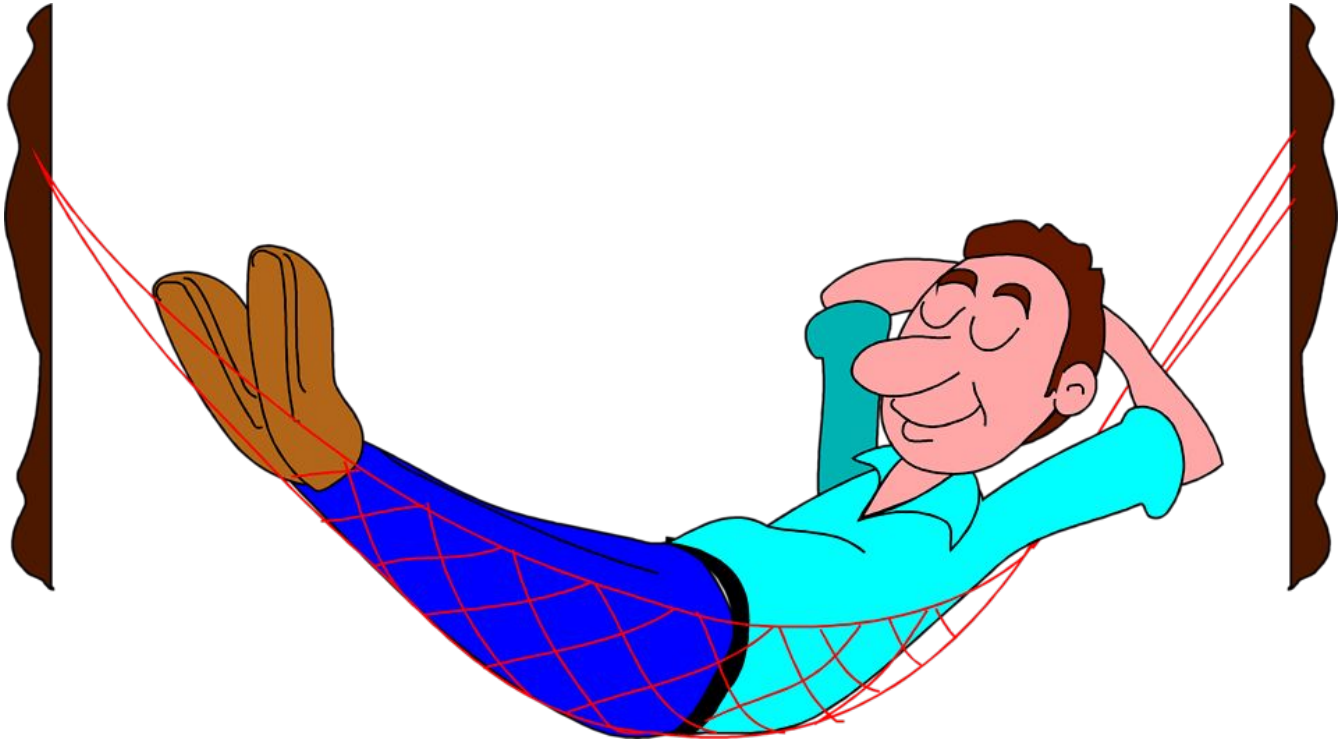
How to do architecture

tomas.kulich@vacuumlabs.com

What are we talking about



It's all about the economics

- if you take everything into account

- debug

- enhance

- introducing new people

- introducing existing people

- what will public think

- aesthetics

Don't think dogmatic

...use pros-cons lists instead

Ego is the enemy!

Can I just "like" some approaches more?

TLDR: No.

Long answer: yes, but..

Example1: Template engine

```
<div>
    <h1><a href="/">Django Boys Blog</a></h1>
</div>

{% for post in posts %}
    <div>
        <p>published: {{ post.published_date }}</p>
        <h1><a href="">{{ post.title }}</a></h1>
        <p>{{ post.text|linebreaksbr }}</p>
    </div>
{% endfor %}
```

```
`<div>
    <h1><a href="/">Django Boys Blog</a></h1>
</div>


${posts.map((post) => `
    <div>
        <p>published: ${post.published_date}</p>
        <h1><a href="">${post.title}</a></h1>
        <p>${post.text || linebreaksbr}</p>
    </div>`).join('')
}`
```

Example2: Databases

```
db.inventory.find( { status: "A", qty: { $lt: 30 } }
```

versus:

```
SELECT * FROM inventory WHERE status = "A" AND qty < 30
```

Easy vs simple

Easy: quick to use

Simple: simple to think about

Must see lecture on the topic: https://www.infoq.com/presentations/Simple-Made-Easy

Some complex stuff:

- implicit behavior (triggered by just naming stuff)

- pre/post something hooks, events

- ORMs

Databases made easy

```
db.inventory.find( { status: "A", qty: { $lt: 30 } }
```

versus:

```
SELECT * FROM inventory WHERE status = "A" AND qty < 30
```

versus:

```
db('inventory').where('status', 'A').whereLess('qty', 30)
```

```
arr.filter(x => x % 2 == 0).map(x => x/2).sum()
```

```
sum(map(x => x/2, filter(x => x % 2 == 0, arr)))
```

```
arr.filter(x => x % 2 == 0).map(x => x/2).sum()
```

```
sum(map(x => x/2, filter(x => x % 2 == 0, arr)))
```

```
thread(arr,
  [filter, x => x%2 == 0, $],
  [map, x => x/2, $],
  [sum, $])
```

```java
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class CalculateCircleAreaExample {

    public static void main(String[] args) {

        int radius = 0;
        System.out.println("Please enter radius of a circle");

        try
        { ...
```

How to build abstractions

- Do I need it at all?
  - don't generalize stuff you're going to need once
  - premature generalization antipattern
  - Interface implemented once
  - HOF called once
  - Protocol realized once

- Do I have the knowledge to do it now?
  - There is milion way how to build abstraction, have you considered more than one?

- Baklava is a good pastry, bad code
- Leaking abstractions

- What should be in the scope? **What shouldn't?**
- If you aim for everything, you end up with nothing
- NO is valid design decision!

- Java typesystem sucks
  - so does Dart's!
- JS objects cannot override hash and equality
  - same for Go

Decisions you may be ashamed of

- global variable can be a good idea
- copy paste can be a good idea
- shout on error can be a good idea
- monorepo very often is a good idea
- **"We'll need this later" is a myth,** "We'll rewrite this later" is a valid design decision
- "Portability is for people who cannot write new programs" --LT--

Wisdom of Python (Zen)

- Explicit is better than implicit
- Simple is better than complex
- Namespaces are for preventing name collisions not for creating taxonomies
- Readability counts
- There should be one-- and preferably only one --obvious way to do it

Ask Me Anything