phone can be assigned to a `string`

But only `Phone` can be assigned to a `Phone`

Define a Brand<Type, Name>

```
// unique symbol = can't conflict with other properties  
declare const __brand__: unique symbol  
  
type Brand<BaseType, BrandName> = BaseType & {  
  readonly [__brand__]: BrandName  
}
```



What client code looks like

```
type Phone = Brand<string, 'Phone'>
```

```
declare function addPhone(employee: Employee, phone: Phone): Employee
```


Confusion is gone, a valid type is enforced



**“But how do I create a
valid Phone?”**

A single source of truth



```
declare function validatePhone(phone: string): Phone 
```

TypeScript will push your colleagues to use this

Pit of Success

It solves the **Duplication** and helps with **Security**

What validation code looks like

```
function validatePhone(phone: string): Phone {  
    if (phone.length < 10) {  
        throw new Error("Phone number is too short");  
    }  
    // ... more rules  
  
    return phone as Phone;  
}
```

Pros & cons of Brand Types

Pit of Success

Helps others do the right thing when in a rush

Remove Duplication

When you get the type, you *know* it was validated

Redefine base operations

Quantity + Quantity works, but returns a number

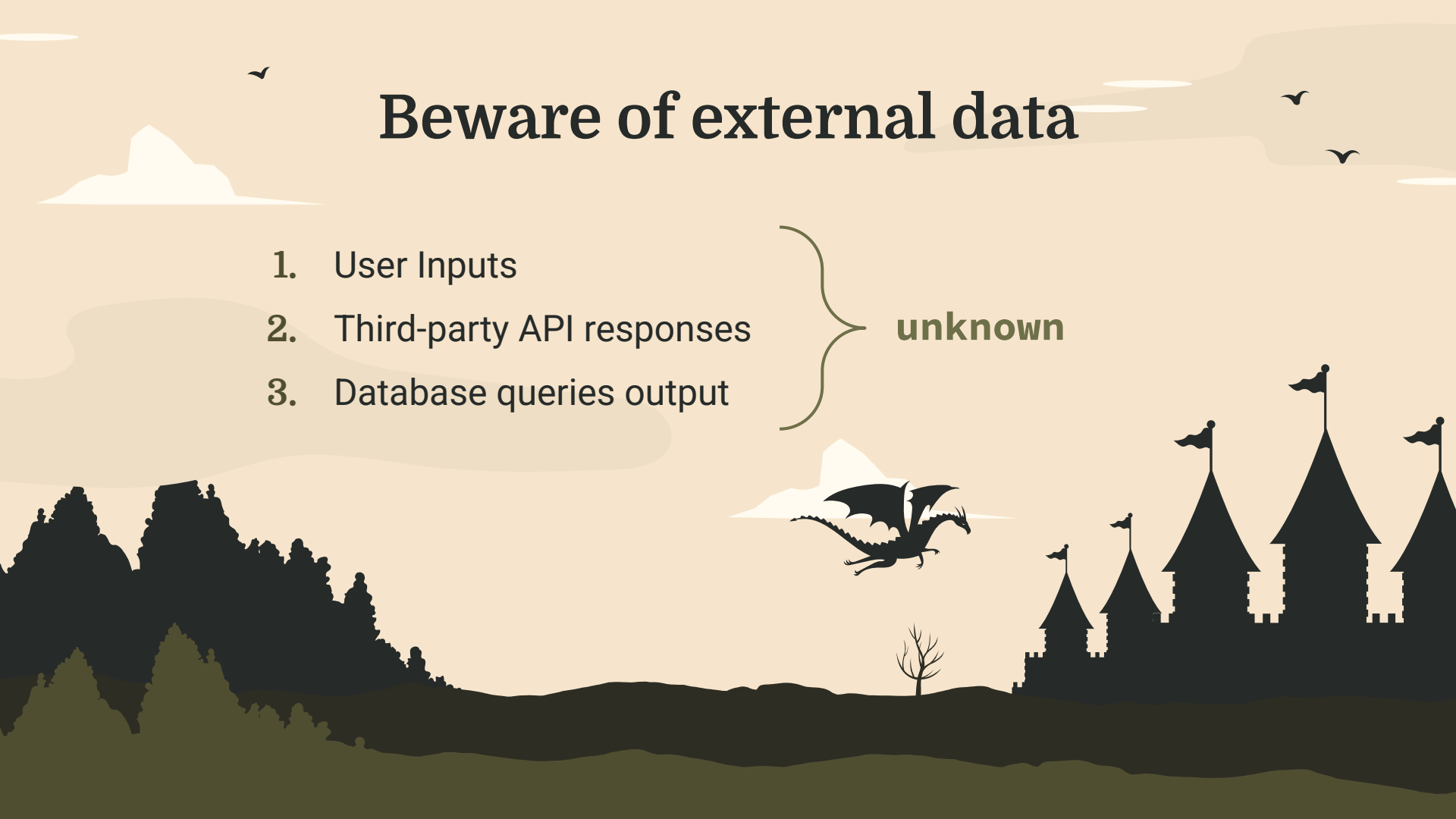
Need to teach it

You need to change the way your team do things

Beware of external data

1. User Inputs
2. Third-party API responses
3. Database queries output

unknown



Delegate this work to Zod

```
const Phone = z.string()  
  .brand("Phone")  
  .refine(validatePhone)  
  
type Phone = z.infer<typeof Phone>
```



```
const validPhone = Phone.parse(incomingData)
```

zod.dev





You are ready for battle!

Discriminated Unions & Brand Types



Fewer states

Impossible scenarios
made unrepresentable



Faster than tests

Catch bugs at
compilation time



Native TS code

You don't need an
external lib (*but use Zod*)



Pit of Success

Easy to write
maintainable code



No duplication

Express your Domain in
a single source of truth



Secure by Design

Bonus side-effect of
expressing your Domain

The journey is only beginning...



Pattern Matching

ts-pattern

to make client code easier
to read and compose

State Machines

xstate

to manage transitions
between possible states

References, for curious minds

1. [ts-reset](#) - Matt Pocock
2. [Domain Modeling Made Functional](#) - Scott Wlaschin
3. [Secure by Design](#) - Dan Bergh Johnsson, Daniel Deogun, Daniel Sawano
4. [Software Engineering at Google](#) - Titus Winters, Tom Manshreck, Hyrum K. Wright
5. [Domain Data Modeling using TypeScript Aliases, Brand Types and Value Objects](#)
- Egghead course by Tomasz Ducin

Farewell

Let's keep in touch!



[Nicolas Carlo](#)



[@nicoespeon](#)

CREDITS: This presentation template was created **by Slidesgo**, including icons **by Flaticon**, infographics & images **by Freepik**

