

Software Architecture

Cours

Architecture

Penser pour moins dépenser

Dans notre domaine du
développement, nous cherchons
constamment à optimiser

Rendre le code

plus rapide

plus lisible

plus maintenable

etc

Ce sont de jolis mots, mais
concrètement ?

Améliorer le code

- Nommer ses variables / fonctions **explicitement**
 - `isVisible` > `b`
 - `disableUser(User user)` > `stop(User u)`
- Séparer son code en blocs logiques
 - Déclarer les variables là où elles seront utilisées
 - Utiliser les sauts de lignes comme un **séparateur d'intention** (comme un paragraphe dans un texte)

Améliorer le code

- Ecrire des fonctions **digérables**
 - Qui tiennent sur 1 écran (quelques dizaines de lignes)
 - Qui ne font pas 1000 choses
- Commenter là où c'est utile
 - Un comportement particulier, un bug, quelque chose de **non-évident**

**Toutes ces actions sont nécessaires
pour faciliter la maintenance**

Plus le code est lisible :

Plus vite on le comprend

Plus vite on peut le modifier

Ces bonnes pratiques sont nécessaires
mais pas suffisantes

Ce qui rend le debug souvent compliqué

C'est de devoir conserver en tête un
modèle mental de ce que fait le
programme

“ Je suis dans la condition X, je sais que la variable a vaut Y, que je suis passé par ici car la fonction Y a été appelée et normalement la classe W a été créé ,”

Garder tout cela en tête est dur et on arrive inéluctablement à...

Des bugs imprévus

Des effets de bords surprenants

**Et plus globalement, le programme
devient dur à maintenir**

Pourtant...

Vous avez bien nommé vos variables, vos fonctions

Le code est bien indenté, etc

Parce-que vous avez uniquement
appliqué des optimisation locales

**Le problème, c'est que vous devez maintenir
un contexte global du programme en tête**

**Donc, vos optimisations doivent
changer d'échelle**

**Vous devez optimiser la structure de
votre programme**

Et donc faire une optimisation globale

**C'est exactement ici qu'intervient
l'architecture software**

**Vous cherchez à concevoir une
structure**

Facile à debugger

Facile à améliorer / étendre

Compliqué vs Complexe

**Admettons que vous vous lancez dans
une belle archi**

Beaucoup de classes, de fonctions

Vous essayez de séparer les choses

**Vous essayez d'anticiper les besoins
futurs**

Résultat des courses

Le code est presque pire qu'avant : encore plus dur à comprendre

Pourquoi ?



Votre code est un
plat de spaghettis

Robin Penea

Quelles sont les propriétés d'un code spaghetti ?

