CUISINE & DEPENDANCES

ZABOU
JEAN-PIERRE BACRI
JEAN-PIERRE DARROUSSIN
SAM KARMANN
AGNES JAOUI

Une pièce de
AGNES JAOUI
et JEAN-PIERRE BACRI

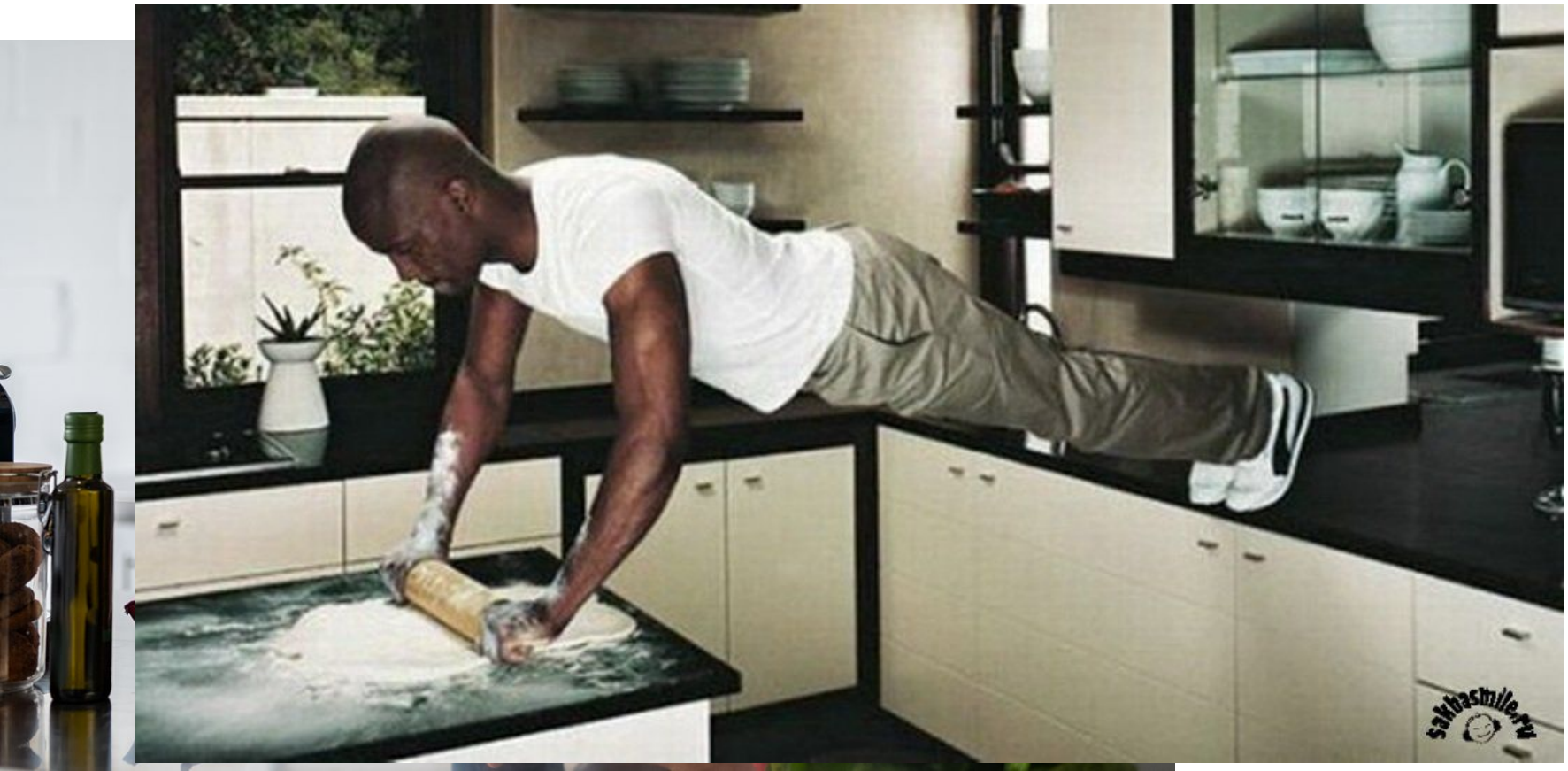
De quoi allons nous parler (peut être...)

- Pourquoi ce sujet?
- Détecter les dépendances
- Quels impacts?
- Dépendances : favorables ou nuisibles?
- Dépendance et addiction
- Dépendances: partout dans le code
- Comment se débarrasser des nuisibles?
- Comment vivre avec?
- Pourquoi devoir “tirer” des dépendances?
- Comment ne pas rechuter?

https://www.linkedin.com/in/guillaumese/
https://gitlab.com/GuillaumeToulouse https://github.com/guillaumeagile

Cuisine ?

Je cuisine, tu cuisines, nous cuisinons...



Apprentissage

Expérimentation

Savoir Faire

Matière premières

et bien sur.... outils !

des outils, encore des outils

Et à la fin...

Et surtout

Code & Dépendances

Synonymes

- Couplage
- Mélangé / Spaghetti
- Adhérence / Friction

Antonymes

- Découplage
- Isolation
- Orthogonalité

https://philippe.kruchten.com/2013/11/24/friction/

Couplage

Afférent

Afferent Coupling (Ca)

number of classes outside the package that depend upon the measured package u "how responsible am I? " (FAN-IN)

Efférent

Efferent Coupling (Ce)

number of classes outside the package that classes within the measured package depend on u "how dependent am I ? " (FAN-OUT)

Instability Factor (I)

Ca / (Ca + Ce)

0 = totally stable; 1 = totally unstable

“Mock setup exposes coupling, remember;
not cohesion or other design properties.”

Gary (again)

https://www.destroyallsoftware.com/blog/2014/test-isolation-is-about-avoiding-mocks

Meilir Page-Jones

Comparing Techniques by Means of Encapsulation and Connascence

Today the object-oriented approach to software development is at the height of fashion. As such, it threatens to replace the structured approach which was the staple development approach of the 1970s and 1980s.

Both approaches have their genesis in programming. Structured programming began in the late 1960s and is usually traced to workers such as Dijkstra [4]. Although object orientation is a river with many tributaries, the source of object-oriented programming is often traced—inter alia—to the late 1960s and the Simula language or to the Smalltalk work of Alan Kay and others at Xerox PARC, *circa* 1970 [5, 6].

Therefore, the structured approach rapidly grew into design and analysis techniques. Indeed, almost contemporaneously with the inception of structured programming, Larry Constantine and others were establishing the foundations of structured design [2]. Throughout the 1970s, workers such as Doug Ross [11] and Tom DeMarco [3] were developing a technique that came to be known as structured analysis.

Structured programming saw its initial rise in popularity in the early 1970s; structured design grew in popularity in the mid-1970s and structured analysis gained many adherents in the late 1970s. By contrast, object-oriented programming was a relative curiosity until the early 1980s. Object-oriented design and object-oriented analysis made their first significant appearances in the late 1980s [1].

During most of the two decades of the coexistence of the structured and the object-oriented approaches, there has been little interaction between researchers of the two approaches. Consequently, the two fields tended to develop different vocabularies and different foci of utilization. These apparent differences have tended to obscure some of the similarities between the structured approach and the object-oriented approach.

On the other hand, the two approaches are not—as some people would imply—simply isomorphs with disparate terminology. There are some deep structural distinctions between them.

In this article, I discuss the similarities and differences between the structured and object-oriented approaches to software design by means of two principles: encapsulation and connascence. I conclude by suggesting experiments involving connascence, the use of tools to mediate its effects and a set of steps necessary to define any new design paradigm.

