DDD et programmation fonctionnelle

Des alliés naturels

Agile Tour Sherbrooke - 26 mai 2018

Olivier Lafleur, M.Sc., PSM
Coach agile
Mentor dans le programme de résidence
@olilafleur

Qui sait ce qu’est le DDD ?

Qui a déjà fait de la programmation fonctionnelle?

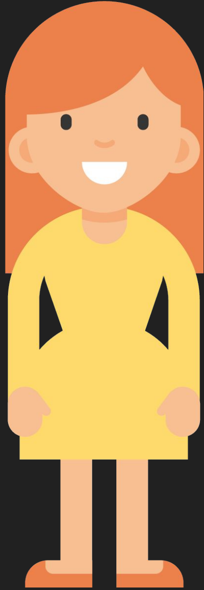
Domain-Driven DESIGN

Tackling Complexity in the Heart of Software

Eric Evans

Foreword by Martin Fowler

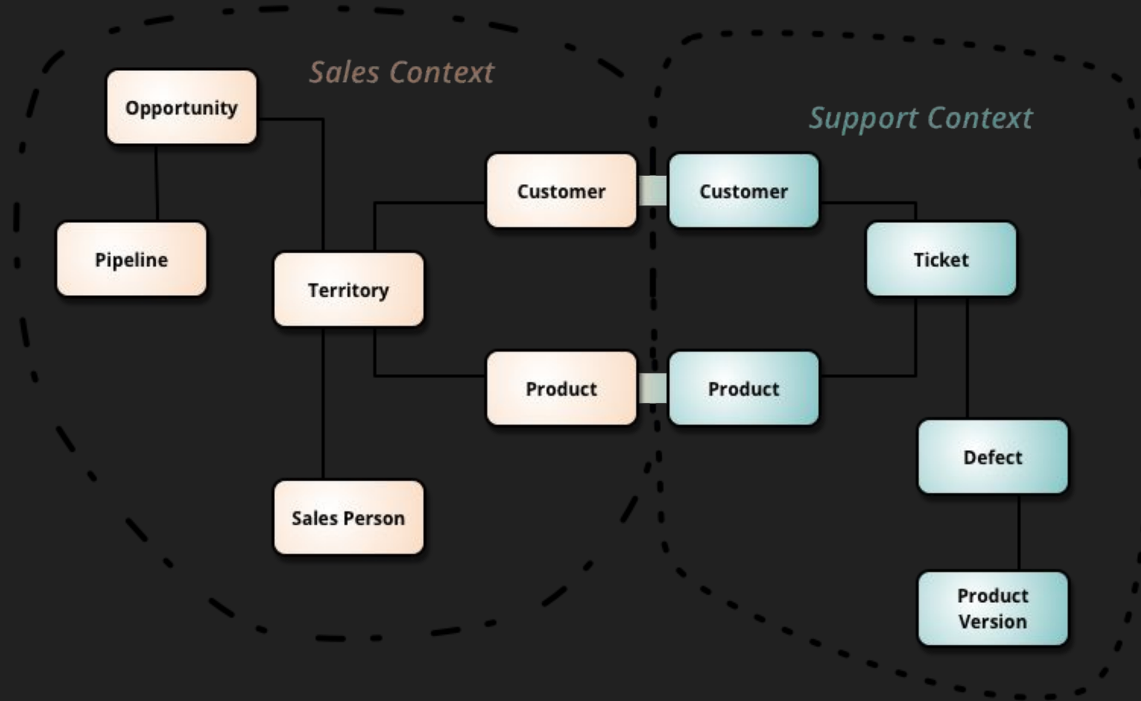
Langage commun (*ubiquitous*)



