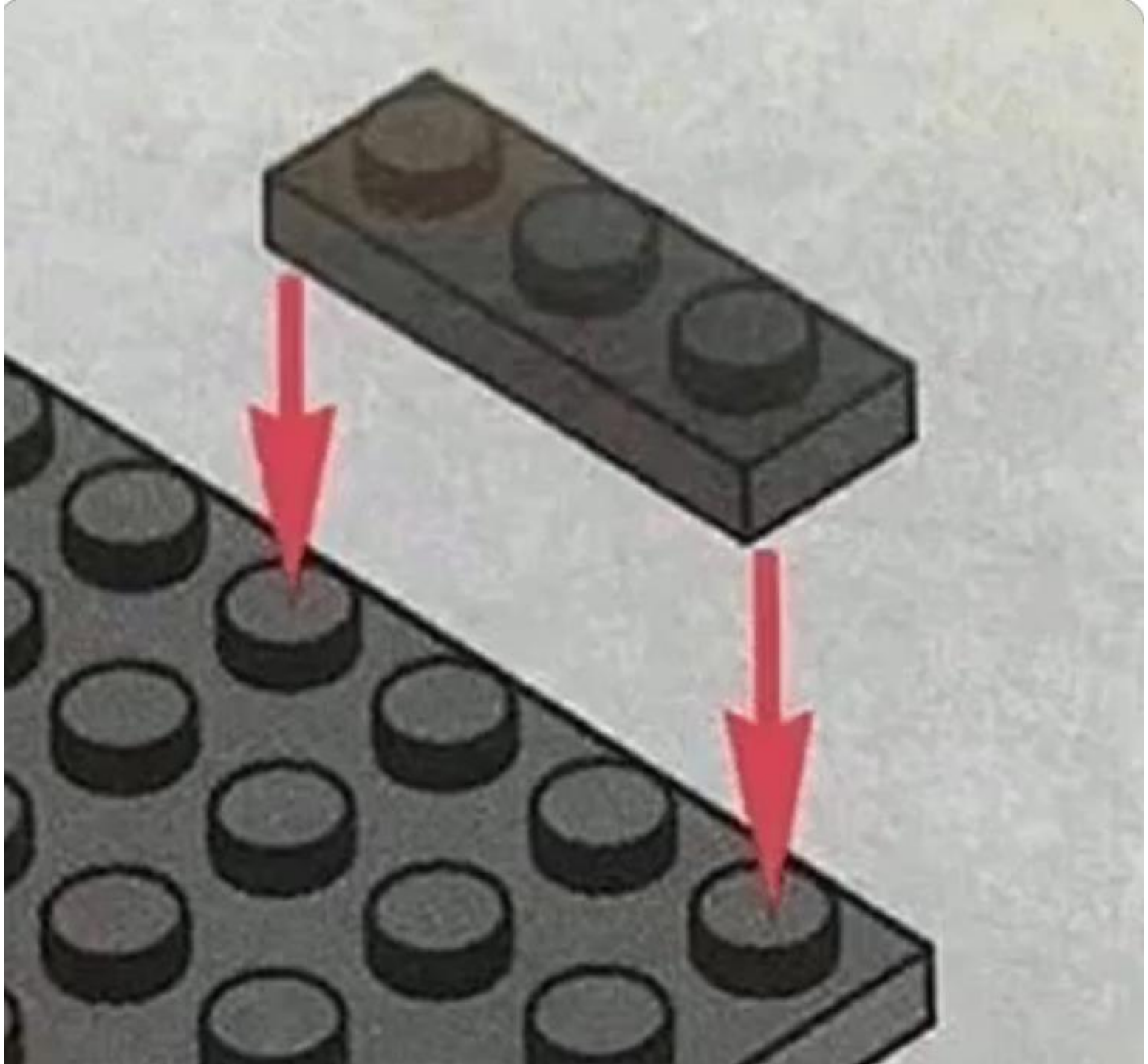
