POLYTECH
NICE SOPHIA

# Une aiguille dans une botte de code

## Rétro-Ingénierie, Maintenance et Évolution du Logiciel

Johann Mortara

13/01/2020

# Mon parcours

2014-2016 : PeiP à Polytech Nice Sophia

2016-2019 : Sciences Informatiques à Polytech Nice Sophia (Parcours AL en 5A)

2019 : Thèse au laboratoire I3S

# Mon parcours

2014-2016 : PeiP à Polytech Nice Sophia

2016-2019 : Sciences Informatiques à Polytech Nice Sophia (Parcours AL en 5A)

2019 : Thèse au laboratoire I3S

**Mon sujet :**

**Identification et visualisation de variabilité au sein de grands systèmes hautement variables**

# Mon parcours

2014-2016 : PeiP à Polytech Nice Sophia

2016-2019 : Sciences Informatiques à Polytech Nice Sophia (Parcours AL en 5A)

2019 : Thèse au laboratoire I3S

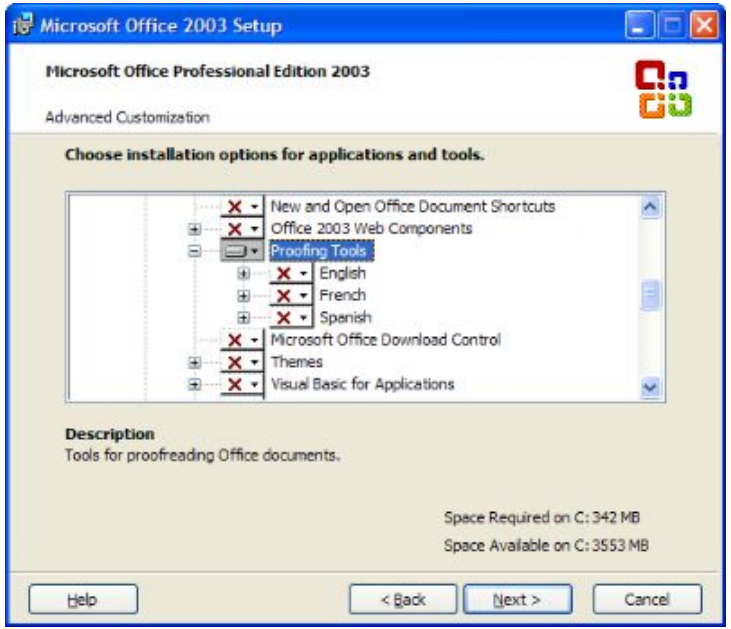