Développeurs

Experts du domaine d'affaires

Séparer les contextes



Modéliser les règles d’affaires, en faisant abstraction de la plateforme

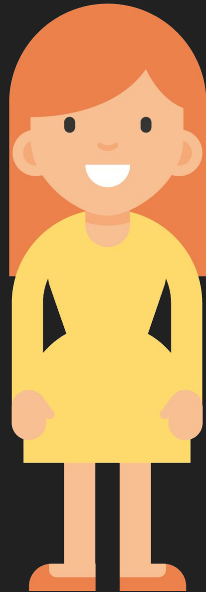
Macintosh HD — top — 80×24

```
Processes: 210 total, 2 running, 9 stuck, 199 sleeping, 901 threads    23:30:03
Load Avg: 1.40, 1.75, 1.00  CPU usage: 4.15% user, 4.40% sys, 91.44% idle
SharedLibs: 1648K resident, 0B data, 0B linkedit.
MemRegions: 31278 total, 1892M resident, 117M private, 564M shared.
PhysMem: 5893M used (1191M wired), 10G unused.
VM: 523G vsize, 1026M framework vsize, 0(0) swapins, 0(0) swapouts.
Networks: packets: 12105/8925K in, 11907/1964K out.
Disks: 80156/2205M read, 21235/425M written.

PID  COMMAND      %CPU TIME     #TH  #WQ  #PORT MEM    PURG   CMPR PGRP PPID
592  screencaptur 0.0  00:00.02 7    5    55+   1952K+ 20K+   0B   262  262
590  mdworker     0.0  00:00.01 3    0    44    2032K  0B     0B   590  1
589  mdworker     0.0  00:00.01 3    0    44    1572K  0B     0B   589  1
588  top          1.7  00:00.51 1/1  0    22+   2860K  0B     0B   588  584
584  bash         0.0  00:00.00 1    0    15    588K   0B     0B   584  583
583  login        0.0  00:00.01 3    1    28    1228K  0B     0B   583  482
574  auditd       0.0  00:00.00 2    0    25    560K   0B     0B   574  1
567  System Prefe 0.0  00:03.23 3    0    270   39M    8364K  0B   567  1
561  systemstatsd 0.0  00:00.01 2    1    19    1040K  0B     0B   561  1
560  com.apple.We 0.0  00:01.42 9    0    229   25M    0B     0B   560  1
558  com.apple.We 0.0  00:05.07 15   3    224   151M   1716K  0B   558  1
555  bash         0.0  00:00.00 1    0    15    604K   0B     0B   555  554
554  login        0.0  00:00.01 3    1    28    1176K  0B     0B   554  482
550  bash         0.0  00:00.00 1    0    15    608K   0B     0B   550  549
```

Quand **ne pas** utiliser le DDD ?

Quand **ne pas** utiliser le DDD ?

La communication entre clients et devs et gens d’affaires peut être problématique