Propriétés code spaghetti

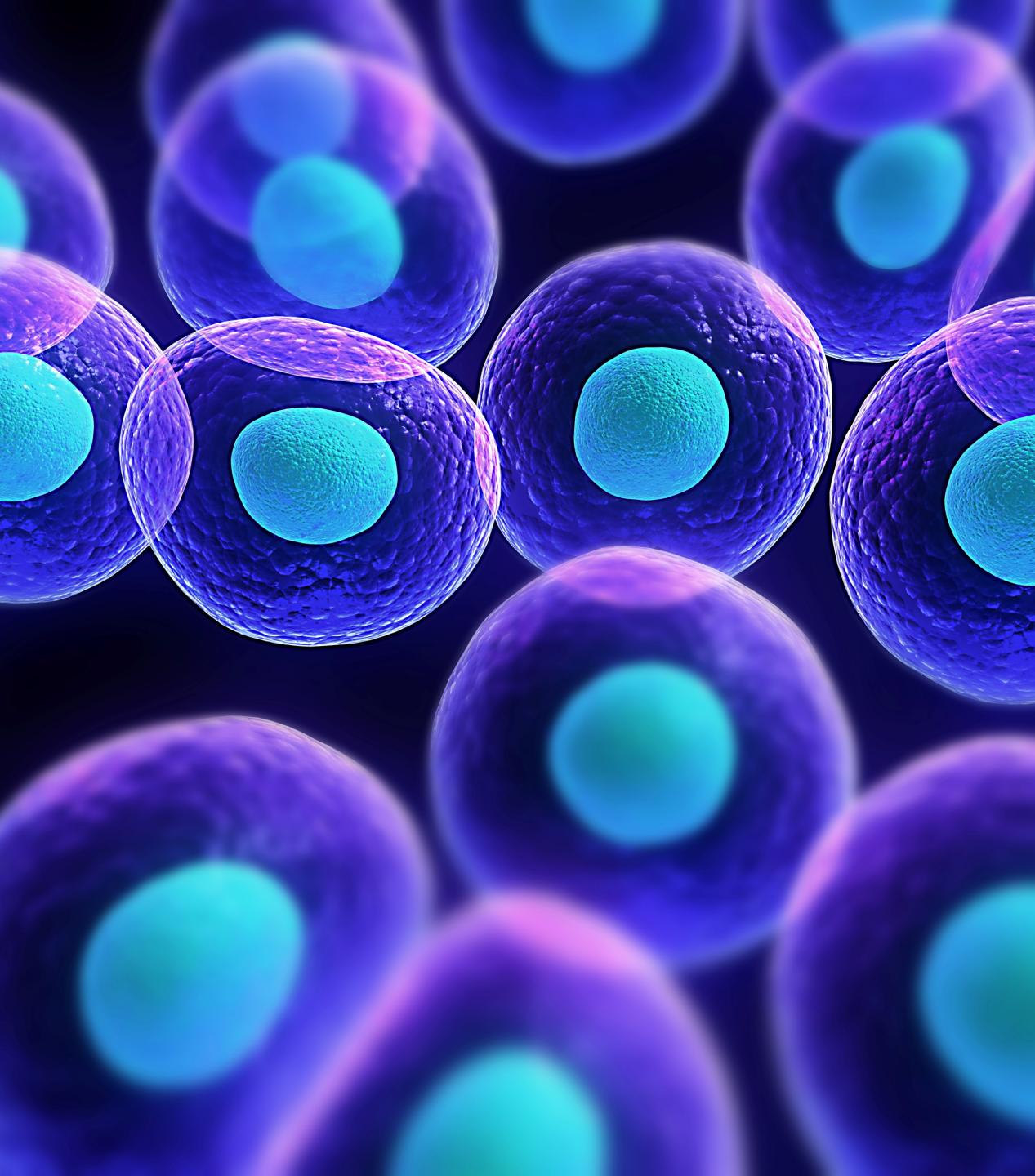
- On touche à un endroit, on ne sait pas ce qui sera **modifié par ricochet**
- Il y a pleins de manières de modifier quelque chose, chacune avec de **subtiles variations**

```
public boolean startElement(int, XMLAttrList) throws Exception {
...
    if (... && !fValidating && !fNamespacesEnabled) {
        return false;
    }
...
    if (contentSpecType == -1 && fValidating) {
...
    }
    if (... && elementIndex != -1) {
...
    }
    if (DEBUG_PRINT_ATTRIBUTES) {
...
    }

    if (fNamespacesEnabled) {
        fNamespacesScope.increaseDepth();
        if (attrIndex != -1) {
            int index = attrList.getFirstAttr(attrIndex);
            while (index != -1) {
...
                if (fStringPool.equalNames(...)) {
...
                } else {...}
            }
            index = attrList.getNextAttr(index);
        }
    }
    int prefix = fStringPool.getPrefixForQName(elementType);
    int elementURI;
    if (prefix == -1) {
...
        if (elementURI != -1) {
            fStringPool.setURIForQName(...);
        }
    } else {
...
        return contentSpecType == fCHILDRENSymbol;
    }
}
```

**En conclusion, c'est un code qui est
compliqué**

Ce qui est souhaitable



Un code
organisé comme
des cellules

Robin Penea

Code "cellules"

- Chaque entité à des **règles simples**, on touche à un endroit il n'y a que cette endroit qui est impacté
- Ces entités sont facilement **composables**, on peut les combiner pour former un système complexe

**La complexité n'est pas forcément
facile à comprendre au début**

**Mais une fois que l'on a compris, on
peut intervenir n'importe où dans le
système**

Comment en arrive-t-on à la situation du code compliqué ?

Souvent quand on développe sur un
nouveau projet,

on fait 2 choses à la fois

1. On découvre le domaine métier

- Réservation de taxis, jeu vidéo, IA, contrôle à distance, etc...
- Toutes ces **règles métiers** que vous découvrez

2. On modélise les règles métiers en code

- Une liste d'éléments X
- Il ne peut y avoir qu'un seul Y
- La valeur maximale est Z

**Donc vous codez les règles en même
temps que vous les apprenez**

Mécaniquement, votre compréhension
des règles va évoluer

**Mais la structure de votre code ne
suivra pas naturellement**

C'est donc à vous de faire du
refactoring

La question à chaque étape :

**Est-ce que la structure de mon code reflète
les règles du métier ?**

Question restreinte

**Est-ce que mes structures de données
reflètent le domaine métier ?**

Structure de données

- Une liste ou dictionnaire ?
- Plusieurs actions possibles ou une seule à la fois ?
- Quels sont les états possibles ?

“ Show me your flowcharts and conceal your tables, and I shall continue to be mystified. Show me your tables, and I won't usually need your flowcharts; they'll be obvious. ”

**Fred Brooks - Mythical Man-Month: Essays on Software Engineering
(1975)**

Pendant ce cours, nous allons aussi
bien écrire que réécrire du code

Mon objectif pour ce cours

**Plus de questions, et moins de solutions
toutes faites**

Explorer le sujet

- Livre [Clean Code - Robert C. Martin](#)
- Livre [Making Software - Andy Oram](#)
- Tech Talk [Livable Code - Sarah Mei](#)
- Tech Talk [Boundaries - Gary Bernhardt](#)
- Tech Talk [Clean Archi in Python - Brandon Rhodes](#)