

How to `onCommunicate` `isClearly()` ?!

Thomas 'gossi' Gossmann

GOAL

Improve communication within an interdisciplinary team

WHERE

I will shake and disrupt your naming skills

WHY

Trigger different ways of thinking for new vocabulary

1. Why we are not `onCommunicating isClearly()` !
2. How to `onDetect` to `isNotCommunicatingClearly()` ?
3. How to communicate clearly :)

Part I

Why we are not onCommunicating isClearly() !

Why we are NOT onCommunicating isClearly() !

Bold Statement:

- onPurpose
- onIntentionally
- inFullAwareness

=> Because of theLanguage we onUse

Developers speak ...

dev-lish

- Dialects:
 - CRUDlish
 - RESTlish
 - FrontendLish
 - BackendLish

Developers speak ...

dev-lish

- Dialects:
 - CRUDlish
 - RESTlish
 - FrontendLish
 - backend_lish

Example: A Developer in a Restaurant...

“Hey <FoodOrderAndDeliveryProvider>, here is my onOrder handler and whenever you think I am isReady, please call it”

Nouns vs. Verbs

we eat nouns

we buy nouns from the store

we sit on nouns

we sleep on nouns

Nouns

boring - just things

static

Verbs

interesting

dynamic

Developers and English Language (Verbs, Nouns, ...)

- Not all of us are native english speakers
- Technical language has a very high gravity on our words (CRUD, REST, click)
- The two hardest problems in computer science:
 0. Cache invalidation → We'll receive education to solve the hardest algorithms
 1. Naming things → Who had a linguistic course?

Part II

How to onDetect to isNotCommunicatingClearly() ?

Methods for Detection

1. The Sound of Code
 2. Find the Imposter
 3. Do you speak Domain Language?
- The Trap we Built

Method 1: The sound of code?

Sonofication of Code

- Does the code tell a story?
- Is the story the same of the business/domain?
- Does the code *sound* wrong?

Method 1: The sound of code

1. Underline self-named words

```
interface ThingArgs {
  thing: {
    isOn: boolean;
    onClick: () => void; // will set isOn to true
  };
}

export class ThingComponent extends Component<ThingArgs> {
  @action
  onClick(): void {
    this.args.thing.onClick();
  }

  static template = hbs`
    <Icon @icon={{this.thing.isOn}} />
    <Button @onClick={{this.onClick}}>
      on
    </Button>
  `;
}
```

Method 1: The sound of code

1. Underline self-named words

Thing

onClick

isOn

```
interface ThingArgs {  
  thing: {  
    isOn: boolean;  
    onClick: () => void; // will set isOn to true  
  };  
}  
  
export class ThingComponent extends Component<ThingArgs> {  
  @action  
  onClick(): void {  
    this.args.thing.onClick();  
  }  
  
  static template = hbs`  
    <Icon @icon={{this.thing.isOn}} />  
    <Button @onClick={{this.onClick}}>  
      on  
    </Button>  
  `;  
}
```

Method 1: The sound of code

1. Underline self-named words

Thing

onClick

isOn

2. Connect the words in code execution order to tell a story:

onClick the button to onClick
the thing to be isOn

```
interface ThingArgs {  
  thing: {  
    isOn: boolean;  
    onClick: () => void; // will set isOn to true  
  };  
}  
  
export class ThingComponent extends Component<ThingArgs> {  
  @action  
  onClick(): void {  
    this.args.thing.onClick();  
  }  
  
  static template = hbs`  
    <Icon @icon={{this.thing.isOn}} />  
    <Button @onClick={{this.onClick}}>  
      on  
    </Button>  
  `;  
}
```


Method 1: The sound of code

1. Underline self-named words

Thing

onClick

isOn

2. Connect the words in code execution order to tell a story:

onClick the button to onClick
the thing to be isOn

3. Find the Action:

```
interface ThingArgs {  
  thing: {  
    isOn: boolean;  
    onClick: () => void; // will set isOn to true  
  };  
}  
  
export class ThingComponent extends Component<ThingArgs> {  
  @action  
  onClick(): void {  
    this.args.thing.onClick();  
  }  
  
  static template = hbs`  
    <Icon @icon={{this.thing.isOn}} />  
    <Button @onClick={{this.onClick}}>  
      on  
    </Button>  
  `;  
}
```

Method 1: The sound of code?

3. The Action:

```
<Button @onClick={{this.onClick}}>
```

Method 1: The sound of code?

3. The Action:

```
<Button @onClick={{this.onClick}}>
```

4. Reverse Engineer code into User Story:

As a...