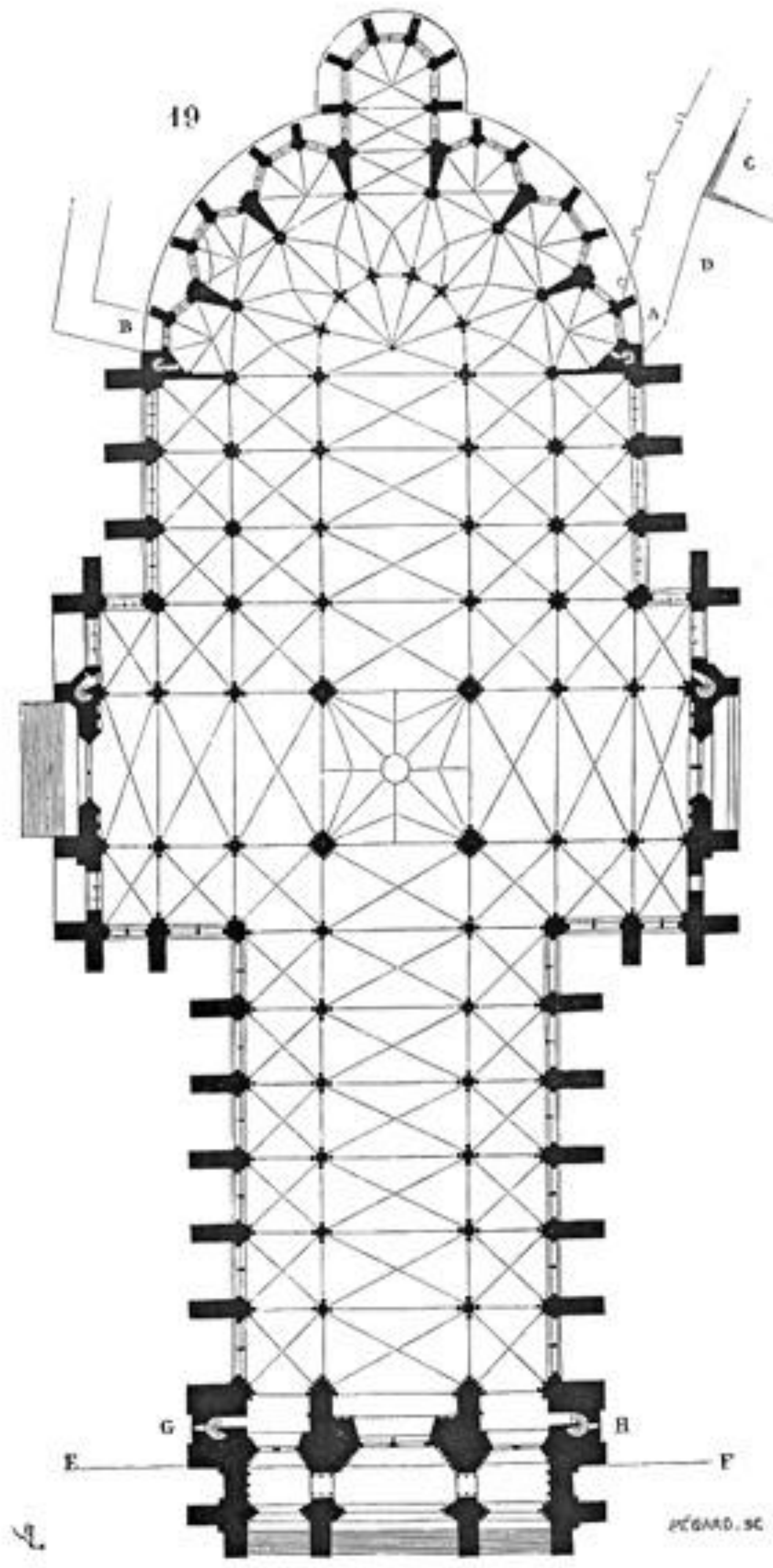
BDD

```
Given …
When …
Then …
```

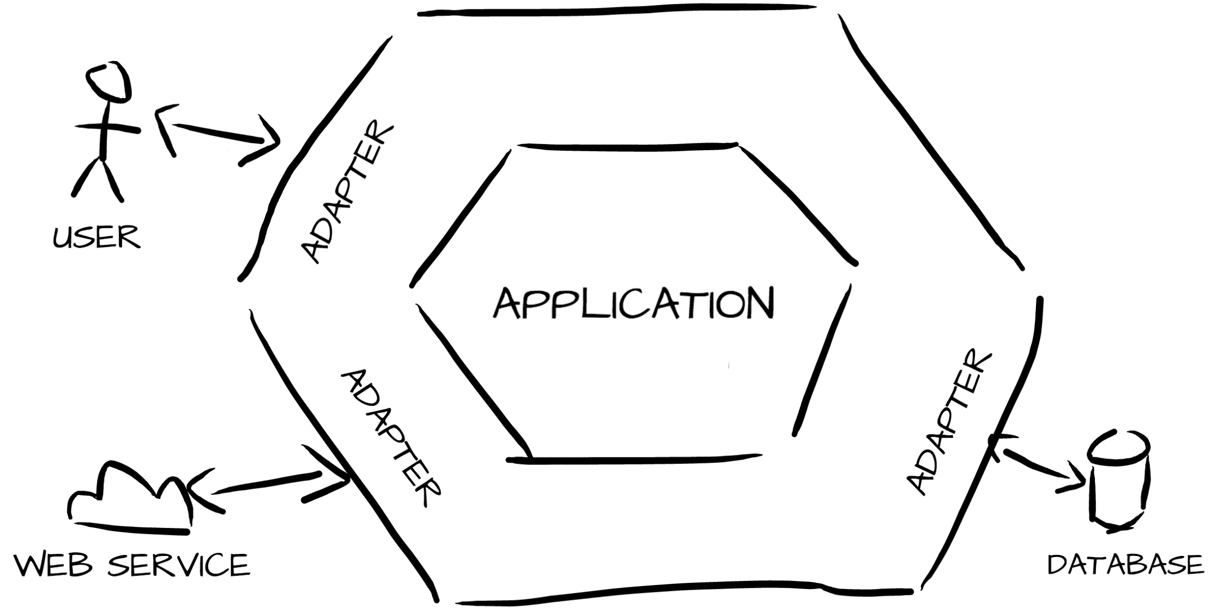
Peut-on viser une documentation exécutable (i.e. du code) lisible par des gens d’affaires ?