Encapsulation and Connascence

Both the structured and object-oriented approach rely on encapsulation—the technique by which a set of software components is aggregated into a structure that can be treated as a unit from an external point of view.

Structured design encapsulates lines of code into procedural units that Constantine called *modules*. Examples in traditional languages include procedures, subroutines and functions. I will term this type of encapsulation level-1 or "L1" encapsulation as seen in Figure 1.

Object-oriented design encapsulates units called *classes*.[1] The structure of a class is more complex than that of the structured-design module, since a class is a template or descriptor representing a set of modules (more often called *methods*) packaged around an internal structure that will hold the state of each object belonging to the class. This structure is the set of *internal variables*. I will term this type of encapsulation "L2." As Figure 2 indicates, L2 encapsulation includes L1 within it, because the methods of a class are structurally similar to the modules of structured design. (Incidentally, L0 encapsulation would be *null encapsulation*, in which the highest-level construct would be the humble line of procedural code or elementary variable.)

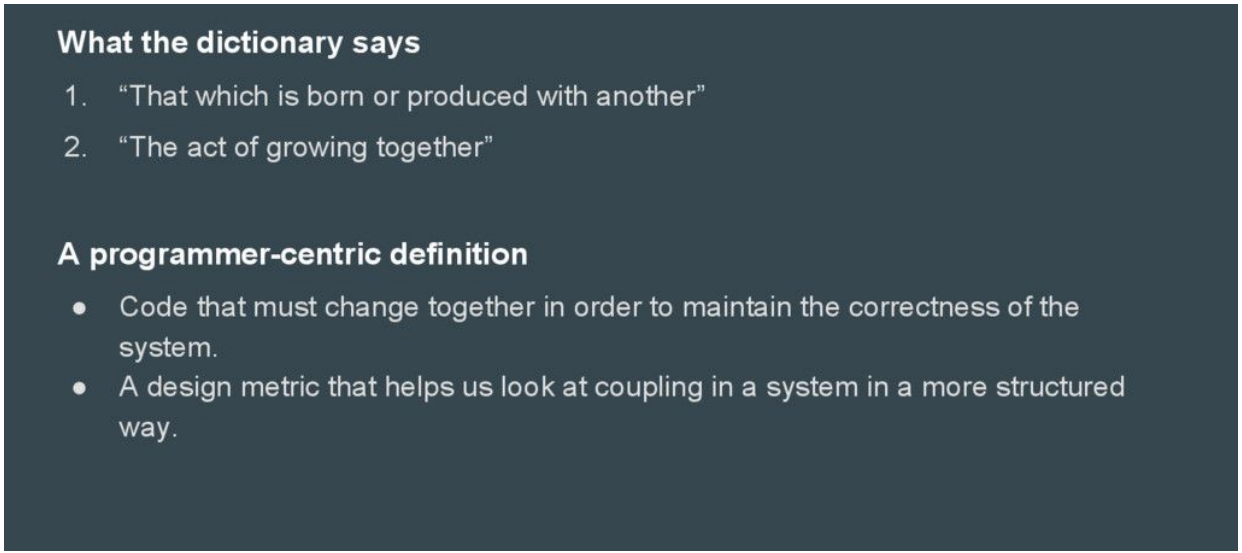
Structured design offers two criteria for evaluating the quality of encapsulation resulting from a given partitioning into modules. These criteria are: coupling, which measures the binding between code elements that are found in distinct modules; and cohesion, which measures the binding between code elements that are found in the same module. The ideal of structured design is to minimize the coupling of a system's design and to maxi-

[1]Objects are the structurally equivalent constructs that are spawned dynamically during system execution. In some implementations the class and the object are degenerately identical.

Définition

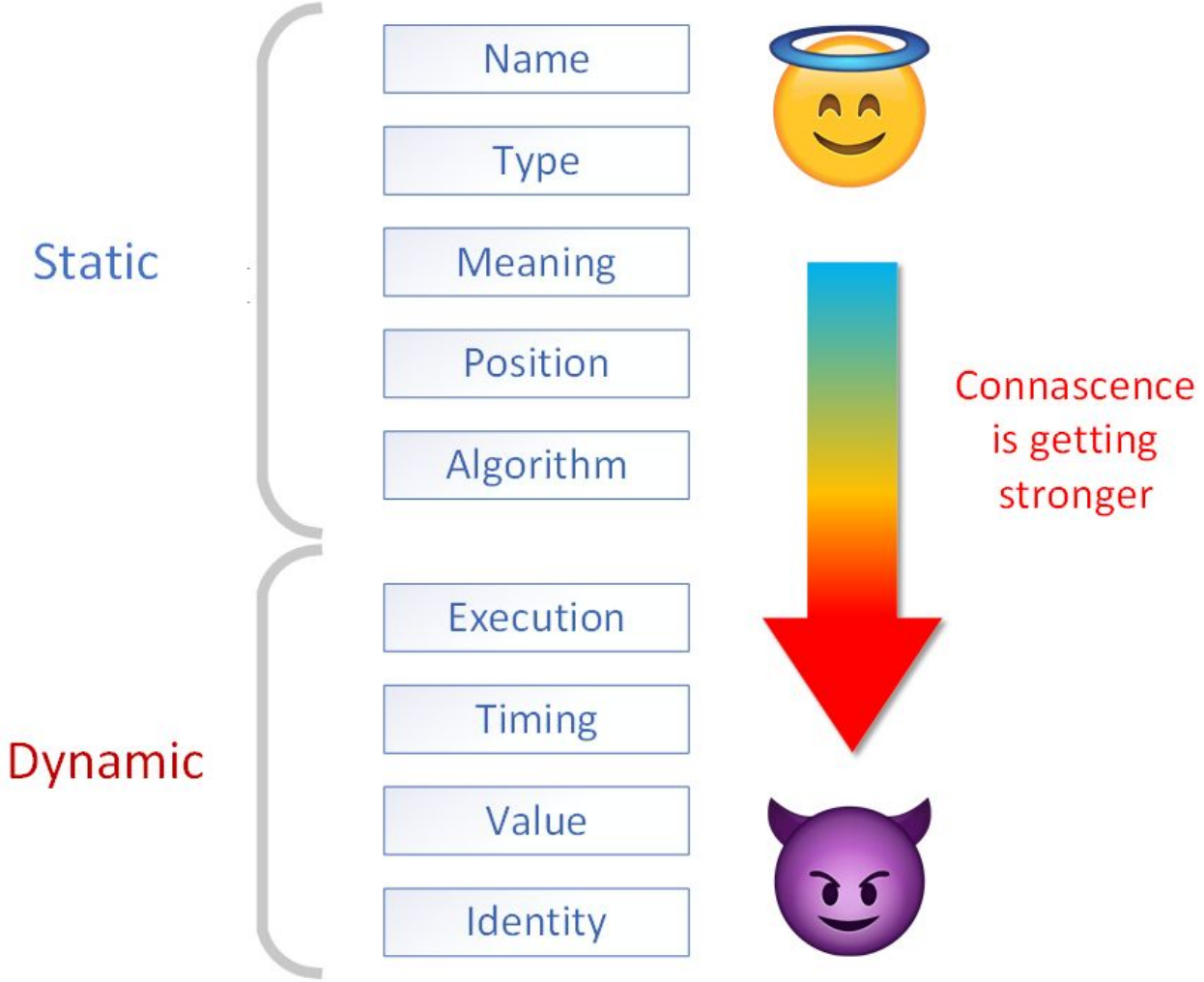
Connascence is a software quality metric & a taxonomy for different types of coupling.