**Mon sujet :**

**Identification et visualisation de variabilité au sein de grands systèmes hautement variables**

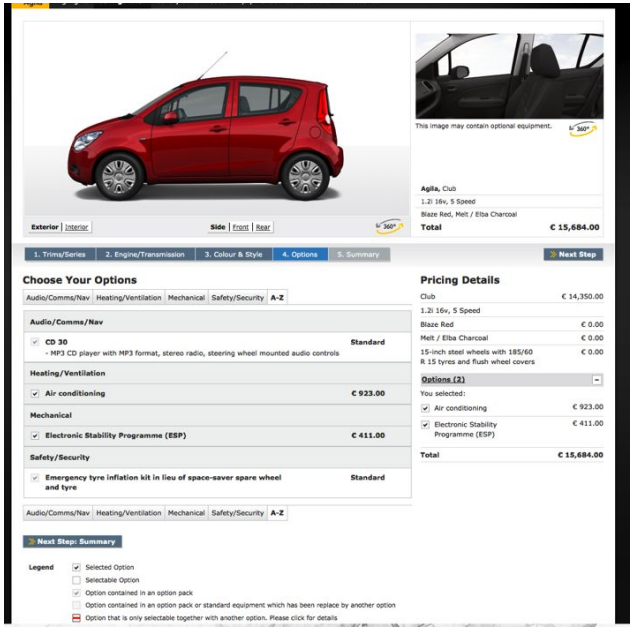
**Pardon ?**

# Vari-quoi ?

**Variabilité** : Ensemble des mécanismes permettant de paramétrer et configurer un système.

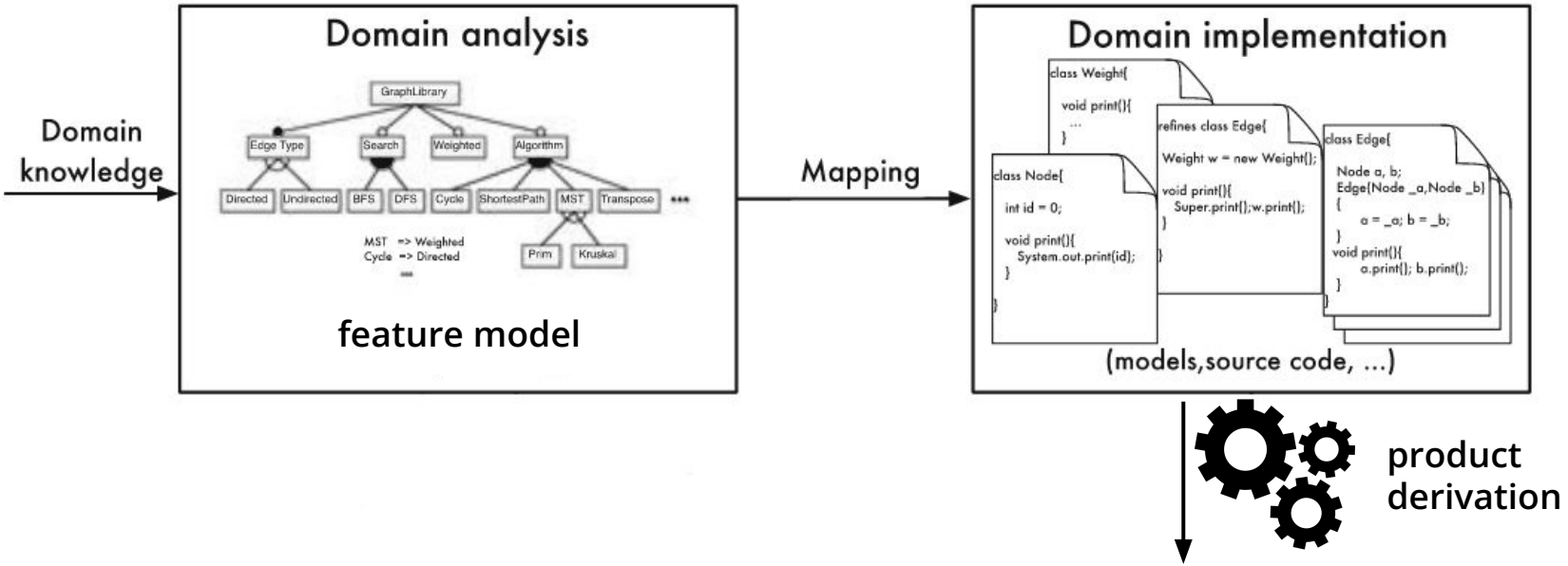



© Microsoft



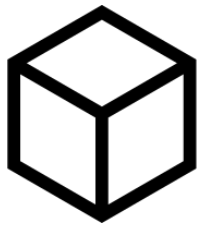

© Opel

# Les Lignes de Produits Logiciels (SPL)



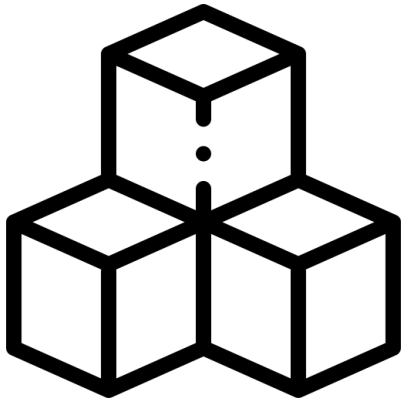

Source : http://stg-tud.github.io/sedc/Lecture/ws16-17/6-SPL.pdf

# Il était une fois...



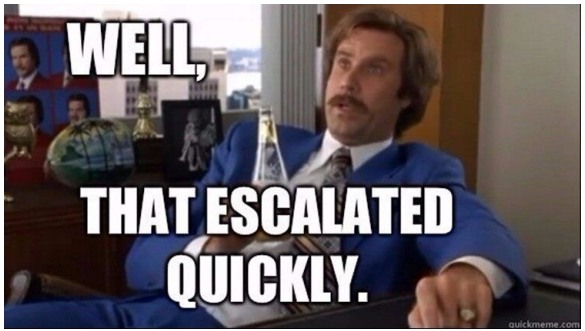
Votre petit projet

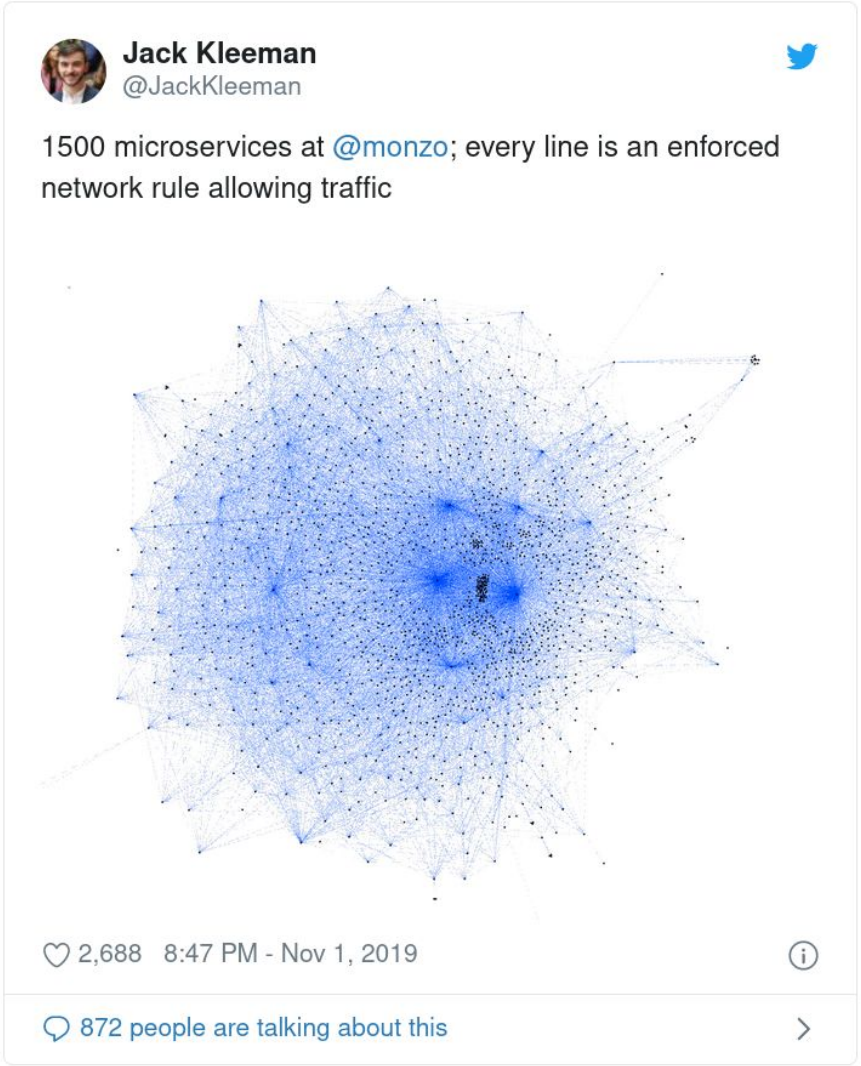
# *“Ah tiens et si on faisait ça aussi ?”*



Votre *“petit”* projet

# Et là, c'est le drame !




Source : https://twitter.com/JackKleeman/status/1190354757308862468?s=09

# Que faire pour revenir à une SPL ?

# Que faire pour revenir à une SPL ?

## On reprend tout à zéro :

- analyse du domaine
- écriture du code de chaque fonctionnalité
- ...

Feature model ⟶ Implémentation

# Que faire pour revenir à une SPL ?

On reprend tout à zéro :

- analyse du domaine
- écriture du code de chaque fonctionnalité
- ...

Feature model ⟶ Implémentation

On extrait le code des fonctionnalités présentes pour reconstruire un *feature model*.

Feature model ⟵ Implémentation

# Que faire pour revenir à une SPL ?

On reprend tout à zéro :

- analyse du domaine
- écriture du code de chaque fonctionnalité
- ...

Feature model ⟶ Implémentation

On extrait le code des fonctionnalités présentes pour reconstruire un *feature model*.