Oui !

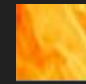
C’est quoi une bonne architecture?


19
B
C
D
A
G
H
E
F

Architecture hexagonale

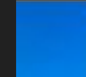


Mais où est la programmation fonctionnelle là-dedans ?



Danger / Chaos

Bordure / Mapping



Confortable / Sécure

source : http://slides.com/jfbourget/formation-functional-programming#/3/2

En fonctionnel, on modélise avec des **types** et des **fonctions**.

```
type Sorte = Coeur | Trèfle | Pique | Carreau

type Face = Deux | Trois | Quatre | Cinq | Six | Sept |
  Huit | Neuf | Dix | Valet | Dame | Roi

type Carte = {
   sorte: Sorte
   face: Face
}

type Main = Carte list
```

Avez-vous détecté l’erreur dans le code?

```
type Sorte = Coeur | Trèfle | Pique | Carreau

type Face = Deux | Trois | Quatre | Cinq | Six | Sept |
  Huit | Neuf | Dix | Valet | Dame | Roi | As

type Carte = {
   sorte: Sorte
   face: Face
}

type Main = Carte list
```

A language that doesn’t affect the way you think about programming is not worth knowing.
— Alan Perlis

Uncle Bob Martin
@unclebobmartin

Abonné

Functional Programming, Object Programming, and Structured Programming are all part of the same whole: Programming. The structure of any system will benefit from a synthesis of all three; and will be less than optimal if one or more are excluded.