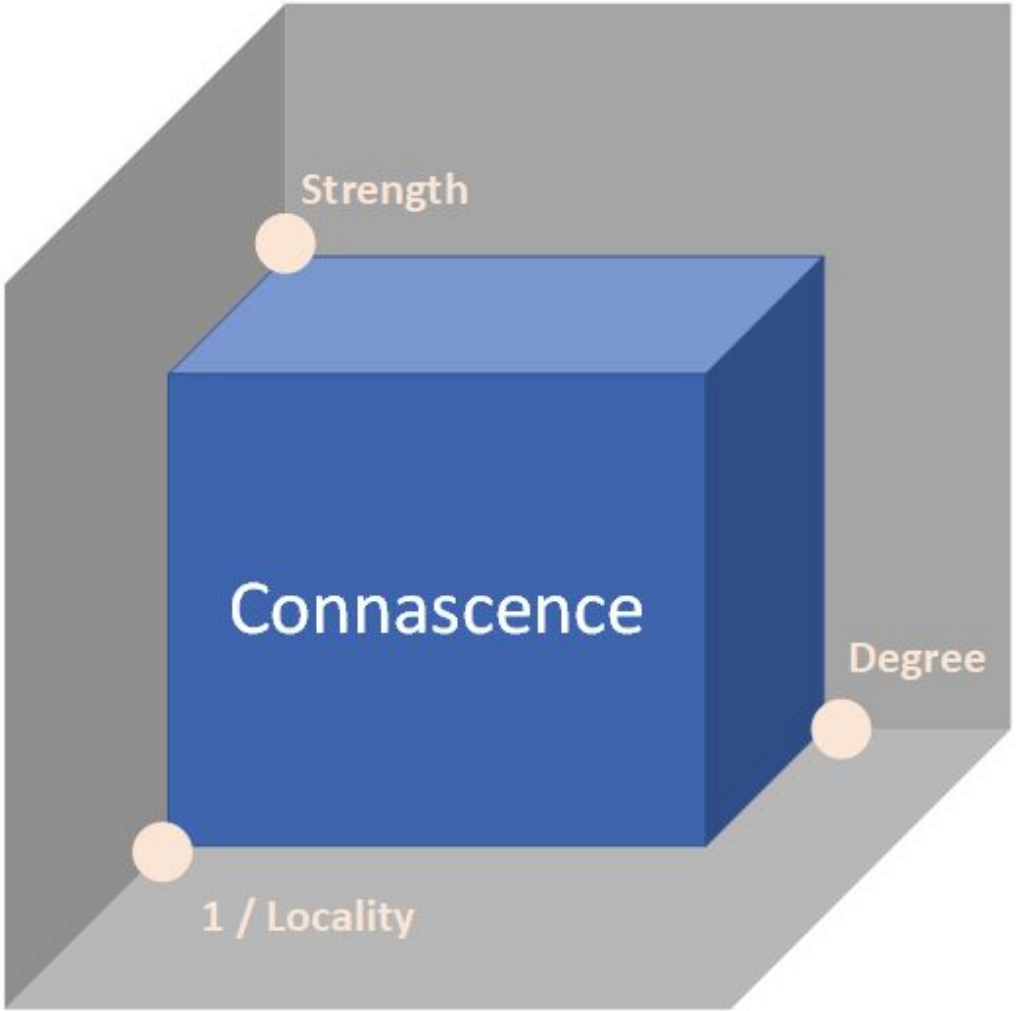
1. “connascence and coupling have the same definition, so i use the older term” - Kent Beck



Types of Connascence

- **Name:** when multiple components must agree on the name of an entity.
- **Type:** when multiple components must agree on the type of an entity.
- **Meaning:** when multiple components must agree on the meaning of specific values.
- **Position:** when multiple components must agree on the order of values.
- **Algorithm:** when multiple components must agree on a particular algorithm.
- **Execution (order):** when the order of execution of multiple components is important.
- **Timing:** when the timing of the execution of multiple components is important.
- **Value**: when the value of two components are related
- **Identity:** when multiple components must reference the entity.


Static
Name
Type
Meaning
Position
Algorithm
Dynamic
Execution
Timing
Value
Identity
Connascence
is getting
stronger



Connascence is a metric, and like all metrics is an imperfect measure. However, connascence takes a more holistic approach, where each instance of connascence in a codebase must be considered on three separate axes:

1. Strength. Stronger connascences are harder to discover, or harder to refactor.
2. Degree. An entity that is connascent with thousands of other entities is likely to be a larger issue than one that is connascent with only a few.
3. Locality. Connascent elements that are close together in a codebase are better than ones that are far apart.

Formes statiques

Name

Connaissance du nom:

- des paquetages/espace de nom
- des interfaces
- des classes
- des champs
- des méthodes

Name

Naissance du nom:

- un nom pour désigner, expliquer, lever l'ambiguïté
- éviter les mots valises
- si vous ne pouvez le nommer c’est que vous ne pouvez l’expliquer
- les noms composés ont un problème (SRP)
- poser un nom lors du TDD
  - qu’est ce que je manipule ?
  - que sait il faire ?
  - qu’est ce que j’attends de lui ?

Mal nommer les choses, jugeait Camus, c'est ajouter au malheur du monde. Ne pas nommer les choses, c'est nier notre humanité.

apocryphe Suite à un accident grave de voyageur (2013) de Eric Fottorino

Of names

Un objet se définit par ce qu'il est => pas de huuuu_**er**

Une méthode définit ce qu'il fait

```
public class CashFormater
 {
     private readonly int dollars;

     public CashFormater(int dollars)
     {
         this.dollars = dollars;
     }

     public string Format()
     {
         return string.Format("%d $", this.dollars);
     }
 }
```

meaningless names

Manager

Controller

Handler

Dispatcher

Helper

Runner

Formater

Writer, Reader

Validator

Router

Processor

Listener

Observer

Encoder

Decoder

Sorter

meaningful names

~~Manager~~

~~Controller~~

~~Handler~~

~~Dispatcher~~

~~Helper~~

~~Runner~~

~~Formater~~

~~Writer, Reader~~

~~Validator~~

~~Router~~

~~Processor~~

~~Listener~~

~~Observer~~

~~Encoder~~

~~Decoder~~

~~Sorter~~

CashValidated

RoutedMessage

EncodedText

FormatedData

SortedList

...

Names for Methods

Constructeurs	Noms
Manipulateurs	Verbes
Transformateurs / Mutations	Verbes

Anglais? Français? Franglais? Qui parle?

Le langage de programmation est en anglais.

Le métier est en français.

Pour le compilateur, les noms sont des symboles.

Compiler n’est pas lire.

Qui lit?

Vous avez dit : fluent ?

“Oui mais les librairies que j’utilise n’ont des noms anglais” 🤓

```
public string LireDepuisFichier(string cheminVersFichier)
        {
            return
System.IO.File.ReadAllText(cheminVersFichier);
        }
```

=> Dépendance!

DRY: noms et pas types

Ne répétez pas!

- Enums
- Listes
- Abstract
- Implementation
- Interfaces

Le compilateur sait, donc vous savez!

Des choses simples pour se mettre sur la voie:

- Ce qui est collectionnable est pluriel

Type

A quoi servent les types?

Que nous disent les types?

-> Capacité à

Les Signatures sont des types

compile complete:
0
errors

NULL pointer exception
at stack trace:
main() 54

Progress...?

Connascence of Meaning

Mais que veut dire ce Null ?

```
var result = GetTheThing();
if(result == null)
//thing does not exist
```

Comment le remplacer?