Feature model ⟵ Implémentation

⇒ **rétro-ingénierie**

# Et la visualisation dans tout ça ?

Mapping automatisé difficile à obtenir

Mais savoir où la variabilité est située permet d'effectuer un mapping manuel.

⟶ besoin de mettre en exergue ces zones dans de grandes bases de code

⟶ utilisation d'une visualisation

Premier objectif : aider un développeur à savoir où est la variabilité du projet et assister sa **rétro-ingénierie**

# *“Par où on commence, chef ?”*

**Identification et visualisation de variabilité au sein de grands systèmes hautement variables**

**Identification et visualisation de variabilité au sein de grands systèmes hautement variables**

**Identification et visualisation de variabilité au sein de grands systèmes hautement variables**

# En avant pour l'aventure !

Image by Mila Kusmenko from Pixabay

# Intuition

Objectif : identifier des implémentations de variabilité

Question : Comment s'implémente la variabilité ?

Exemples communs :

- paramètres
- propriétés

Exemples spécifiques aux systèmes orientés objet :

- héritage / implémentation d'interfaces
- surcharge de méthodes
- surcharge de constructeurs
- patrons de conception

# Intuition

Objectif : identifier des implémentations de variabilité

Question : Comment s'implémente la variabilité ?

**Personne n'est allé là !**

Exemples communs :

- paramètres
- propriétés

**Exemples spécifiques aux systèmes orientés objet :**

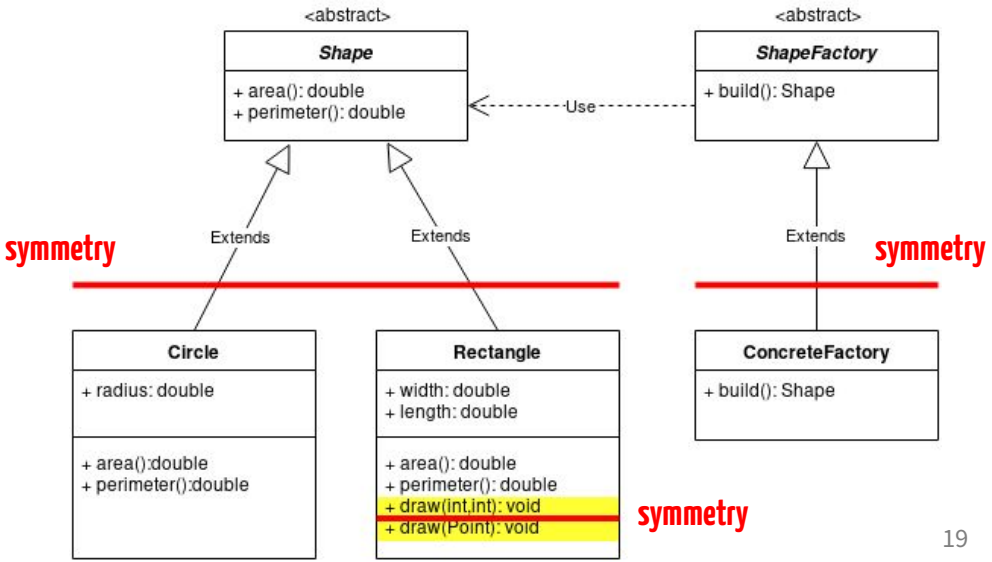
- **héritage / implémentation d'interfaces**
- **surcharge de méthodes**
- **surcharge de constructeurs**
- **patrons de conception**

# Intuition

- Présence de **symétries dans des bases de code orientées objets** [Coplien2019] inspiré de la théorie des centres de Christopher Alexander [Alexander2002].

- Ces symétries sont présentes dans les **mécanismes d'implémentation de la variabilité**.

**⇒ Utilisation des symétries pour détecter les implémentations de variabilité ?**



# Vérification de l'intuition

**On commence petit !**

- Prendre un petit morceau de code qui a les propriétés recherchés,
- Vérifier qu'on retrouve bien ces propriétés.