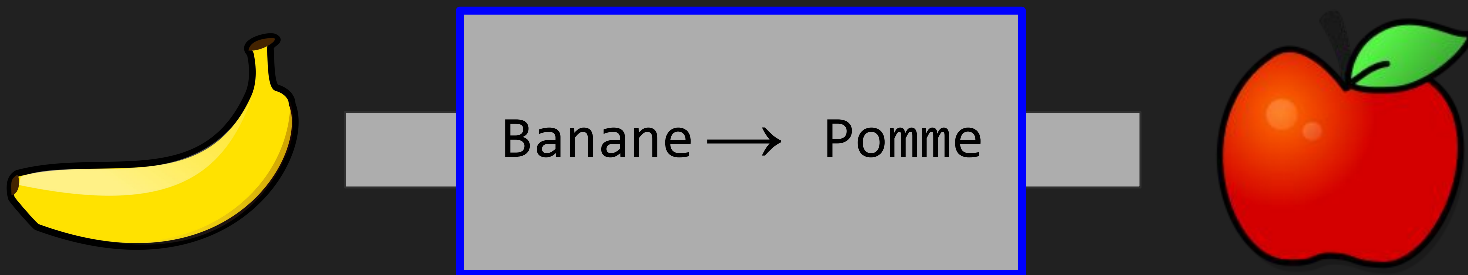
Traduire le Tweet

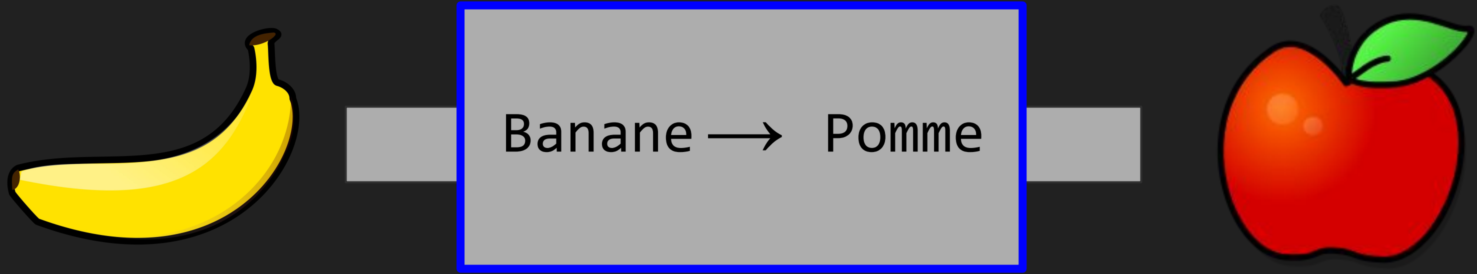
09:34 - 18 mai 2018

163 Retweets **413** J'aime

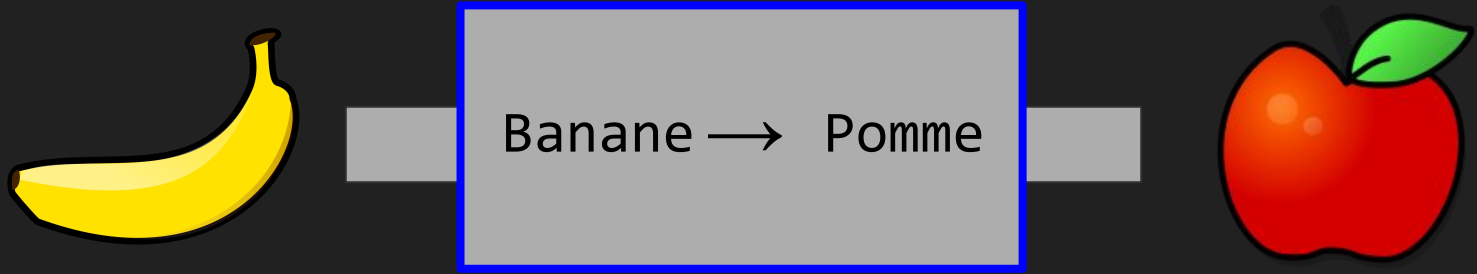
13 163 413

Une fonction, c'est quoi?





Un tunnel de transformation

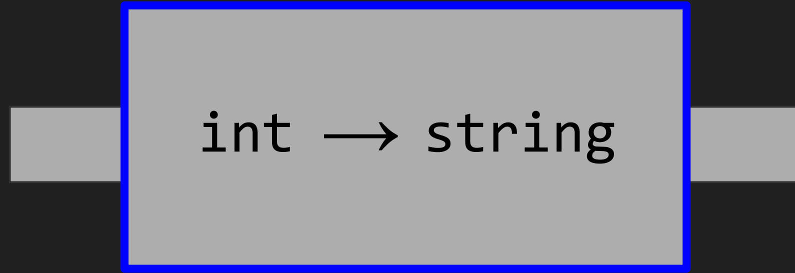


Aucun effet “extérieur”

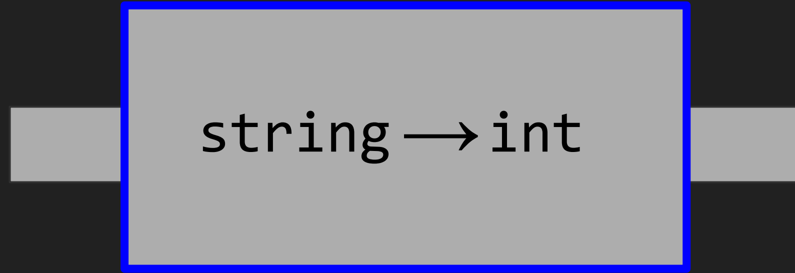
En F#, tout est une fonction :

```
let carré x = x * x // signature int -> int

let grandeur = 6 // signature unit -> int

let addition x y = x + y // signature (int, int) -> int
```

