

# Project Development Guide

[fivos.doganis@gmail.com](mailto:fivos.doganis@gmail.com)



KISS



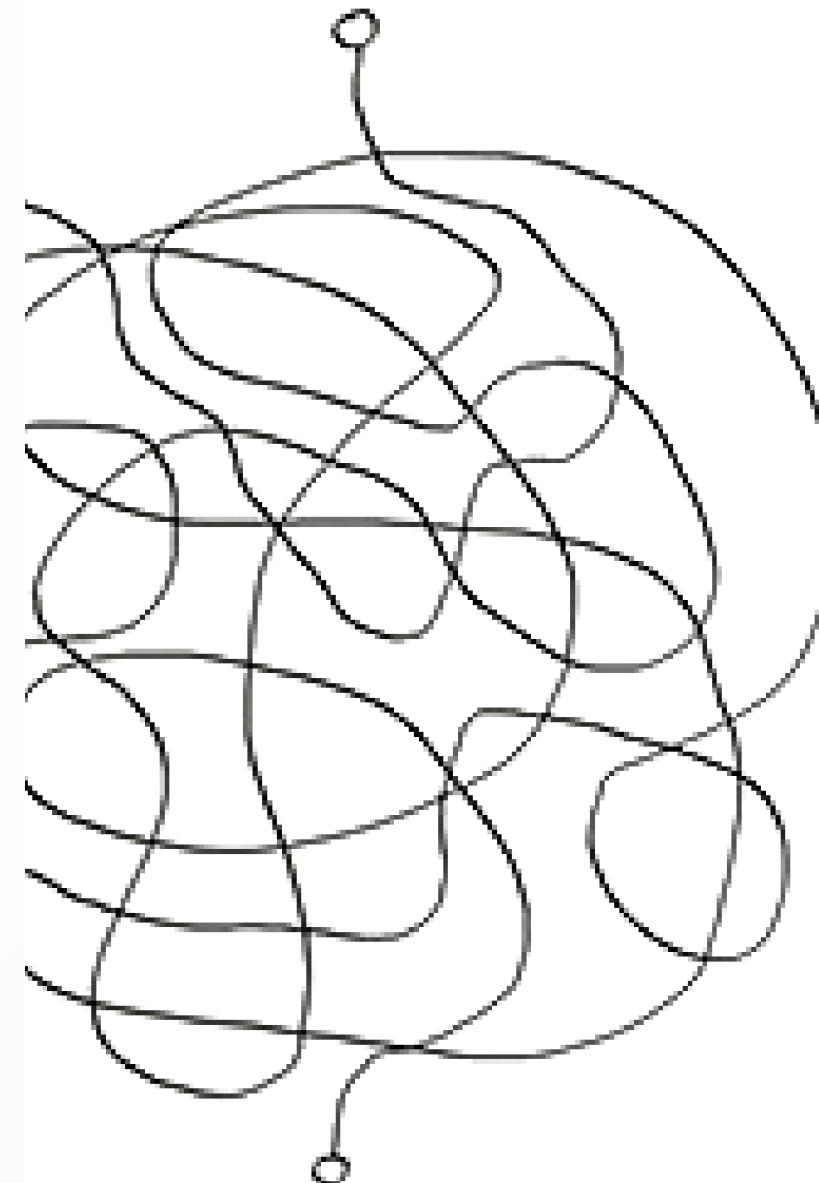
**K**eep

**I**t

**S**imple

**S**tupid

**!= Easy**



**"Simplify, Simplify, Simplify"**

*Steve Jobs*

YAGNI



You

Ain't

Gonna

Need

It

Code for now,  
not for the future.

Refactor often.

ROADS?  
WHERE WE'RE GOING,  
WE DON'T NEED ROADS.