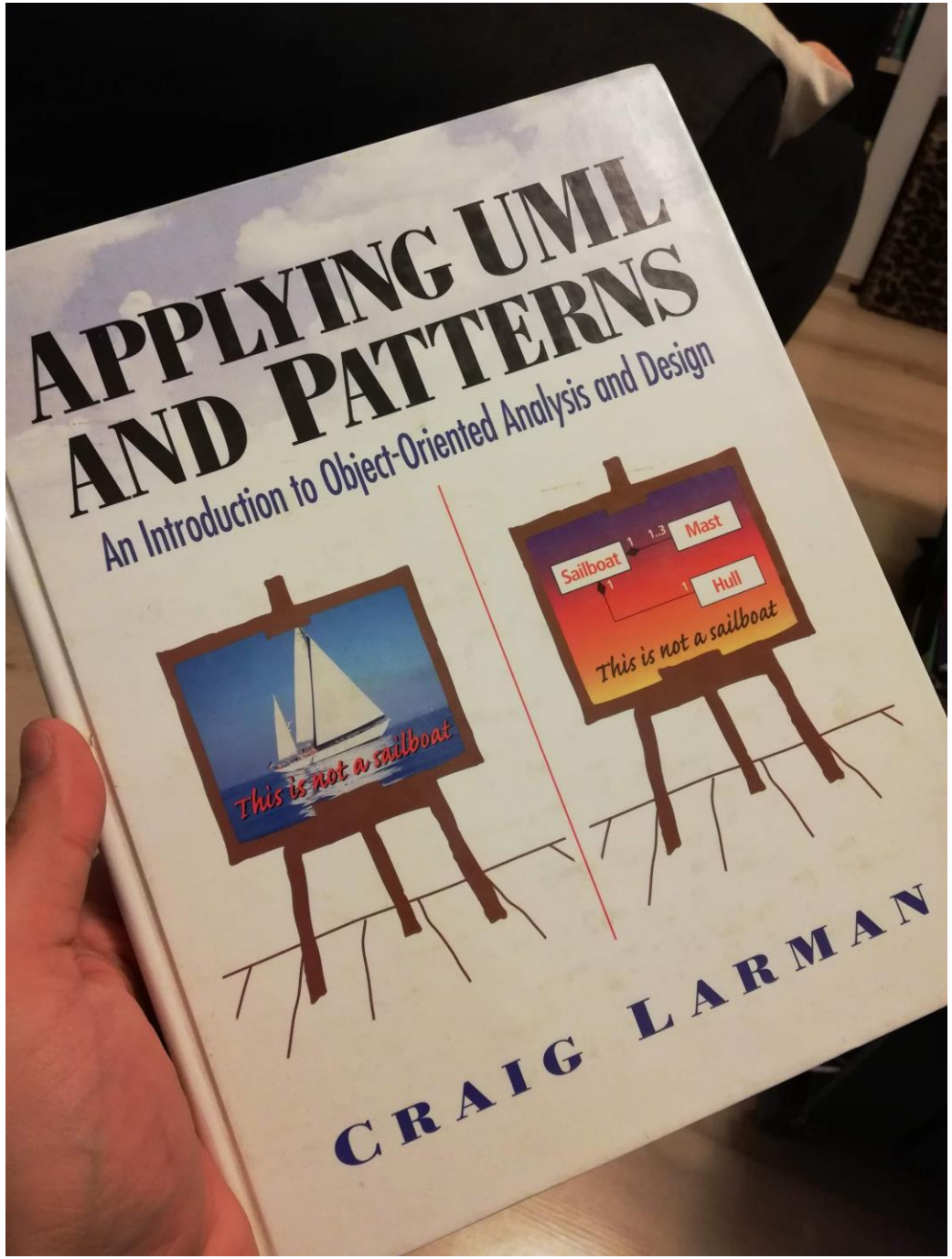
intive