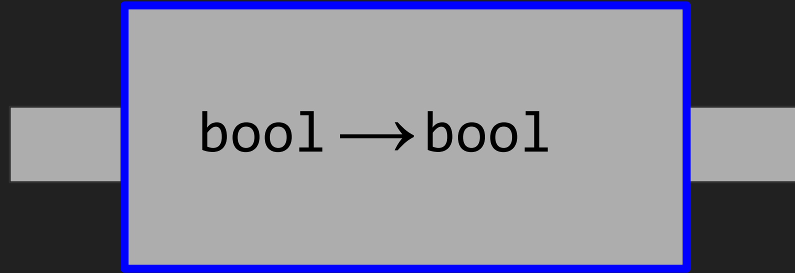
Même si les types sont implicites, ils sont plus “forts” que dans certains langages populaires.



Un exemple de fonction qui a cette signature?



Un exemple de celle-ci?



Combien existe-t-il de fonctions qui ont cette signature?

La composition de types

```
type SaladeDeFruits = {
   Pomme: VariétéDePomme
   Banane: VariétéDeBanane
   Cerise: VariétéDeCerise
}
```

La composition de types

```
type SaladeDeFruits = {
   Pomme: VariétéDePomme
   Banane: VariétéDeBanane
   Cerise: VariétéDeCerise
}
```

C’est un **ET**, une combinaison de plusieurs éléments

```
type VariétéDePomme =
  | GoldenDelicious
  | GrannySmith
  | Fuji

type VariétéDeBanane =
  | Cavendish
  | GrosMichel
  | Manzano
```

```
type VariétéDePomme =
  | GoldenDelicious
  | GrannySmith
  | Fuji

type VariétéDeBanane =
  | Cavendish
  | GrosMichel
  | Manzano
```

C’est un **OU**, le choix entre plusieurs éléments

Pattern matching

Par exemple :

```
let imprimerTypePomme variete =
  match variete with
  | GoldenDelicious ->
     printfn "Une Golden Delicious"
  | GrannySmith ->
     printfn "Une Granny Smith"
  | Fuji ->
     printfn "Une Fuji"
```

Et parfois aussi un choix simple

```
type CodeDeProduit =
  | CodeDeProduit of string

// ou

type CodeDeProduit = CodeDeProduit of string
```

`null`

types simples

Mais alors, comment on représente un champ “nullable” ou plus généralement, le concept d’absence?

optional

Ok, mais est-ce que tout ça est utilisable dans la vraie vie?

Modélisation d’un cas réaliste

Le paiement d’une facture

Pour le paiement, on veut :

- un montant
- une devise
- une méthode de paiement.

Le montant du paiement

```
type MontantPaye = MontantPaye of decimal // emballage

type Devise = EUR | USD | CDN // type "OU"
```

La méthode de paiement

```
type NoCheque = NoCheque of int

type NoCarte = NoCarte of string
type TypeCarte = Visa | Mastercard

type InfoCarteCredit = {
    TypeCarte: TypeCarte
    NoCarte: NoCarte
}
```

```
type MéthodePaiement =
  | Comptant
  | Chèque of NoCheque
  | CarteCredit of InfoCarteCredit
```

Le paiement comme tel

```
type Paiement = {
    Montant: MontantPaye
    Devise: Devise
    Méthode: MéthodePaiement
}
```

Le consommateur

```
type Consommateur = {
    Prénom: string
    DeuxièmeNom: string
    NomDeFamille: string
}
```

Le consommateur

```
type Consommateur = {
    Prénom: string
    DeuxièmeNom: string
    NomDeFamille: string
}
```