D.R. EMMETT BROWN - BACK TO THE FUTURE

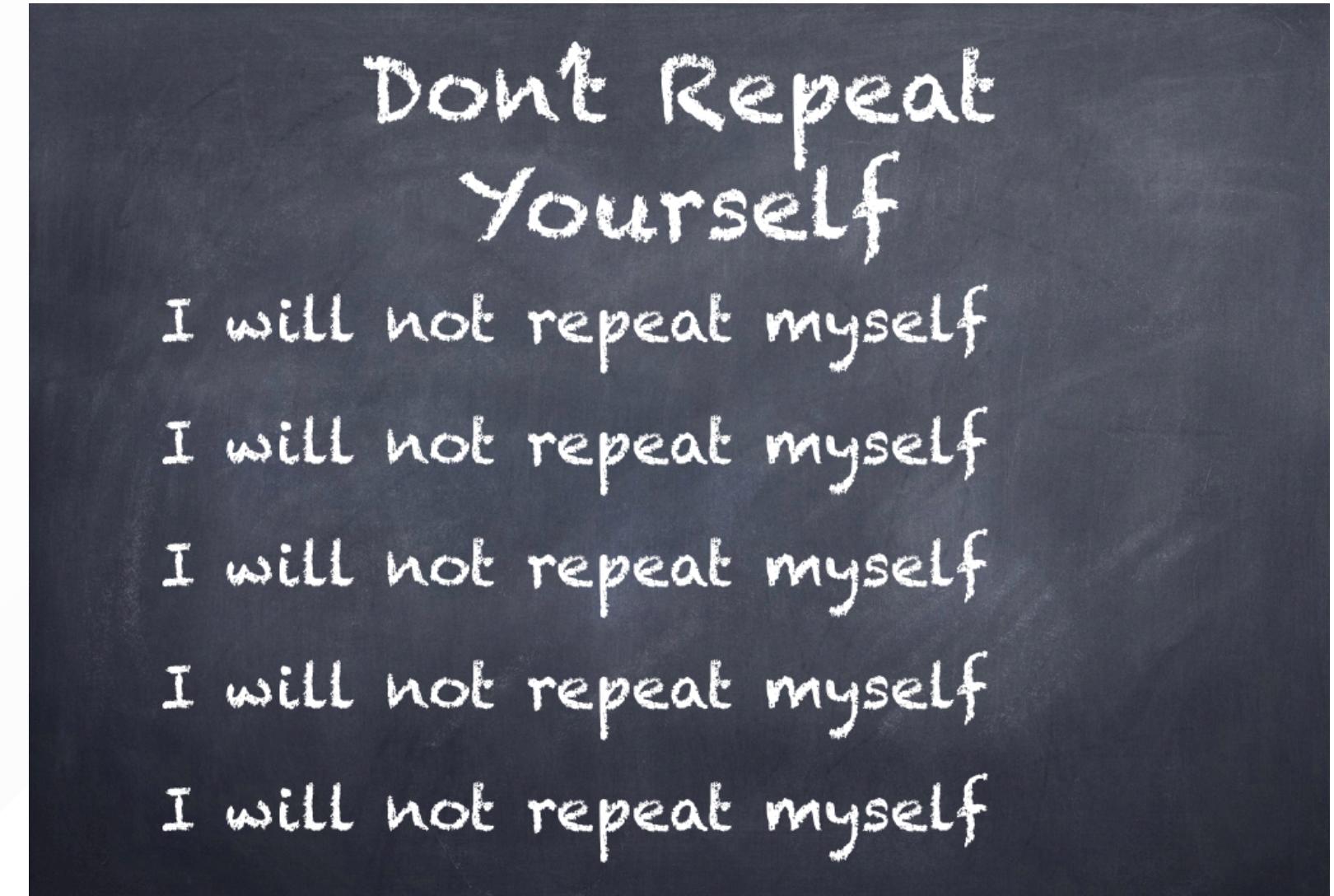


DRY



**D**on't  
**R**epeat  
**Y**ourself

!= WET :  
Write  
Everything  
Twice



# No Copy Paste!

**Copy Paste == Code Debt**

**Copy Paste == Refactoring Opportunity**

# Copy Paste

Stop!

Think

Refactor Now!