User

I want to...

onClick a button

So that...

I can onClick

Method 2: Find the Imposter



Verbs...

- Are the **actions/instructions** into a system
- Are questions to **ask** for **facts** about the system

Commands

```
requestNounTermination()  
sellNoun()  
purchaseNoun()
```

Queries

```
whichNounMethod()  
isNounAvailable()  
canPurchase(noun)
```

= Command-Query-Separation (C-Q-S)

Method 2: Find the Imposter

- Are your actions ... commanding?
- Are your questions.... asked?

Thing	✓
onClick	✗
isOn	✗

```
interface ThingArgs {
  thing: {
    isOn: boolean;
    onClick: () => void; // will set isOn to true
  };
}

export class ThingComponent extends Component<ThingArgs> {
  @action
  onClick(): void {
    this.args.thing.onClick();
  }

  static template = hbs`
    <Icon @icon={{this.thing.isOn}} />
    <Button @onClick={{this.onClick}}>
      on
    </Button>
  `;
}
```

Are your Actions ... commanding?

onClick the button to **onClick**
the **thing** to be **isOn**

- Turn code into prose
- **Actions** are the **verbs** in your sentence
- Grammar: **Subject verb object**.

Are your Actions ... commanding?

```
class Customer {  
  payWithCreditCard(thing) {  
    // pay here  
  }  
}
```

- Sound: **Customers** **pays** (for) a **thing**.
- As Question: Can a **customer** **pay** (for) a **thing**?

Are your Actions ... commanding?

```
class Customer {  
  onPayWithCreditCard(thing) {  
    // pay here  
  }  
}
```

- Sound: **Customers on** (for) a **thing**.
- As Question: Can a **customer on** (for) a **thing**?

! THIS SENTENCE NO VERB !



onVerbNoun

... wants to be command, “does” on

Are your Questions ... asked?

```
if (hasPotatoes) {  
    // what is true here?  
}
```

*“When you go to the supermarket,
can you bring 5 eggs and if they
have potatoes can you bring 10?”*

Jack. 32.

A 10eggs developer

Are your Questions ... asked?

```
let eggs = 5;  
  
if (hasPotatoes) {  
  eggs = 10  
}
```

Question known: `hasPotatoes`

Question asked: `hasPotatoes()`

Verbs vs. Facts

- The **answer** of a **question** is a **fact** about the system
- **Facts** are for conditions

```
const potatoesAvailable = hasPotatoes();  
  
if (potatoesAvailable) {  
  
}
```

Computed Facts

```
class Supermarket {  
    get potatoesAvailable() {  
        return this.potatoes.length > 0;  
    }  
}
```

Imposter **Facts** or **Questions**?

```
class Supermarket {  
  get hasPotatoes() {  
    return this.potatoes.length > 0;  
  }  
}
```



get **verbNoun**()

- ... wants to be a **fact**, masked as a **question**
- ... wants to be a **question**, masked as a **fact**

Where is this coming from?

Do we transport the `<element onclick="">` into dev-lish?

Hungarian Notation?

- bNoun → isNoun, hasNoun
- fnVerb → onVerb

The Hungarian Notation of Frontend?

Method 3: Do you Speak Domain Language?

- How we use our favorite **verbs**?
- What stories do we **tell**?

Method 3: Do you Speak Domain Language?

CRUD

CREATE

READ

UPDATE

DELETE

“You can’t tell a bedtime story with only the verbs create, read, update and delete” (Golo Roden)



Roden, G. (2022a)

Once upon a time, there was a king and queen who wished themselves a daughter. Their wish came true and a princess was created. One day, that princess updated her location to the big dark forest and retrieved there is a big grey wolf. In an assault the big grey wolf updated the isDeleted flag of the princess to true. King and queen updated their profileStatus to sad. The king also updated the hunters location to the big dark forest and the task to delete the big grey wolf. The brave hunter deleted the big grey wolf and updated the isDeleted flag of the princess to false.

And if they haven't been deleted, they lived happily ever after...

Method 3: Do you Speak Domain Language?

REST

POST

GET

PUT/PATCH

DELETE

Method 3: Do you Speak Domain Language?

Frontend

click

mousedown

touchStart

...

Developers: Our verbs...

CREATE

READ

UPDATE

DELETE

POST

GET

PUT/PATCH

DELETE

click

mouseDown

touchStart

...

→ Technical Language

Which feature is this?

click → POST → CREATE

The Trap we Built...