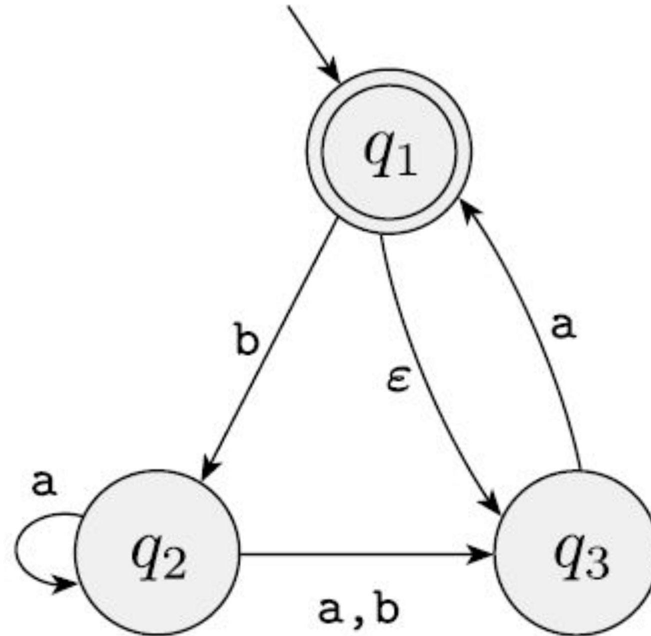
Comment rendre le deuxième nom facultatif?

Le consommateur

```
type Consommateur = {
    Prénom: string
    DeuxièmeNom: Option<string>
    // ou DeuxièmeNom: string option
    NomDeFamille: string
}
```

La modélisation du processus de paiement

Machine à états (automate)



Un item

```
type Item = ???

type DonneesPanierActif = { ItemsNonPayes: Item list }
type DonneesPanierPaye = { ItemsPayes: Item list; Paiement: Paiement }

type Panier =
  | PanierVide
  | PanierActif of DonneesPanierActif
  | PanierPaye of DonneesPanierPaye
```

Ajouter un item

```
let ajouterItem panier item =
 match panier with
  | PanierVide -> // créer un nouveau panier actif avec un item
      PanierActif { ItemsNonPayes = [item] }
  | PanierActif { ItemsNonPayes = itemsExistants }
      // créer un nouveau panier avec l'item ajouté
      PanierActif { ItemsNonPayes = item :: itemsExistants }
  | PanierPaye _ ->
      // on ignore
      panier
```

Faire un paiement

```
let fairePaiement panier paiement =
    match panier with
    | PanierVide ->
      panier
    | PanierActif { ItemsNonPayes = itemsExistants } ->
      PanierPaye { ItemsPayes = itemsExistants; Paiement = paiement }
    | PanierPaye _ ->
      panier
```

Comment gérer les erreurs dans ce monde “parfait” ?

En étant explicite

```
type ErreurValidationAdresse = ErreurValidationAdresse of string

type VérifieAdresseExiste =
 AdresseNonValidée -> Result<AdresseVerifiee, ErreurValidationAdresse>
```

Et l’injection de dépendances?

En orienté objet : classes sont passés en paramètres / injectés

En fonctionnel : ce sont des fonctions ayant certain types qui sont passées en paramètres

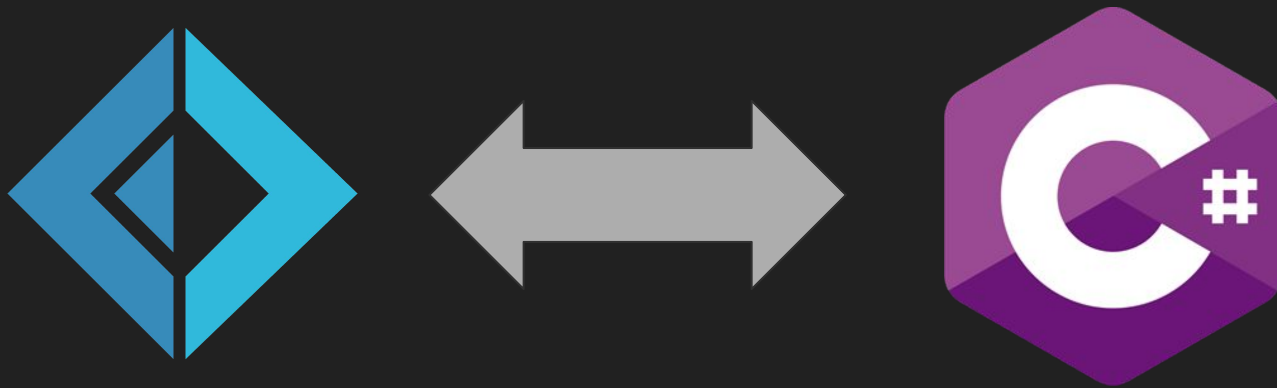
Et les tests unitaires?