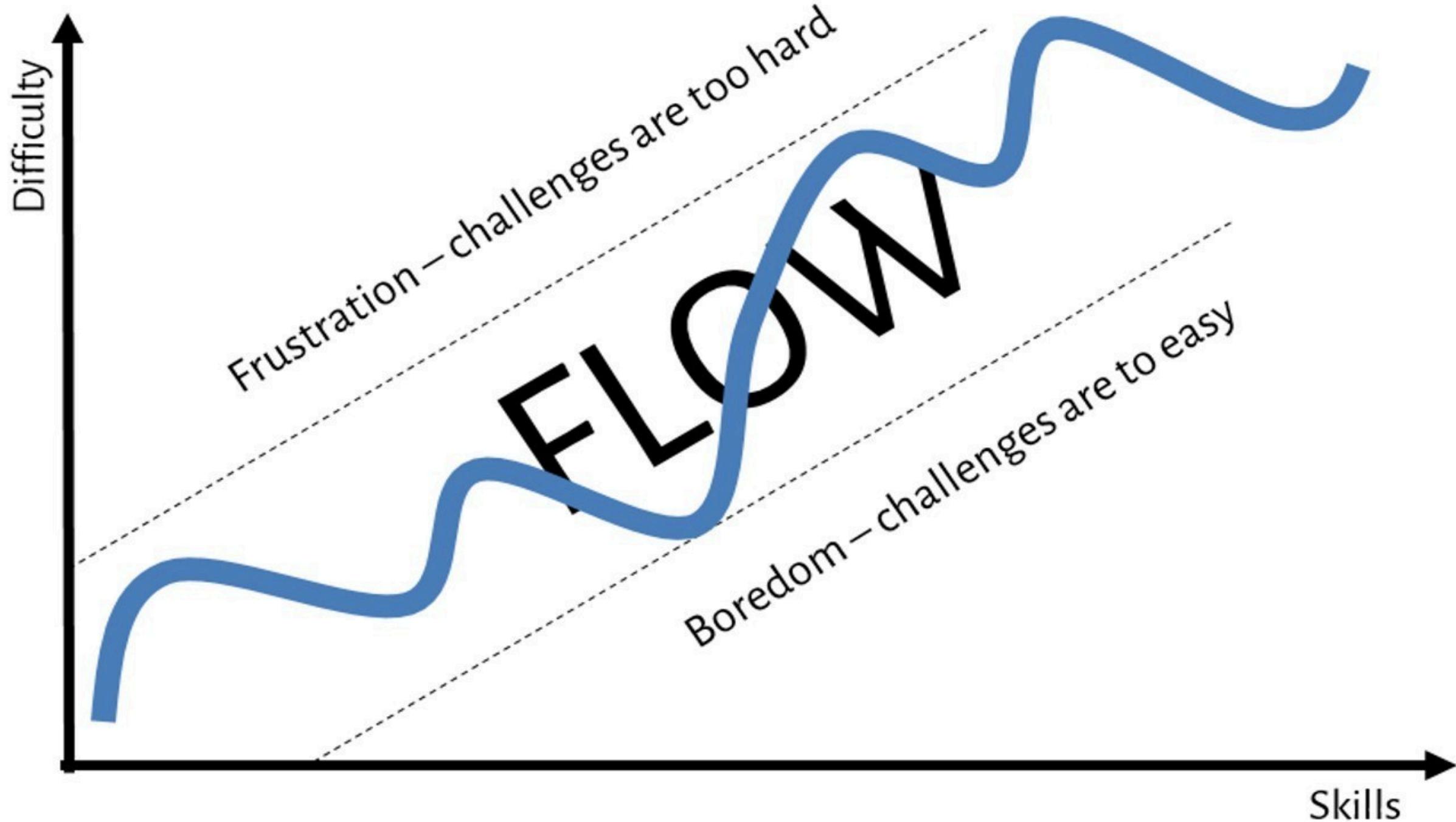


# Flow Theory

**The art of staying in the zone**

by Mihály Csíkszentmihályi







# Stuck?

## Fight procrastination



TODO ;)

# Fighting procrastination

- **OK to fail**
  - ~~"Die and retry"~~ ➔ Code and refactor
- Decompose into **micro actions**
  - Github issues + tags ("easy", "good first issue", etc.)
- Start just for **5 minutes**
  - identify easy issue
  - fix it to get the ball rolling
- Adjust skills and difficulty to stay in the zone