# Vérification de l'intuition

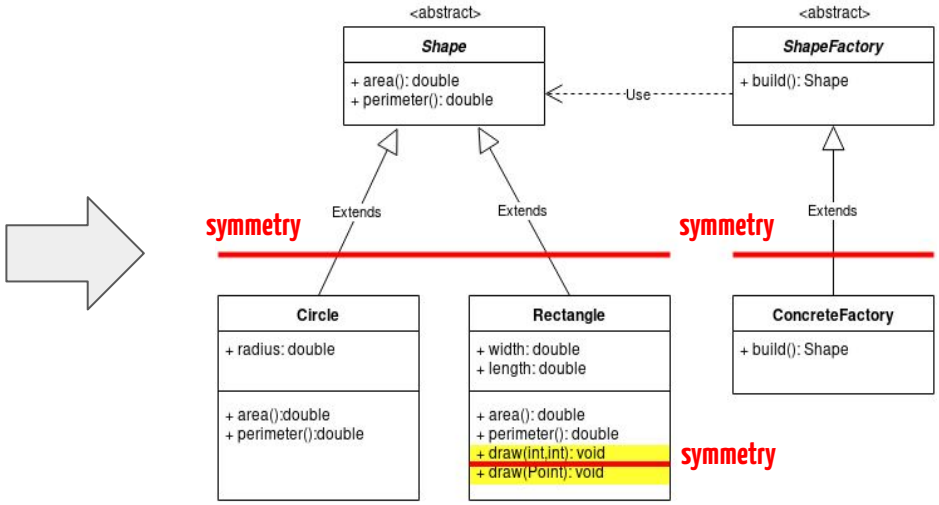
## Code source

```java
public abstract class Shape {
  public abstract double area();
  public abstract double perimeter(); /*...*/
}
```

```java
public class Rectangle extends Shape {
  private final double width, length;
  // Constructor omitted
  public double area() {
    return width * length;
  }
  public double perimeter() {
    return 2 * (width + length);
  }
  public void draw(int x, int y) {
  // rectangle at (x, y, width, length)
  }
  public void draw(Point p) { // Point defined
  // rectangle at (p.x, p.y, width, length)
  }
}
```

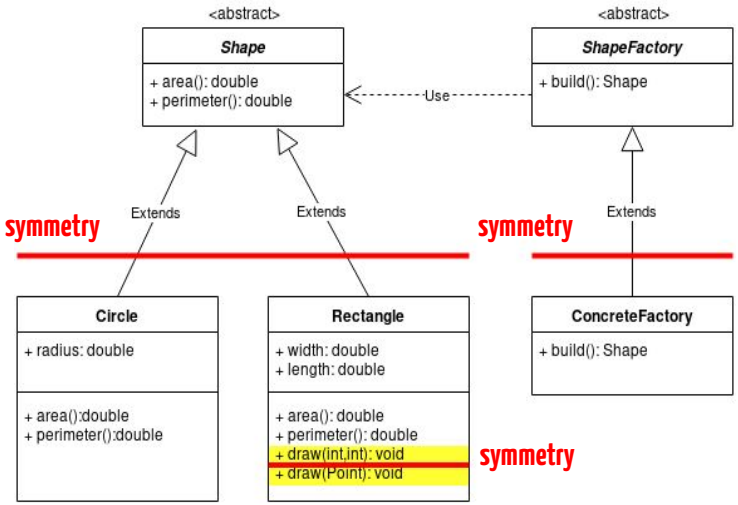
```java
public class Circle extends Shape {
  private final double radius;
  // Constructor omitted
  public double area() {
    return Math.PI * Math.pow(radius, 2);
  }
  public double perimeter() {
    return 2 * Math.PI * radius;
  }
}
```