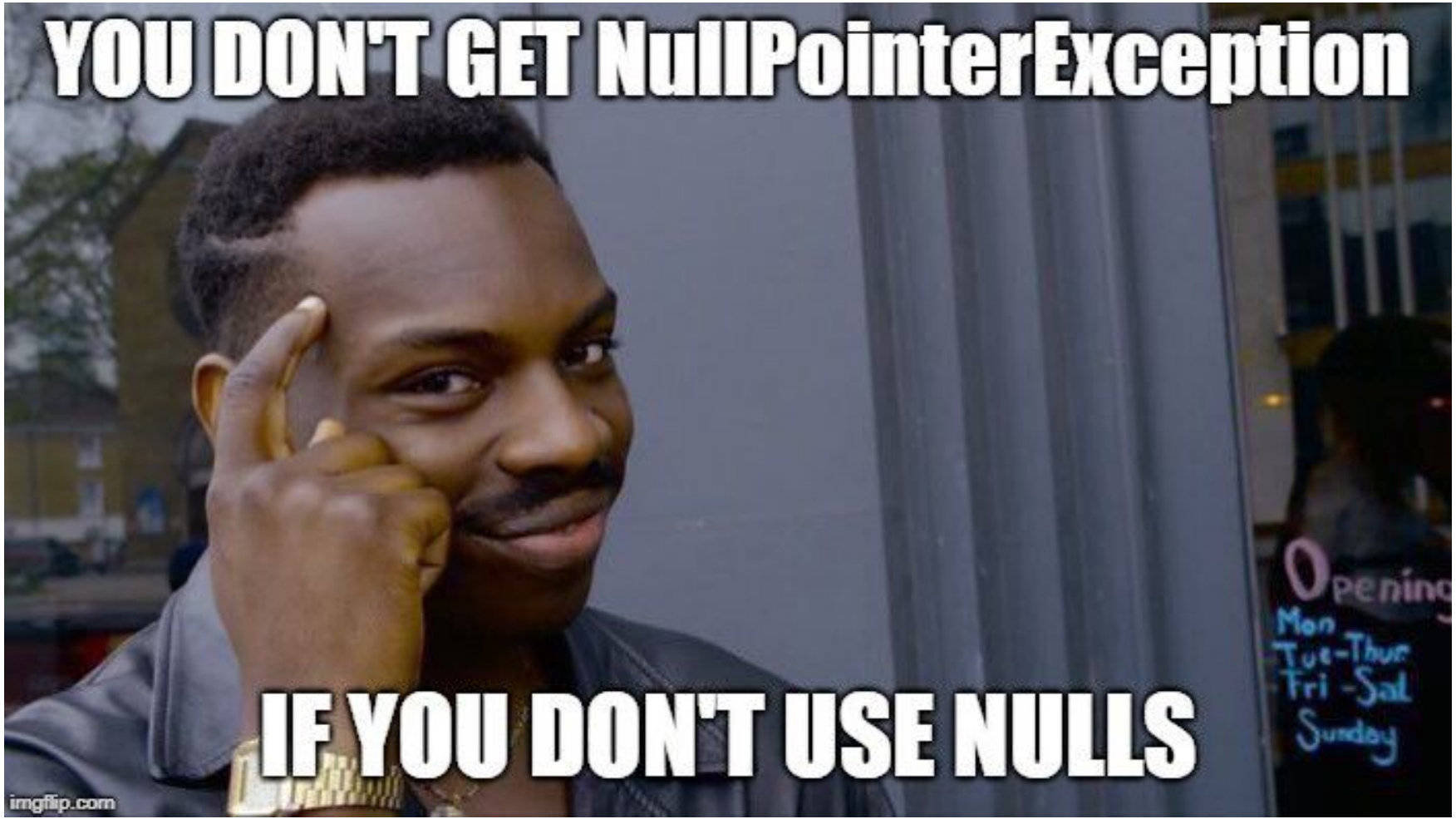
```
var result = GetTheThing();
if(result == string.Empty)
//thing does not exist
```

A suivre

Connascence of Execution

Seul le TDD peut vous sauver...

Oh, vraiment?


YOU DON'T GET NullPointerException
IF YOU DON'T USE NULLS
Opening
Mon
Tue-Thur
Fri -Sat
Sunday
imgflip.com

Connascence of Meaning

Qu’est ce que c’est que ces Strings?

- `"4917484589897107"`
- `“1700131555000”`
- “DE75512108001245126199”
- “1010-10-10”
- “31500”
- 1
- true

Comment les remplacer?

Connascence of Meaning

~~Strings~~ ?

- `"4917484589897107"`
- `“1700131555000”`
- “DE75512108001245126199”
- “1010-10-10”
- “31500”
- 1
- true

Types !

- `VisaCardNumber`
- `NIR:` numéro d'inscription au répertoire national d'identification des personnes physiques (RNIPP)
- IbanNumber
- DateAuFormatUniversel
- CodePostalFrançais
- NombreDePatates
- JeSuisUnDéveloppeur

“Primitive Obsession”

Public constants = hard coded coupling

```
public const String EOL = “\n\r”
```

```
public struct EOL {

.... make it immutable

}
```

Connascence of position

```
 public string SauverUtilisateur(string nom, string prenom, string ville)
        {
                return "ok";
        }  // 3 degrés de connaissance
```

Connascence of position

```
  public string SauverUtilisateur(string nom, string prenom, string ville)
        {
                return "ok";
        }  // 3 degrés de connaissance


SauverUtilisateur(nom: "albert", prenom: "charles", ville:"londres");
SauverUtilisateur(Utilisateur unUtilisateur)
```

Connascence of position

```
SauverUtilisateur(Utilisateur unUtilisateur)


  var truc = new object[] {"un", 11.3, new {a=1}};
var montant = truc[1];  //1 degré de connaissance (fois 3)
```

Connascence of Algorithm

```
        public string UneFonction(string data)
        {
            return Util.MicMacEndoder(  data );
        }

        public bool UneAutreFonction(string data)
        {
            if (!data.StartsWith("MICMAC"))
                throw new System.Exception();
            return true;
        }
```

Routes Map: Url -> Controller/Action

/login -> AuthController.Login()

Views Generate Url.

Log In

Dynamic Connascences

Connascence of Execution Order

```
public void EnvoyerCourrier(string expediteur, string corps)
        {
            var mailSystem = new Mailer();
            mailSystem.To( expediteur);
            mailSystem.Body(corps);
            mailSystem.Send();
        }
 public void EnvoyerCourrier(string expediteur, string corps)
        {
            var mailSystem = new Mailer();
            mailSystem.To( expediteur);
               mailSystem.Send();
            mailSystem.Body(corps);
        }
```

Couplage temporel

```
  public class Cash
    {
        private int cents;
        private int dollars;

        public int Cents { get => cents; set =>
cents = value; }
        public int Dollars { get => dollars; set =>
dollars = value; }

        public string Print()
        {
            return string.Format("%1.%2 $", dollars,
cents);
        }
    }
```

```
    public class Cash
    {
        private readonly int cents;
        private readonly int dollars;

        public Cash(int cents, int dollars)
        {
            this.cents = cents;
            this.dollars = dollars;
        }

        public string Print()
        {
            return string.Format("%1.%2 $", dollars,
cents);
        }
    }
```