# Iterative Design

**MVP: Minimum Viable Product**



**Not like this....**



1



2



3



4



1



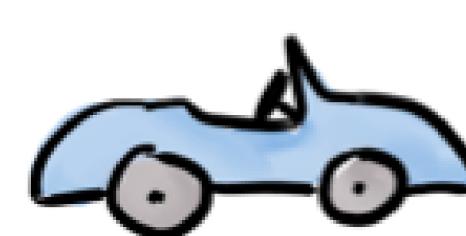
2



3



4



5

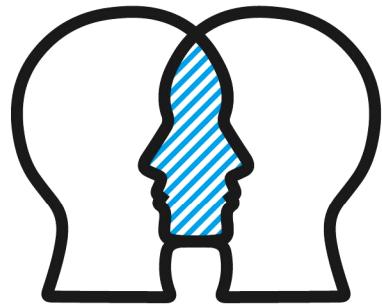
**Like this!**

# Design Thinking

## A Non-Linear Process



# Design Thinking



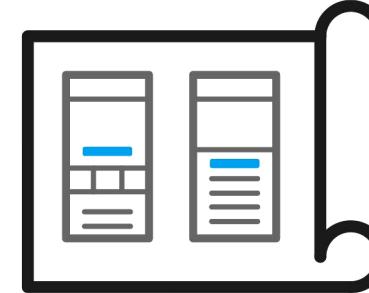
Empathize



Define



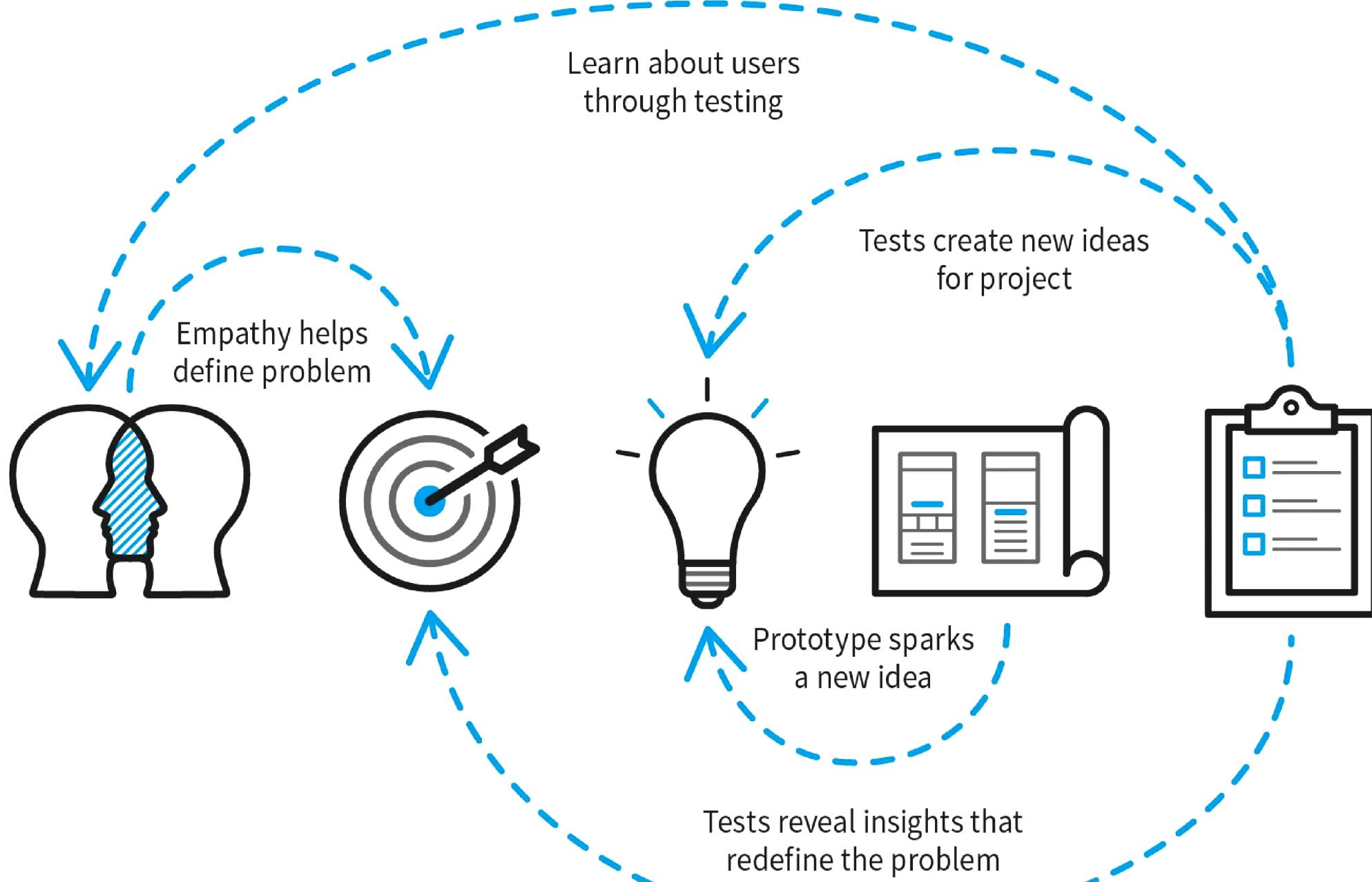
Ideate



Prototype



Test



# UX Basics

What makes a good User Experience?