Nie tylko SOLID. GRASP - jeszcze jeden sposób na kod wysokiej jakości.

Karol Kreft
Senior Software Engineer at intive


eventory
for better events · for your events




APPLYING UML
AND PATTERNS
An Introduction to Object-Oriented Analysis and Design
This is not a sailboat
Sailboat
Mast
Hull
This is not a sailboat
CRAIG LARMAN

Craig Larman

- www.craiglarman.com
- UML, OOP, Analysis & Design
- **GRASP**
- Agile and LeSS

The **GRASP** name was chosen to suggest the importance of **grasping** these principles to successfully design object-oriented software.

General
Responsibility
Assignment
Software
Patterns

General
Responsibility
Assignment
Software
Patterns

intive

Responsibilities

- do something
- know something

```php
final class Email {
    private $email;

    public function __construct(string $email) {
        $this->email = $email;
    }

    public function isValid(): bool {
      return true;
    }
}
```

```php
interface EmailValidator {
    public function validate(Email $email): Violations;
}

final class Email {
    private $email;

    public function __construct(string $email) {
        $this->email = $email;
    }
}
```

intive

GRASP

1. Information Expert
2. Creator
3. Controller
4. Low coupling
5. High cohesion
6. Indirection
7. Polymorphism
8. Pure Fabrication
9. Protected Variations

Information Expert

- Assign a responsibility to the information expert – the class that has the information necessary to fulfill the responsibility.

```php
final class Email {
    private $email;

    public function __construct(string $email) {
        $this->email = $email;
    }

    public function isValid(): bool {
      return true;
    }
}
```


LEGO
CREATOR
EXPERT
Ages/edades
12+
10249
Winter Toy Shop
898
pcs/pzs
Building Toy
Jouet de construction
Juguete para Construir
Light brick
batteries
included
Le magasin de jouets d'hiver
Juguetería Navideña
Brique lumière
Ladrillo luminoso

Creator

- Who should be responsible for creating a new instance of some class?
- Choose class B when
  - B aggregates A objects
  - B contains A objects
  - B closely uses A objects
  - B has the initializing data that are required to creating A

```php
class SmsSender implements NotificationSender {
  private $recipients;

  public function notifyAll(string $message) {
    foreach ($this->recipients as $recipient) {
      $this->sendSMS(new Sms($recipient, $message));
    }
  }

  private function sendSms(Sms $sms);
}
```

intive

Controller

- Who should be responsible for handling system event?

```
class ReadActorsApiController {
    public function getActors(): JsonReponse;
    public function getActorDetails(Request $request): JsonReponse;
}

class DatabaseCleaner {
    public function removeAccount();
    public function anonymiseUserData(UserId $userId);
}

class MessageHanlder {
    public function handle(MessageCommand $command));
}
```

Bloated Controller

- Controllers which handle too many system events leading to **low cohesion**. This can be avoided by addition of a few more controllers.
- Always remember about delegating responsibilities to other objects.

Low coupling

High cohesion


R

intive

Low Coupling

_Assign responsibilities so that coupling remains **low.**

High Cohesion

_Assign responsibilities so that cohesion remains **high.**

intive

Low Coupling

_How to support low dependency and increased reuse?