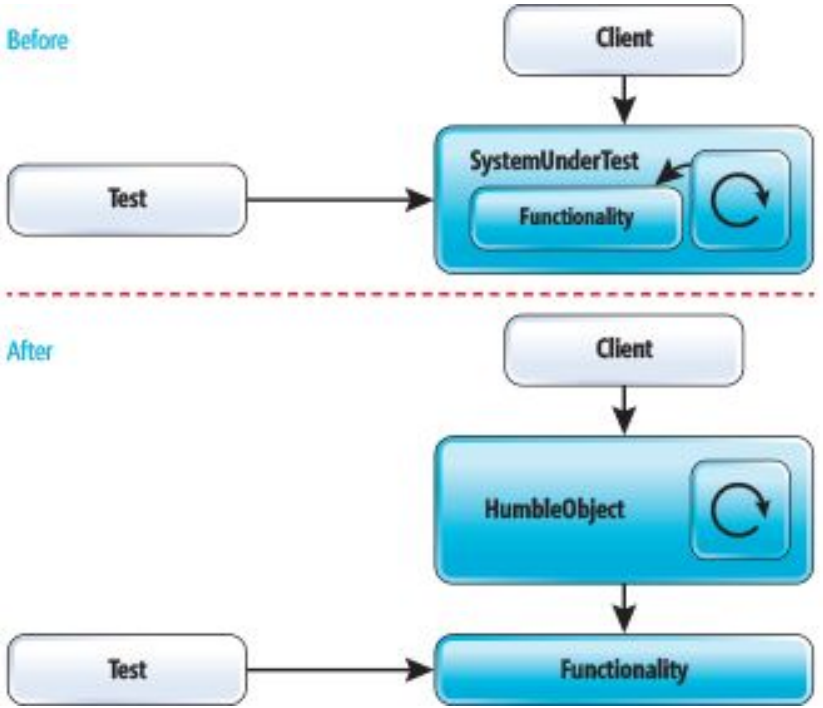
Connascence of timing

👃 **Wait**, **Sleep**, or **Timeout** followed by an actual arbitrary explicit timespan.

When( ).Then( ).AndThen( )

Connascence of timing

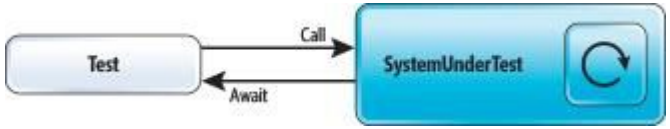
Separate Functionality from Multithreading



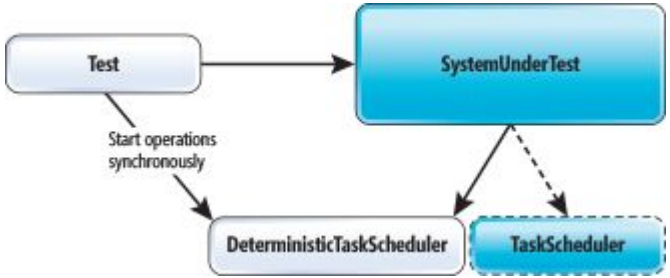
https://docs.microsoft.com/en-us/archive/msdn-magazine/2014/november/async-programming-unit-testing-asynchronous-code-three-solutions-for-better-tests

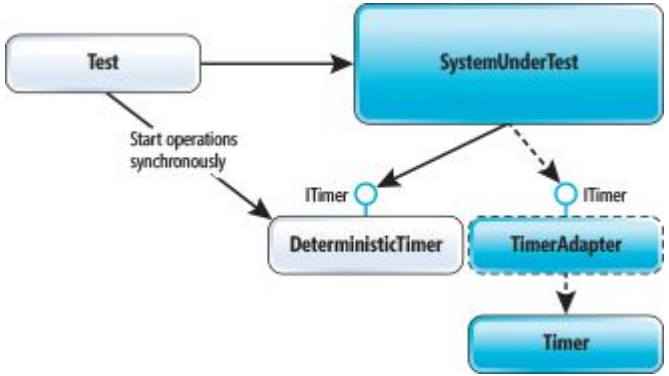
Synchronize Tests

with Instant Promises/Tasks



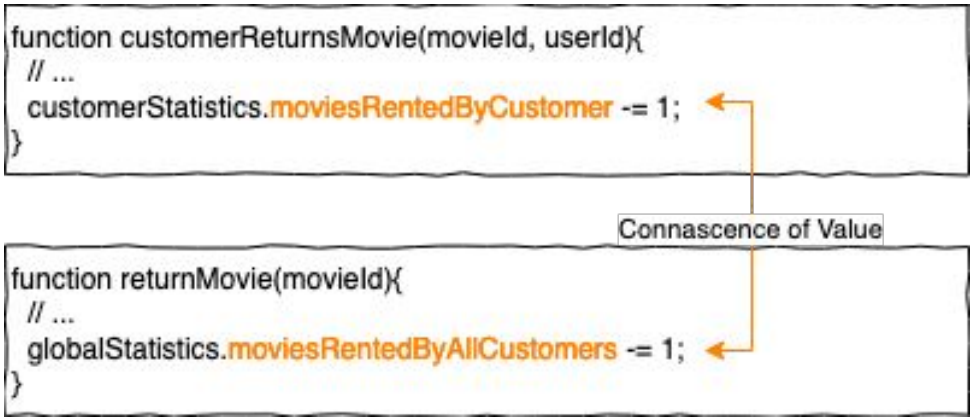
Replace the **Scheduler** by a **Deterministic** one





Connascence of Value (CoV)

when an invariant (permanent condition) states that two or more values must change simultaneously.

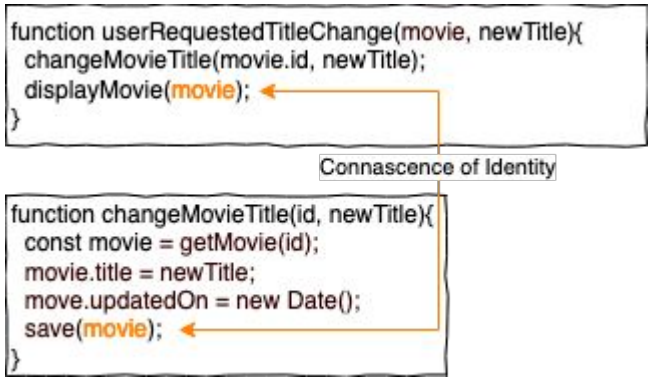


Connascence of Identity (CoI)

Quand le même objet est utilisé dans 2 contextes différents ----->

Plus compliqué:

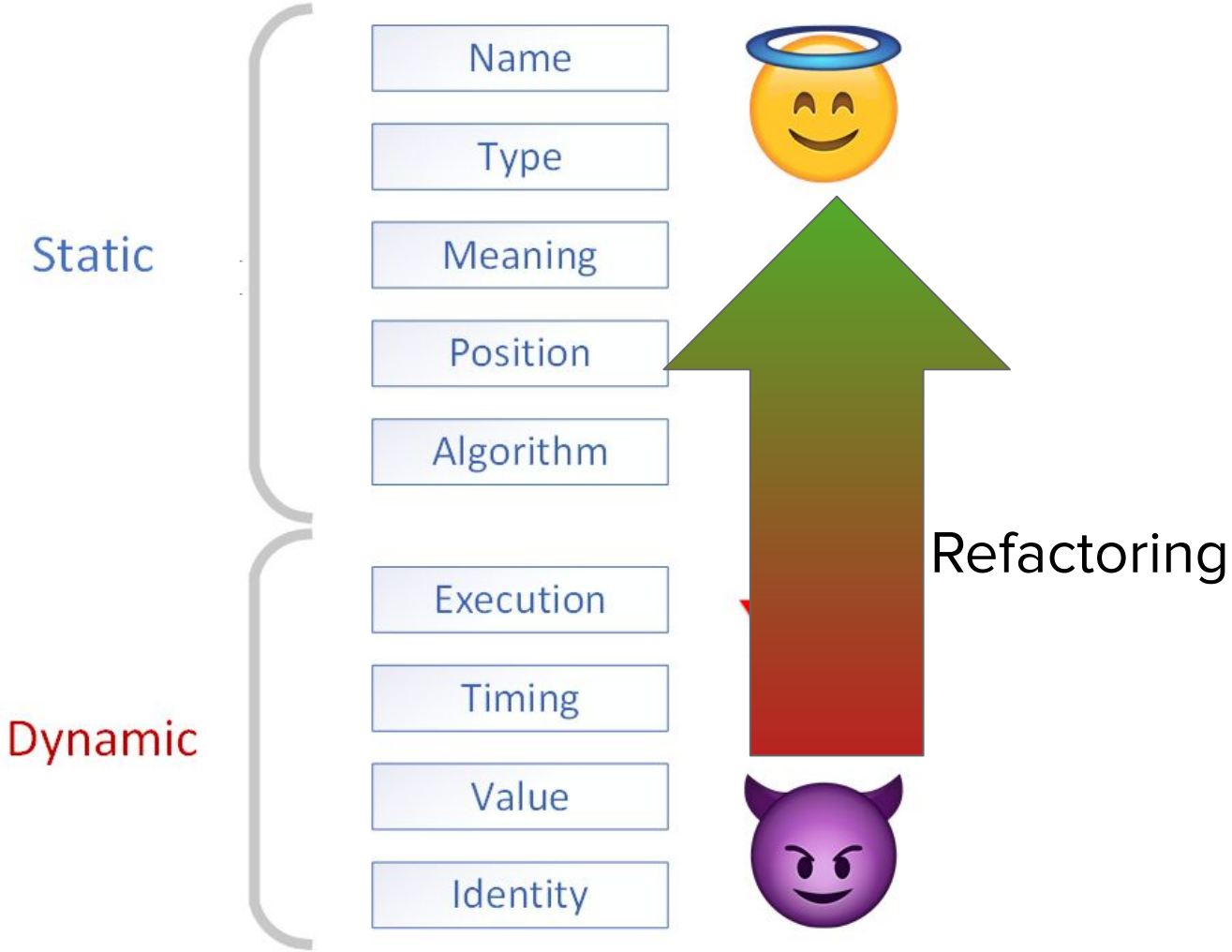
- Client / Server
- Cache
- ORM sans identity map

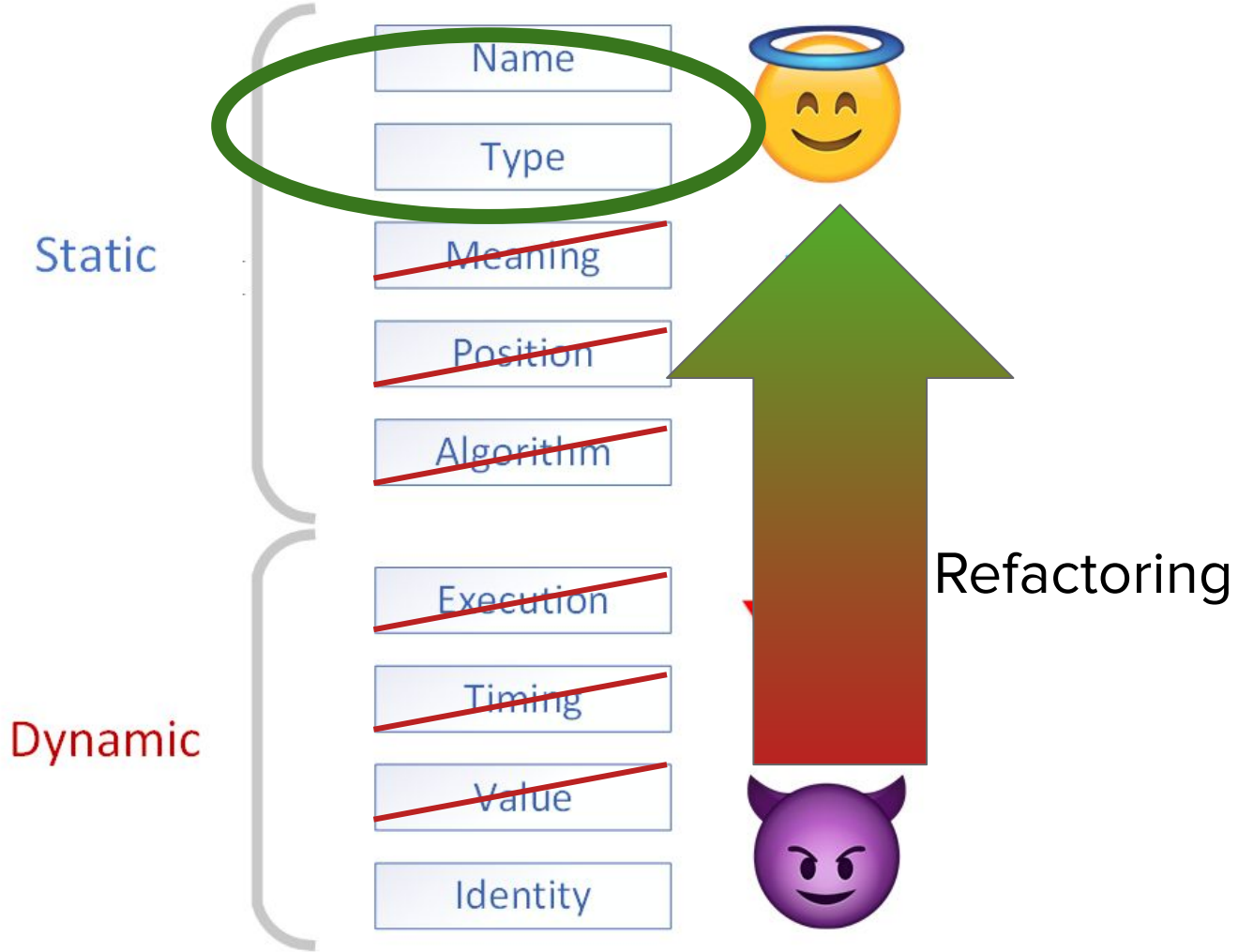


Avoir des Entités repérables par des identifiants Universels

Faire transiter les identifiants, pas les objets.

A la chasse!