# Use **objective** criteria

SUBJECTIVE	OBJECTIVE
<del>INTUITIVE</del>	EFFICIENT: FAST, ERROR PROOF
<del>NATURAL</del>	SIMPLE
<del>REALISTIC</del>	PRECISE

# Affordance

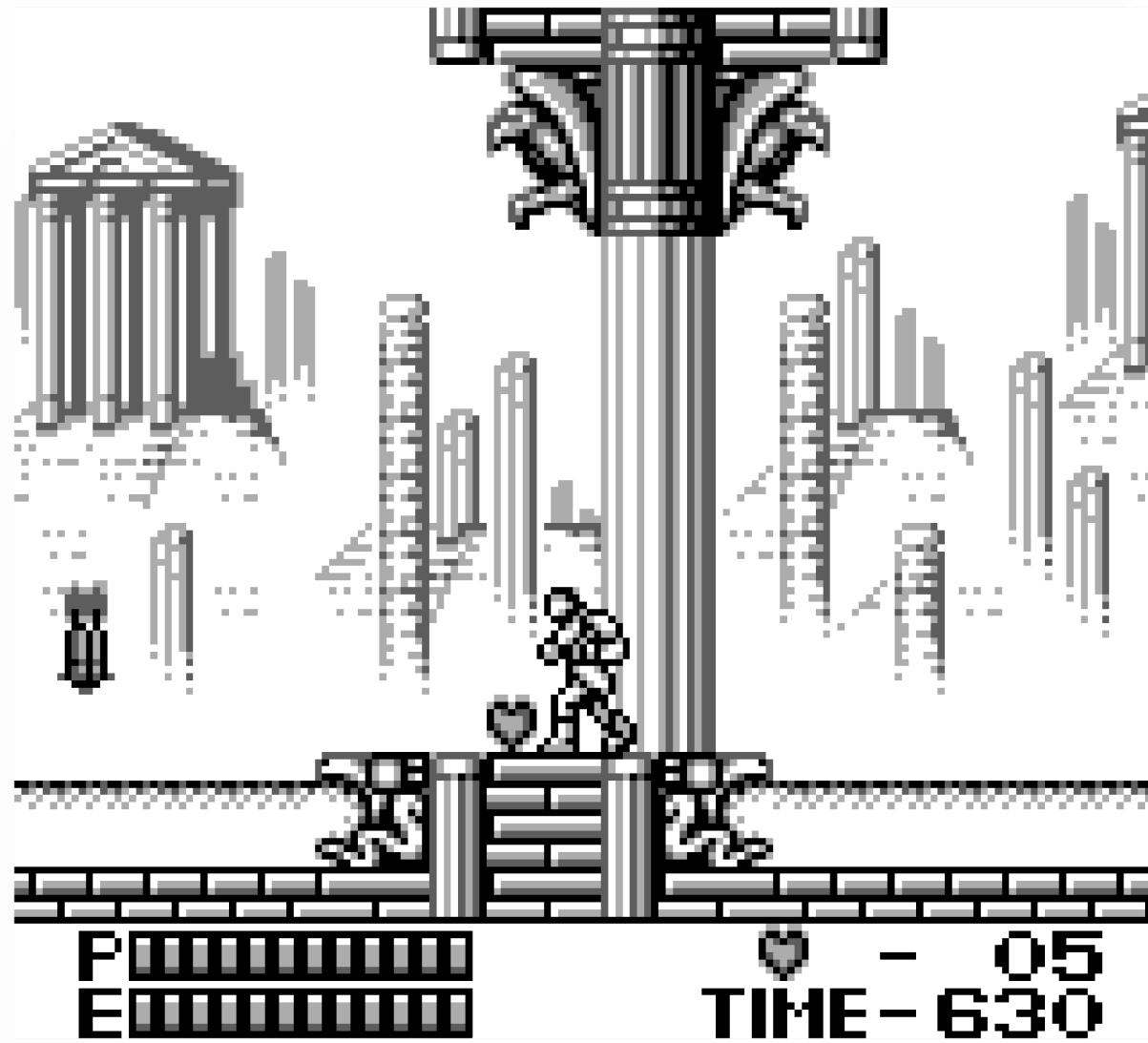
Form → function

- button 🎹
- door knob 🚪
- hammer 🔨

## Discoverability

- user guesses right





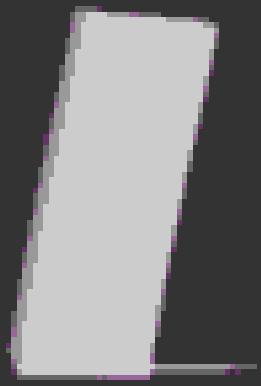
Castlevania ❤ item : **BAD** affordance!