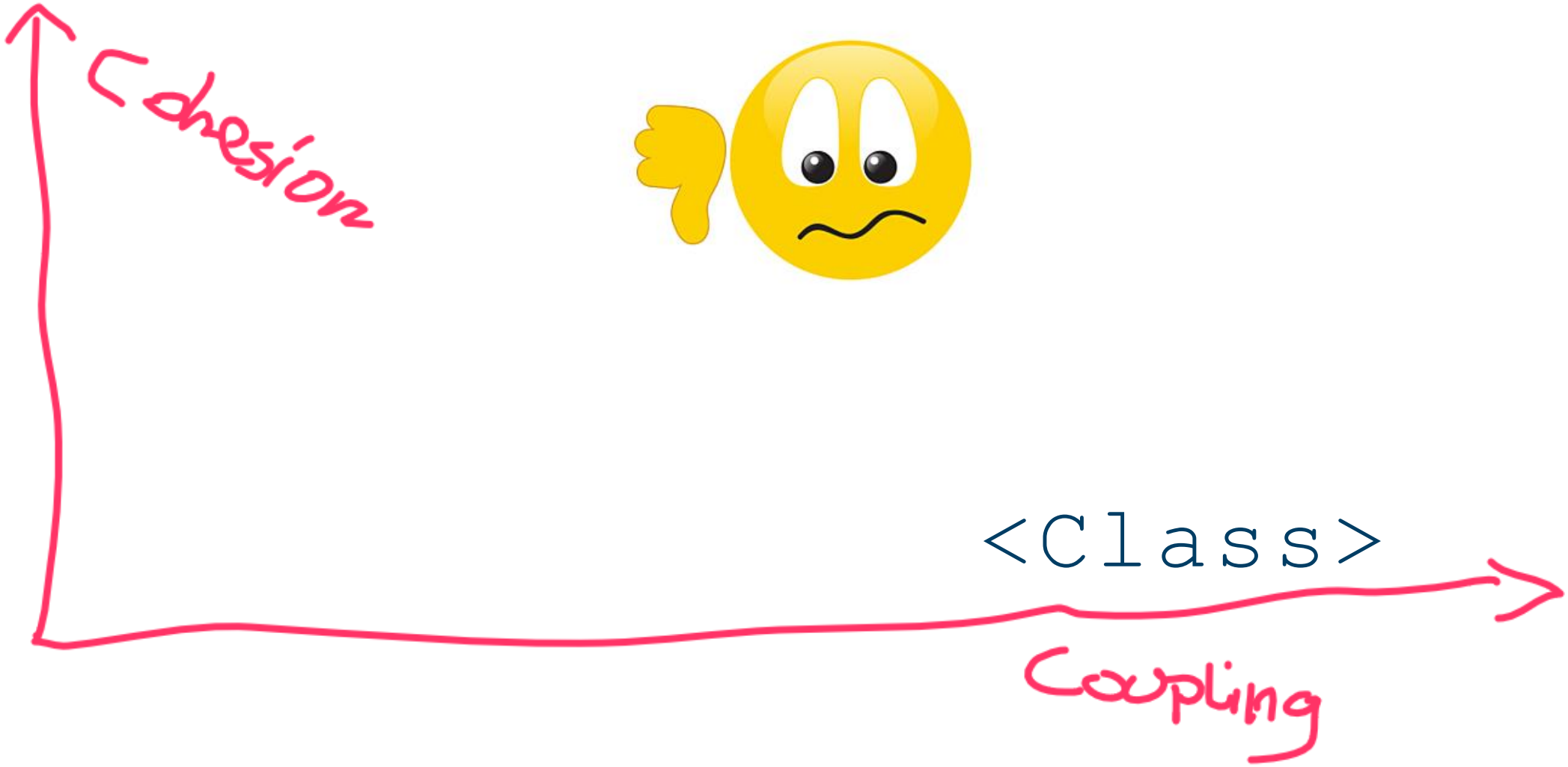
High Cohesion

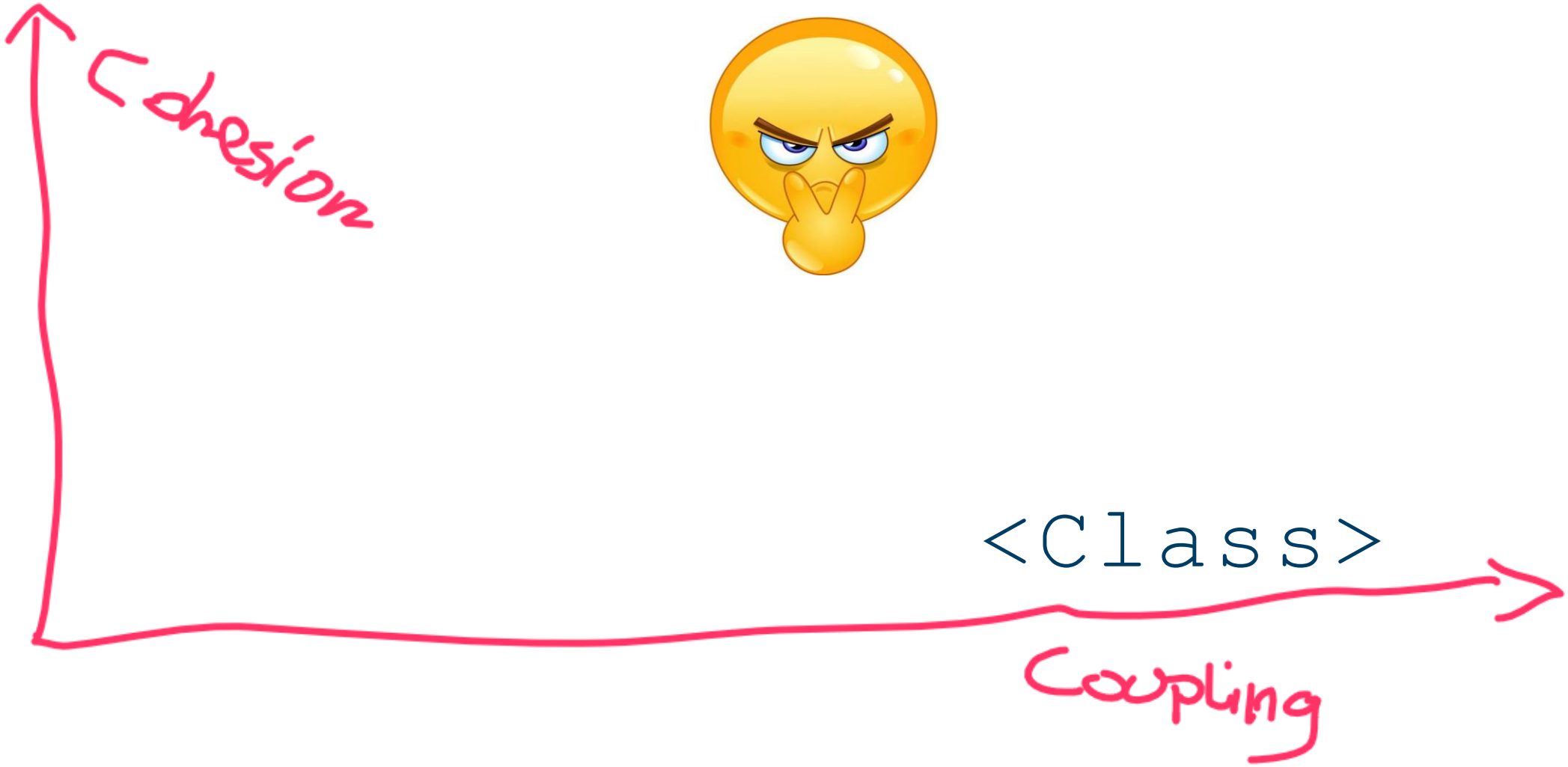
_How to keep complexity manageable?

intive

<Class>


Cohesion
<Class>
Coupling


Cohesion
<Class>
Coupling


Cohesion
<Class>
Coupling

High Coupling

- Changes in related classes force local changes
- Harder to understand in isolation
- Harder to reuse because its use requires the additional presence of the classes it dependent upon

High Coupling

- Changes in related classes force local changes
- Harder to understand in isolation
- Harder to reuse because its use requires the additional presence of the classes it dependent upon
- **Coupling may not be important if reuse is not a goal.**

Low Cohesion

- Hard to comprehend
- Hard to reuse
- Hard to maintain
- Delicate; constantly effected by change

Indirection

- Where to assign a responsibility to avoid direct coupling between two or more things?