- Trap:
 - The sound we onCreate
 - The imposters we onAccept
 - The technical language we onUse

- Discussion:
 - Shall sound more “natural”
 - Faster
 - Focus the uninteresting parts
 - Avoids thinking
 - We think in the grammar of the programming language
 - (Sometimes) prevent us from thinking in facts, questions and actions



Part III

How to communicate clearly :)

1. Use English
2. Translate from Technical to Domain Language
3. Becoming a Product-Minded Software Engineer

Use English: Asking Questions vs. Checking Facts



```
if (hasPotatoes) {  
}
```



```
if (hasPotatoes()) {  
}
```



```
if (potatoesAvailable) {  
}
```

Use English: Computed Facts



```
class Supermarket {  
    get hasPotatoes() {  
        return this.potatoes.length > 0;  
    }  
}
```



```
class Supermarket {  
    get potatoesAvailable() {  
        return this.potatoes.length > 0;  
    }  
}
```

Use English: **Ask** your Questions

```
{{! with octane }}  
{{#if (can "buy potatoes")}}  
    ... something about potatoes  
{{/if}}
```

```
// with polaris  
import { canBuyPotatoes } 'supermarket-logic';  
  
<template>  
    {{#if (canBuyPotatoes)}}  
        ... something about potatoes  
    {{/if}}  
</template>
```

Use English: **Active Verbs** for Actions



```
class Supermarket {  
    get onProceedPayment() {  
        return this.potatoes.length > 0;  
    }  
}
```



```
class Supermarket {  
    get pay() {  
        return this.potatoes.length > 0;  
    }  
}
```

Translate from Technical to Domain Language

CRUD

- Repository
- Event Sourcing

REST

- only GET and POST
- GraphQL
- Sockets

DOM Events

- ?

Translate from Technical to Domain Language


```
{!! components/button.hbs !!}
<button type="button" {{on "click" @push}}>
  {{yield}}
</button>
```



Technical Name

Domain Name

```
<Mailbox>
  <Mail {{swipe "left" this.delete}}></Mail>
</Mailbox>
```



Translate from Technical to Domain Language

As a...
 User
I want to...
 onClick a button
So that...
 I can onClick



As a...
 User
I want to...
 push a button
So that...
 I can turn on the lights

```
<Button  
  @onClick={{this.onClick}}  
>
```

```
<Button  
  @push={{this.turnOnTheLights}}  
>
```


Technical Proxy Methods

```
<form {{on "submit" this.submit}}>
  {{! some fields }}
</form>
```

```
export class NounComponent extends Component {
  @action
  submit(event: SubmitEvent) {
    // collect data
    const data = new FormData(event.currentTarget);

    // pass this off to business logic
    this.pay(data);
  }
}
```

Becoming a Product-Minded Software Engineer

✗ DON'T

- Assimilate with technical language
- Use dev-lish
- Name properties, getters, methods, etc.

✓ DO

- Adapt domain language
- Use english
- Practice naming questions, actions and facts (consciously)
- **Revise prose** (treat your code like a blog article)

- Extract Business Logic from Components
 - Components glue UI and business logic together
 - Unit Test your business logic
 - Write business logic once, connect it from multiple components

Becoming a Product-Minded Software Engineer

- Improve your Naming Skills
- Make an inventory of your ubiquitous language
 - **Aggregates, Entities, Value Objects**
 - **Actions/Commands, Questions and Facts**
 - In collaboration with designers (eg. OOUX)
 - In collaboration with domain experts (eg. Event storming)
 - Practice Domain-driven Design
 - => The vocabulary of your team
- Culture your Domain
 - Better communication with designer, product and qa departments
 - Validate product features and operations

References

- Gossmann, T. (2021). *The Hidden Skill and Art of Naming Things*.
<https://gos.si/blog/the-hidden-skill-and-art-of-naming-things/>.
- Orosz, G. (2019). *The Product-Minded Software Engineer*.
<https://blog.pragmaticengineer.com/the-product-minded-engineer>.
- Pavlutin, D. (2019). *Coding like Shakespeare: Practical Function Naming Conventions*.
<https://dmitripavlutin.com/coding-like-shakespeare-practical-function-naming-conventions/>.
- Roden, G. (2022a). *CRUD? Bloß nicht! // deutsch*.
<https://www.youtube.com/watch?v=MoWynuslbBY>.
- Roden, G. (2022b). *HTTP-Statuscodes: Alle benutzen sie falsch?! // deutsch*.
https://www.youtube.com/watch?v=2ZOFCI3E-_c
- Yegge, S. (2006). *Execution in the Kingdom of Nouns*.
<https://steve-yegge.blogspot.com/2006/03/execution-in-kingdom-of-nouns.html>.

Thank you!