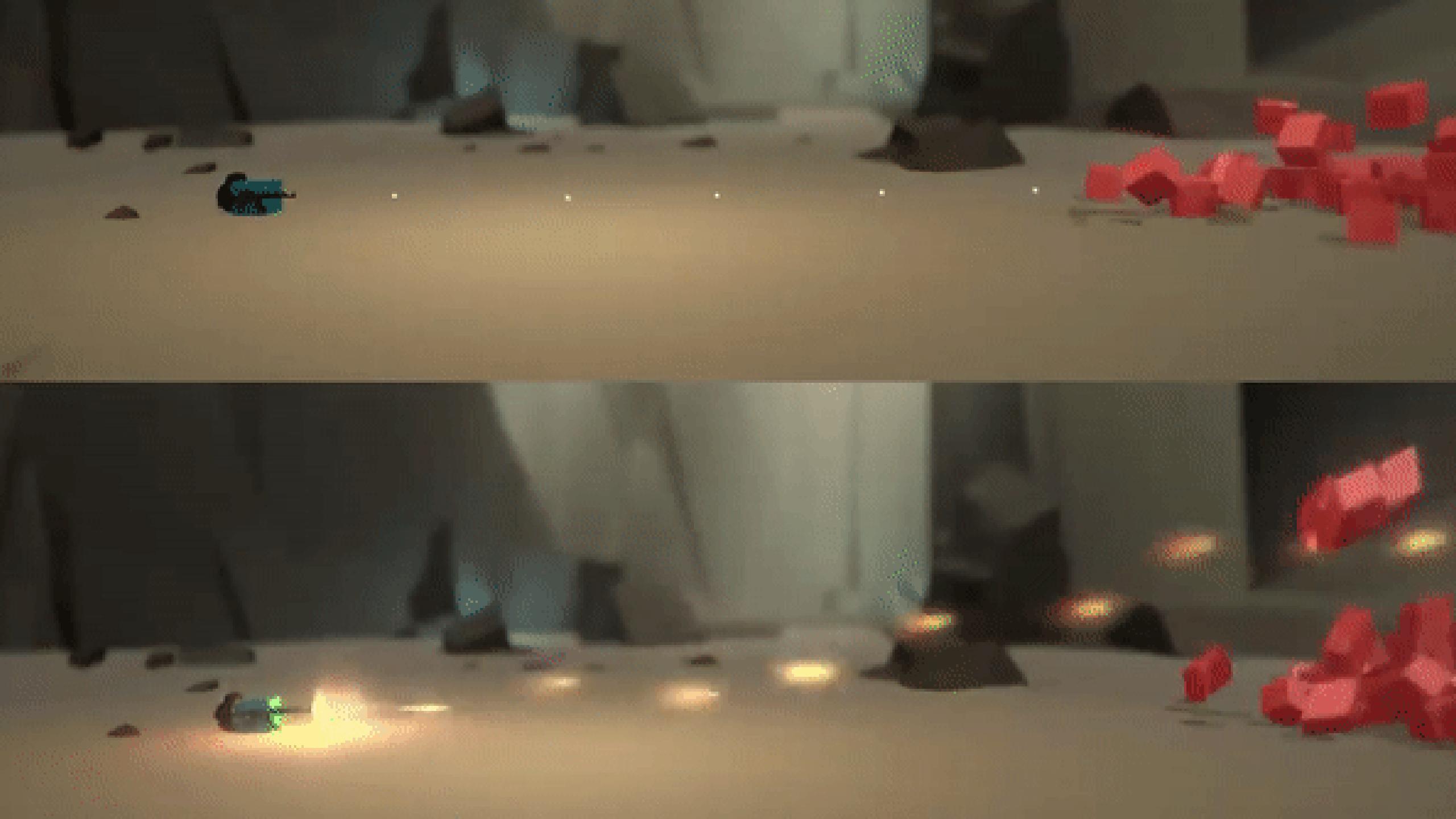
# Juice

## More than eye-candy

- action continuity
- smoother experience (TWEEN / lerp)
  - Robert Penner's equations
- user satisfaction 😍
  - reward mechanism
- ⚠️ avoid bad juice
  - distracting, unexpected







# The laws of UX

## Fitts's law

### **Fitts' Law**

#### **In plain words**

Big and near objects  
are easy to click.

Small and far objects  
are hard to click.



# Find measurable KPIs\*

## Examples

- **How long** does the user take to reach the target?
- **How many** mistakes are made before succeeding at a task?