Indirection

- Where to assign a responsibility to avoid direct coupling between two or more things?
- **Assign the responsibility to an intermediate object to mediate between other components or services to avoid direct coupling.**

Indirection

Fundamental theorem of software engineering

From Wikipedia, the free encyclopedia

The **fundamental theorem of software engineering** (**FTSE**) is a term originated by Andrew Koenig to describe a remark by Butler Lampson[1] attributed to the late David J. Wheeler:[2]

> We can solve any problem by introducing an extra level of indirection.

Indirection

(6) It is easier to move a problem around (for example, by moving the problem to a different part of the overall network architecture) than it is to solve it.

(6a) (corollary). It is always possible to add another level of indirection.

https://tools.ietf.org/html/rfc1925

Pure Fabrication

- Where to assign responsibility to not violate High Cohesion and Low Coupling?

Pure Fabrication

- Where to assign responsibility to not violate

  High Cohesion and Low Coupling?

- Ignore **Information Expert**

???

Pure Fabrication

- Where to assign responsibility to not violate

  High Cohesion and Low Coupling?

- Ignore **Information Expert**

DDD's Application layer!

Pure Fabrication

- Where to assign responsibility to not violate High Cohesion and Low Coupling?
- Ignore **Information Expert**
- **Many existing object-oriented design patterns are examples of Pure Fabrications: Adapter, Observer, Visitor**

Protected Variations

- How to design objects, subsystems, and systems so that the variations or instability in these elements does not have an undesirable impact on other elements?

Protected Variations

- How to design objects, subsystems, and systems so that the variations or instability in these elements does not have an undesirable impact on other elements?

- **Modules should be both open (for extension and adaptation) and closed (to avoid modification that affect clients).**

Protected Variations

- How to design objects, subsystems, and systems so that the variations or instability in these elements does not have an undesirable impact on other elements?

Open-Closed Principle

- **Modules should be both open (for extension and adaptation) and closed (to avoid modification that affect clients).**

Protected Variations

- How to design objects, subsystems, and systems so that the variations or instability in these elements does not have an undesirable impact on other elements?

- **Identify points of predicted variation and create a stable interface around them.**

intive

GRASP

1. **Information Expert**
2. **Creator**
3. **Controller**
4. **Low coupling**
5. **High cohesion**
6. **Indirection**
7. Polymorphism
8. **Pure Fabrication**
9. **Protected Variations**