Static
Name
Type
Meaning
Position
Algorithm
Dynamic
Execution
Timing
Value
Identity
Refactoring


Name
Type
Meaning
Position
Algorithm
Static
Execution
Timing
Value
Identity
Dynamic
Refactoring

Treat Dependencies As Code Smells

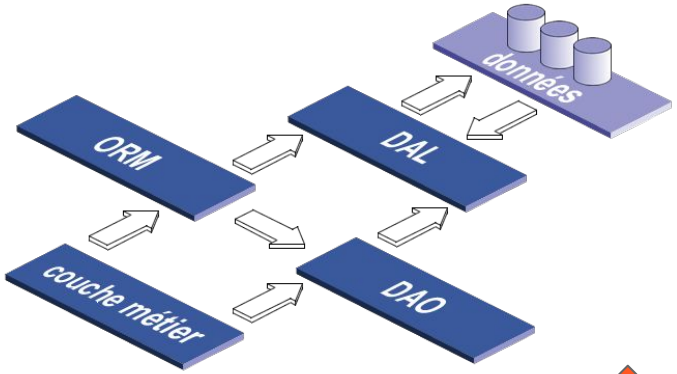
Imports / Using = 👃

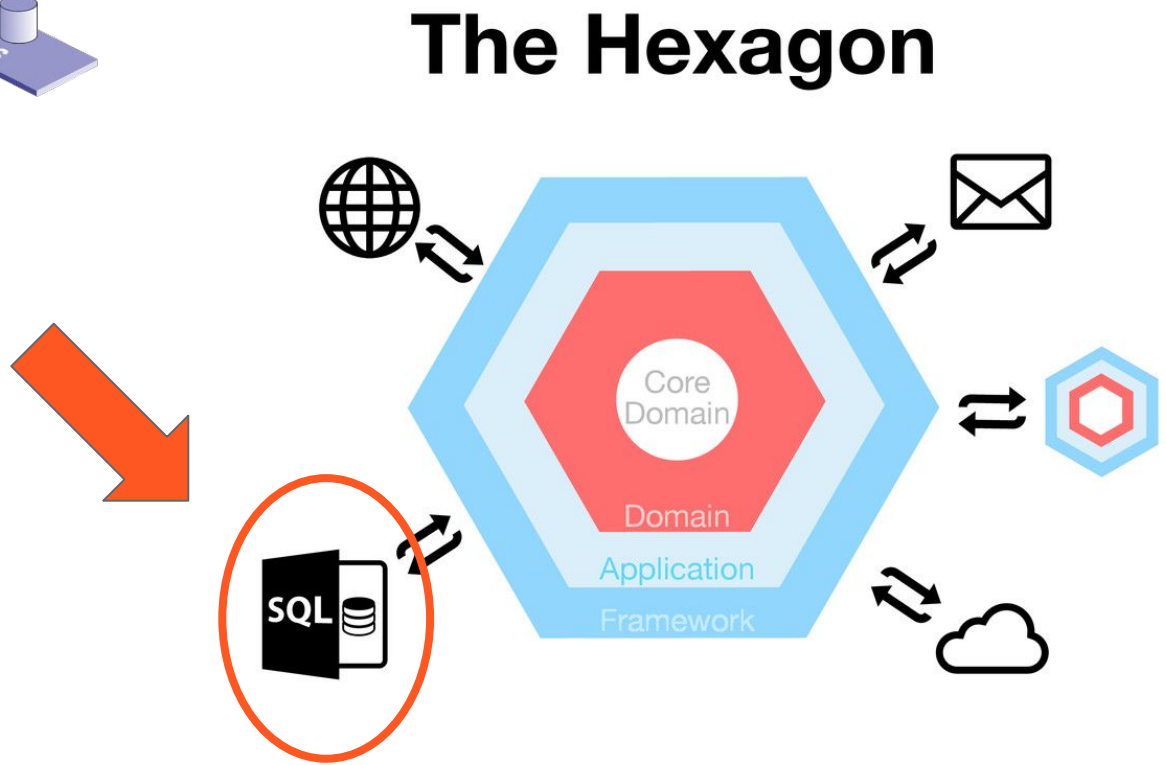
```
using System.Drawing;
using System.Windows.Forms;

using System.Data;
using System.Data.OleDb;
using ADODB;
```

```
using System.Messaging;
```

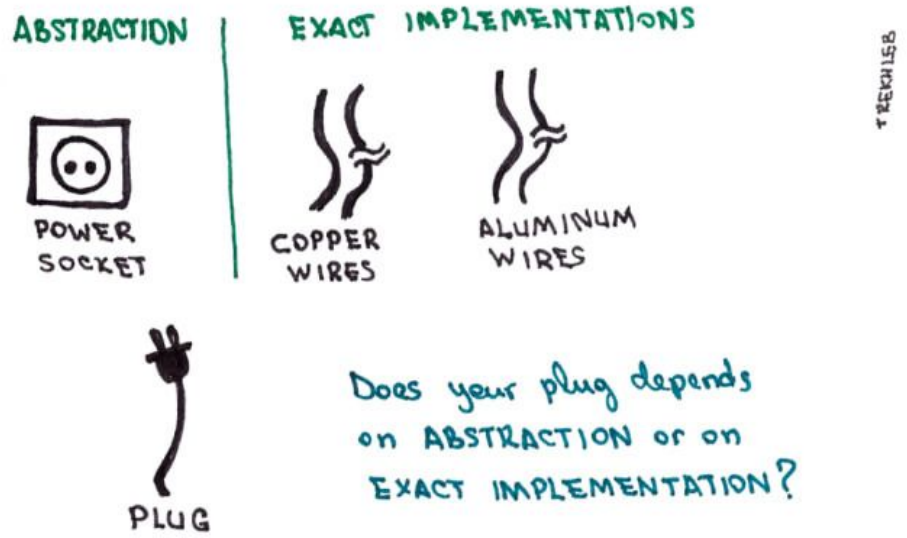
```
using RabbitMQ.Client;
```

No data layer dependency





Inversion de Dépendances



https://francescociulla.com/solid-principles-around-you-in-javascript

S.
O.
L.
I.

DEPENDENCY INVERSION PRINCIPLE

One should depend upon abstractions, not concretions.

- High-level modules should not depend on low-level modules. Both should depend on ABSTRACTIONS
- ABSTRACTIONS should not depend on details. Details should depend on abstrations.

Abstractions!

1. Dépendre uniquement des abstractions
2. Code against interfaces, not implémentations

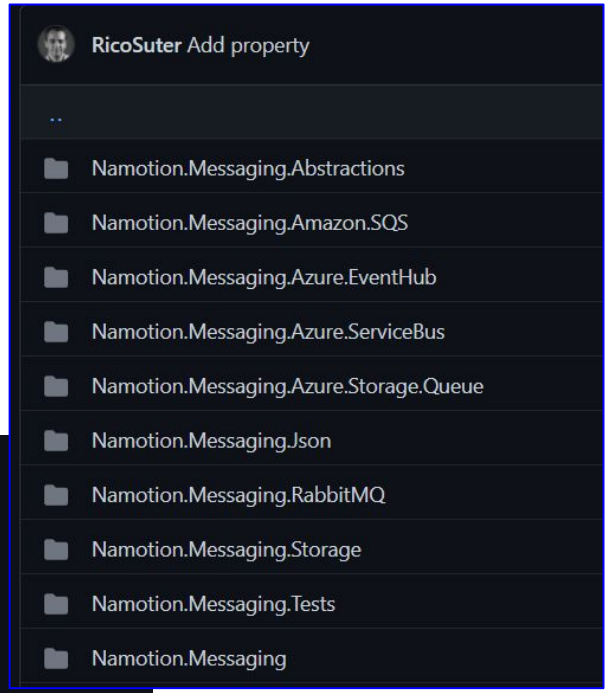
```
// Creating hard coded wheels
   DefaultWheels hardCodedWheels = new DefaultWheels();
   // Injecting hard coded wheels
   Car nonMantainableCar = new Car(hardCodedWheels);
```

```
 Wheels defaultWheels = new DefaultWheelsImpl();
   Wheels AwesomeWheelsImpl = new  AwesomeWheelsImpl();
   // New car with default wheels
   MaintainableCar maintainableCar = new
MaintainableCar(defaultWheels);
// easy change
maintainableCar.changeWheels(AwesomeWheelsImpl);
```

Abstractions everywhere

```
IMessageReceiver receiver = ServiceBusMessageReceiver
        .Create("MyConnectionString", "myqueue");

await receiver.ListenAsync(async (messages, ct) =>
{
    ...
}, CancellationToken.None);
```