- **How accurate** is the result vs a reference?

# Architecture Basics



# Design Patterns

- Builder, Observer, State can be interesting
- ⚠️ avoid Singletons!
- **adapt** the pattern to your code, don't use as-is!

# Game Programming Patterns ➡

- [Game Loop](#)
- [Object Pool](#)

# AntiPatterns

**Noooo, you need to use design patterns  
to make the code better. OOP helps us.  
Just listen to Uncle Bob**

**OOP makes  
everything  
unnecessarily  
convoluted**



**OOP makes  
everything  
unnecessarily  
convoluted**

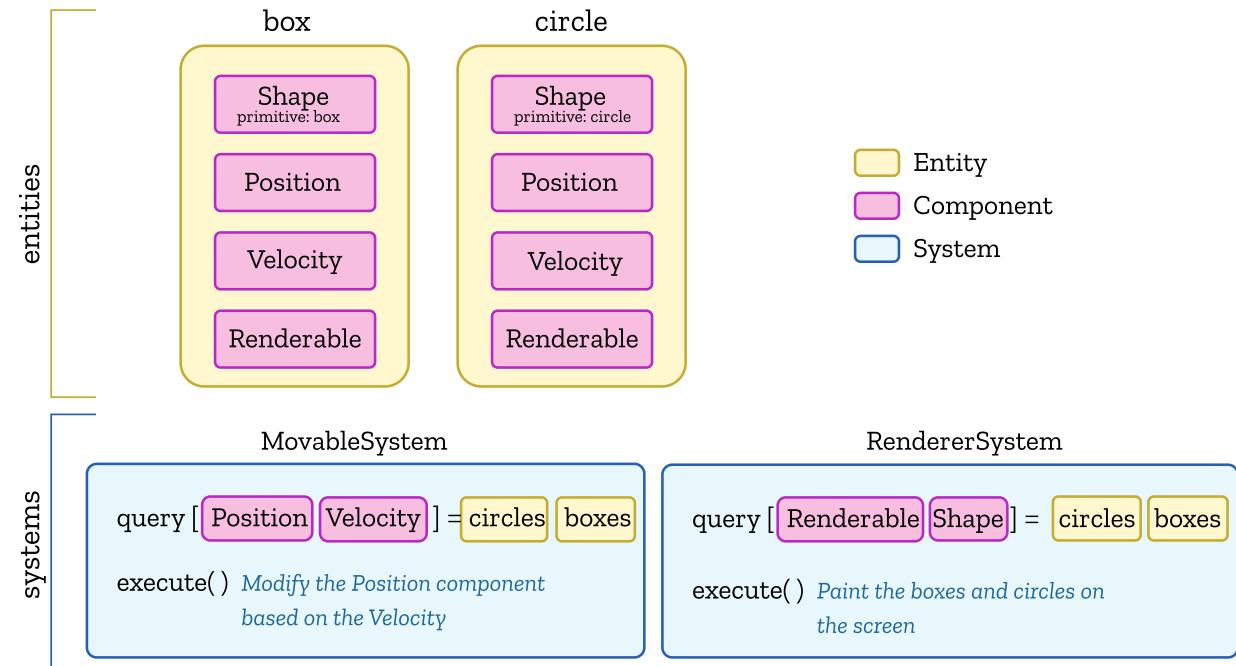
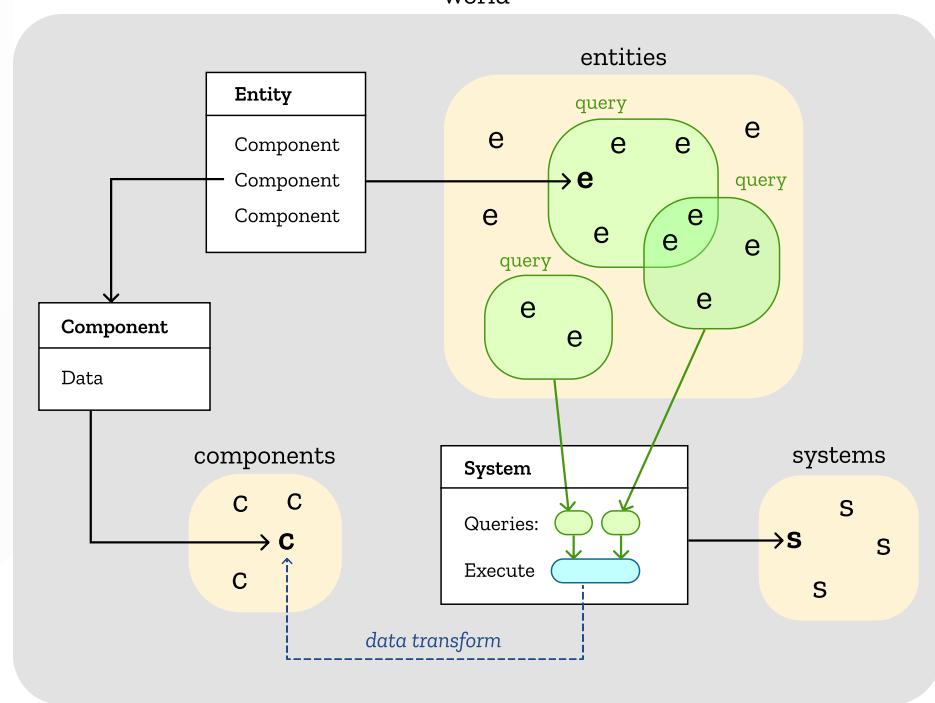