intive

Information Expert

Creator

Controller

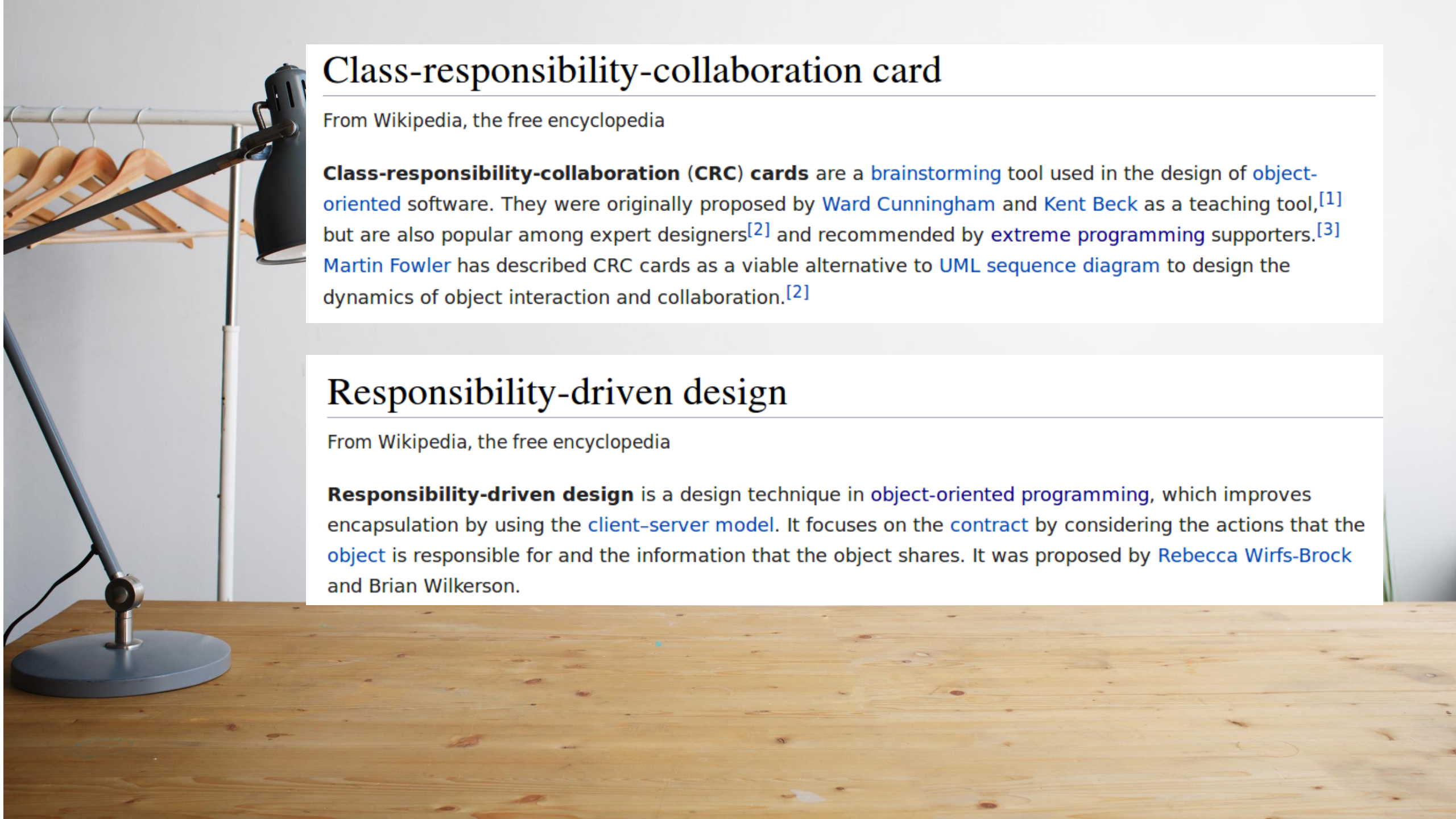
Low coupling
High cohesion

Indirection

Pure Fabrication

Protected Variations

Do we need object oriented design?



Class-responsibility-collaboration card

From Wikipedia, the free encyclopedia

Class-responsibility-collaboration (**CRC**) **cards** are a brainstorming tool used in the design of object-oriented software. They were originally proposed by Ward Cunningham and Kent Beck as a teaching tool,[1] but are also popular among expert designers[2] and recommended by extreme programming supporters.[3] Martin Fowler has described CRC cards as a viable alternative to UML sequence diagram to design the dynamics of object interaction and collaboration.[2]

Responsibility-driven design

From Wikipedia, the free encyclopedia

Responsibility-driven design is a design technique in object-oriented programming, which improves encapsulation by using the client–server model. It focuses on the contract by considering the actions that the object is responsible for and the information that the object shares. It was proposed by Rebecca Wirfs-Brock and Brian Wilkerson.

intive

Q&A?

@karol_kreft

Photos

- Photo by La-Rel Easter on Unsplash
- Photo by Feliphe Schiarolli on Unsplash
- Photo by Andrej Lišakov on Unsplash
- Photo by Rohit Choudhari on Unsplash
- Photo by Javier Allegue Barros on Unsplash
- Photo by Hanny Naibaho on Unsplash
- Photo by Annie Spratt on Unsplash
- Creator - https://zklockow.pl
- Batman and Robin photo: Episode: Batman Sets the Pace. Television Series: Batman (TV series) (1966-1968). The legal owner: WBTV.). (S1E26)