Sont-ils encore nécessaires?

Oui !

Ok, mais lundi matin je fais quoi?

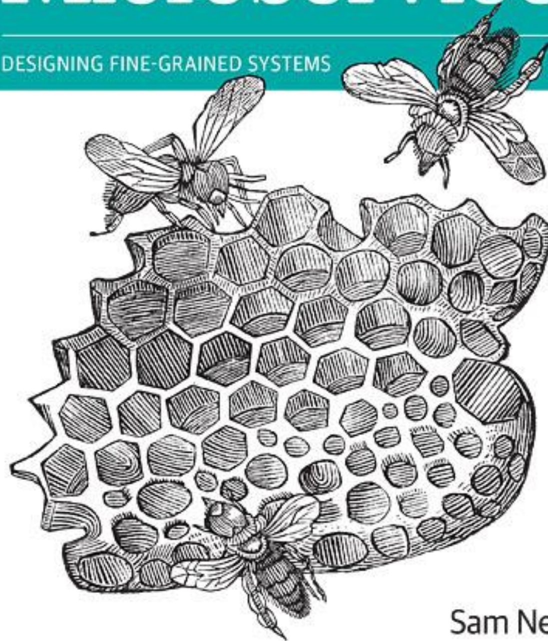
Interopérabilité



O'REILLY

Building Microservices

DESIGNING FINE-GRAINED SYSTEMS



Sam Newman

Et avec d’autres langages?

Et ces langages?

The Pragmatic Programmers

Domain Modeling Made Functional

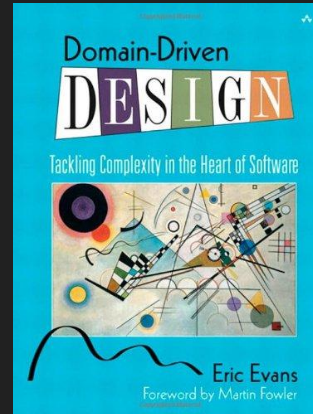
Tackle Software Complexity with Domain-Driven Design and F#

Scott Wlaschin

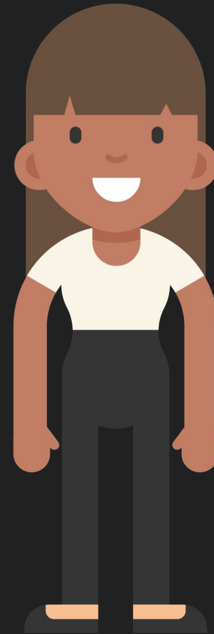
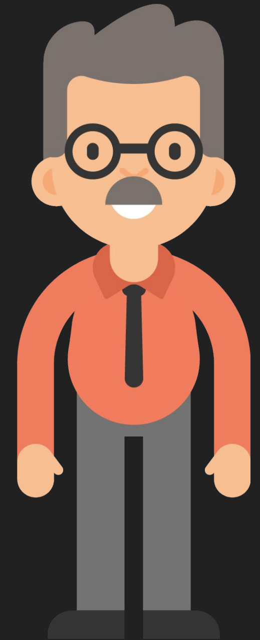
edited by Brian MacDonald

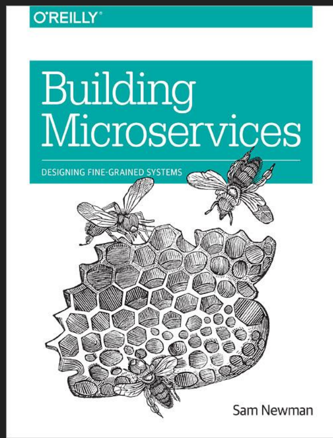
Conclusion

DDD et programmation fonctionnelle : des alliés naturels?



Merci pour votre écoute !



Questions?
Discussions?