# ECS

## Entity Component System

## Articles

- [Mozilla](#)
- [Medium](#)
  - [ECS in 99 lines of code](#)
- [ECS @ Apple \(Video 25'\)](#) ★
- used in [Overwatch](#)



# **Final tips**

**To keep your mind and code sane**



# Self documenting code

- document the "why?"
  - at high-level
  - if needed
- avoid documentation altogether
  - write tests, examples, tutorials instead
  - write **code as documentation**



# Code as documentation

- **BAD** 

```
const float a = 9.81; //gravitational force
float b = 5; //time in seconds
float c = (1/2)*a*(b^2) //multiply the time and gravity together to get displacement.
```

- **GOOD** 

```
float computeDisplacement(float timeInSeconds) {
    const float gravitationalForce = 9.81;
    float displacement = (1 / 2) * gravitationalForce * (timeInSeconds ^ 2);
    return displacement;
}
```



## Tweet



**Leon Bambrick**  
@secretGeek

...

There are 2 hard problems in computer science: cache invalidation, naming things, and off-by-1 errors.

7:50 PM · Jan 1, 2010

<https://martinfowler.com/bliki/TwoHardThings.html>

# Naming Conventions

- "What is this object? What does it do?"
- Object, variable → noun
  - `user`, `accountNumber`, `customerEmail`
- Function, method → verb
  - `user.login()`, `shutDown()`
- Boolean → adjective
  - `allowed`, `disabled`
  - `user.active()` 
  - `user.isActive()` 

# Naming tips

- convey **intention**

```
var d // elapsed time in days ✗ vs var elapsedTimeInDays ✓
```

- name **arguments** too!

```
void copyChars(char a1[], char a2[]) ✗
```

```
void copyChars(char source[], char destination[]) ✓
```

- **pronunciation** (avoid abbreviations)

```
genymdms ✗ vs generationTimestamp ✓
```

- **no magic numbers** (keep your code searchable and maintainable)

```
s / 5 ✗ vs task / WORK_DAYS_PER_WEEK ✓
```

- use **camelCase**

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

*Martin Fowler*

# The End!