- IMessagePublisher<T>
- IMessagePublisher
  - `PublishAsync(messages, cancellationToken)` : Sends a batch of messages to the queue.
- IMessageReceiver<T>
- IMessageReceiver
  - `GetMessageCountAsync(cancellationToken)` : Gets the count of messages waiting to be processed.
  - `ListenAsync(handleMessages, cancellationToken)` : (Fails when connection cannot be established)
  - `ListenWithRetryAsync(handleMessages, cancellationToken)` : Starts listening and processing messages with the `handleMessages` function until the `cancellationToken` signals a cancellation.
  - `KeepAliveAsync(messages, timeToLive, cancellationToken)` : Extends the message lock timeout on the given messages.



https://github.com/RicoSuter/Namotion.Messaging

Abstractions everywhere

https://blog.rsuter.com/logging-with-ilogger-recommendations-and-best-practices/
https://github.com/System-IO-Abstractions/

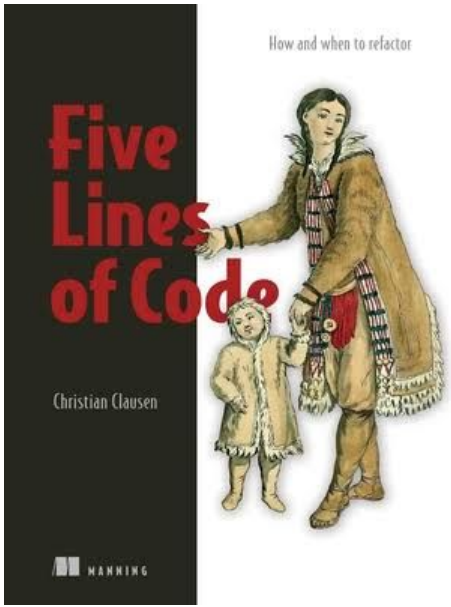
Des recettes de cuisine

Pas d'algorithmes dans les constructeurs!

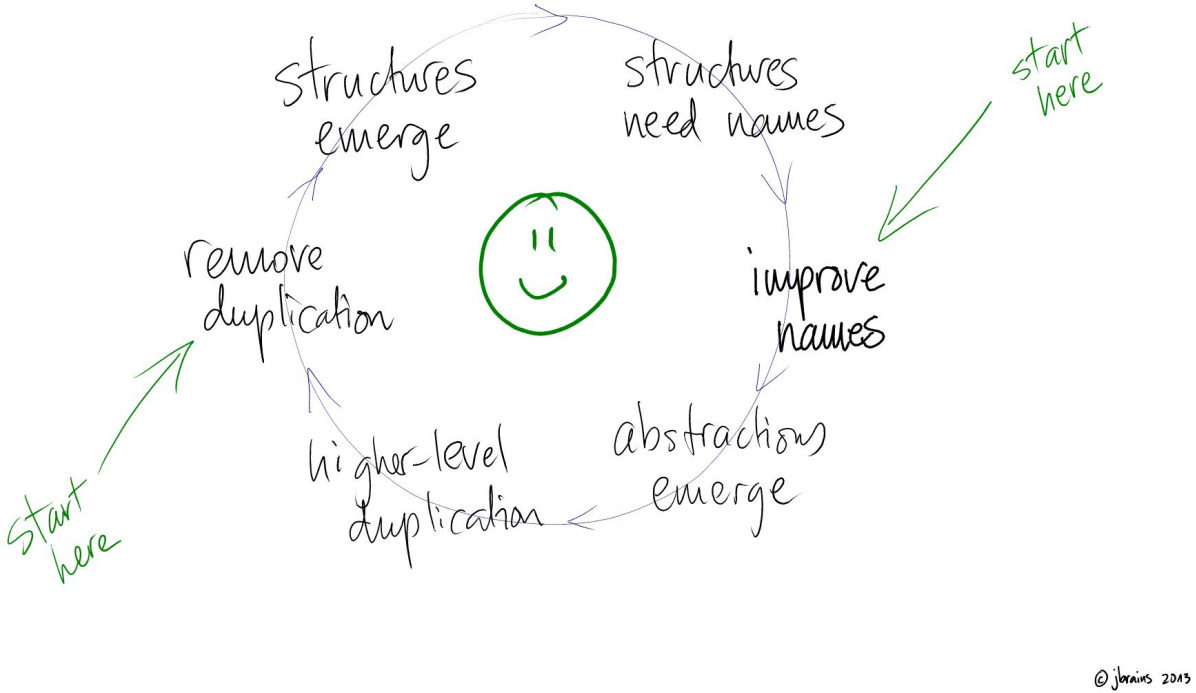
Encapsuler aussi peu que possible
-> dans un objet, pas plus de 4 éléments dans sa composition

Pas plus de 5 méthodes

...



Un travail de tous les jours



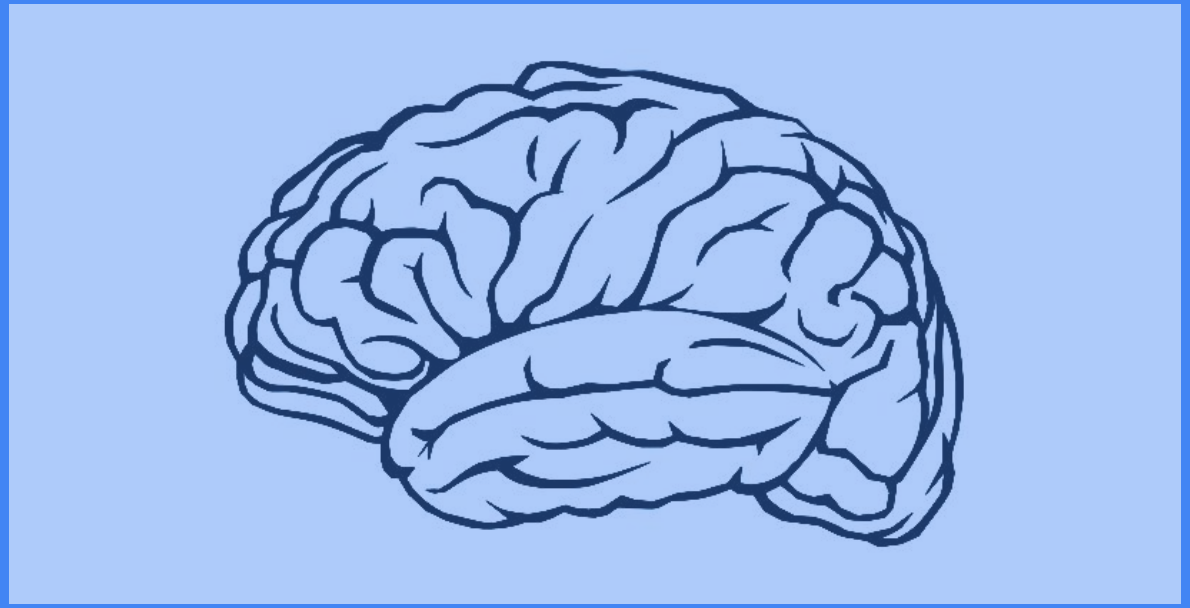
https://blog.thecodewhisperer.com/permalink/putting-an-age-old-battle-to-rest

Coder c’est écrire


Victor Hugo
36,5 %
Marcel
Pagnol
35,8 %
Jules Verne
30 %
Émile Zola
28,2 %
Guy de

Notre prose est imparfaite

Couplée à notre



Implicite

Explicite

Un chef cuisine pour les autres

Si je ne comprends pas ton code, c'est de TA faute.

COMPASSIONATE CODING