## Identification des symétries

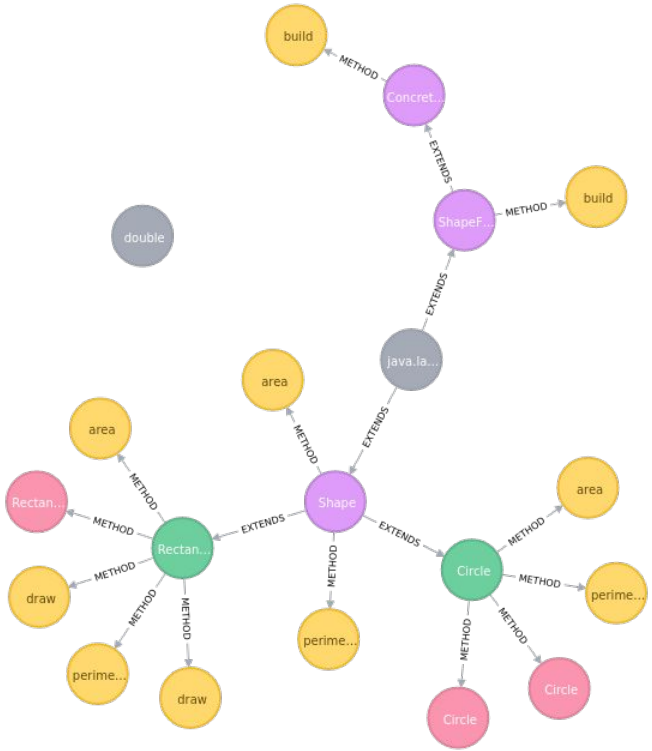


# Vérification de l'intuition

## Identification des symétries




## Points de variation identifiés



# Niveau 1 : atteint

Identification et visualisation de variabilité au sein de grands systèmes hautement variables

Identification et visualisation de variabilité au sein de ~~grands~~ systèmes ~~hautement~~ variables

Identification ~~et visualisation~~ de variabilité au sein de ~~grands~~ systèmes ~~hautement~~ variables

# Niveau 2 : atteint (merci Neo4j)

Identification et visualisation de variabilité au sein de grands systèmes hautement variables

Identification et visualisation de variabilité au sein de ~~grands~~ systèmes ~~hautement~~ variables

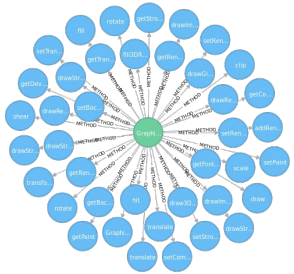
Identification ~~et visualisation~~ de variabilité au sein de ~~grands~~ systèmes ~~hautement~~ variables

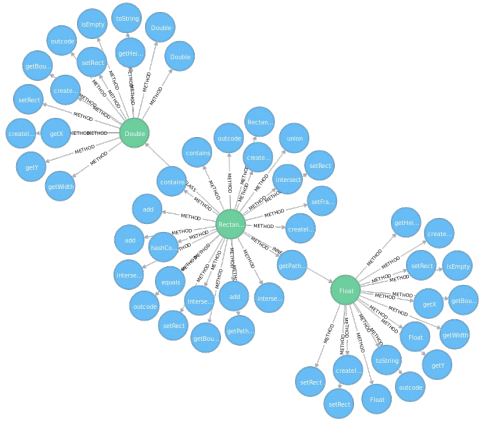
# Et maintenant ?

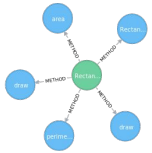
## On essaie sur d'autres petits modèles !

**One step at a time...**

# 1 nœud vert = 1 classe





# Niveau 2 : ~~atteint (merci Neo4j)~~ *here we go again…*

Identification et visualisation de variabilité au sein de grands systèmes hautement variables

Identification et visualisation de variabilité au sein de ~~grands~~ systèmes ~~hautement~~ variables

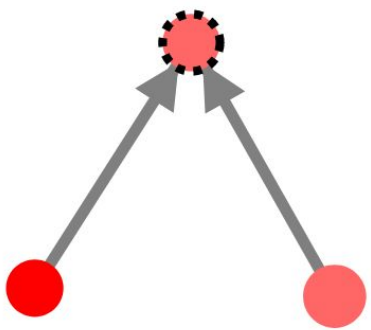
Identification ~~et visualisation~~ de variabilité au sein de ~~grands~~ systèmes ~~hautement~~ variables

# Refonte de la visualisation

Présence de nœuds pour des classes et des méthodes

⟶ confusion

Utilisation de principes de la visualisation




- 1 nœud = 1 classe
- propriétés du nœud = propriétés de la classe

# Niveau 2 : atteint *(enfin !)*

**Identification et visualisation de variabilité au sein de grands systèmes hautement variables**

**Identification et visualisation de variabilité au sein de ~~grands~~ systèmes ~~hautement~~ variables**

**Identification ~~et visualisation~~ de variabilité au sein de ~~grands~~ systèmes ~~hautement~~ variables**

# Et maintenant ?

**On essaie sur de vrais projets !**

**Lesquels ?**

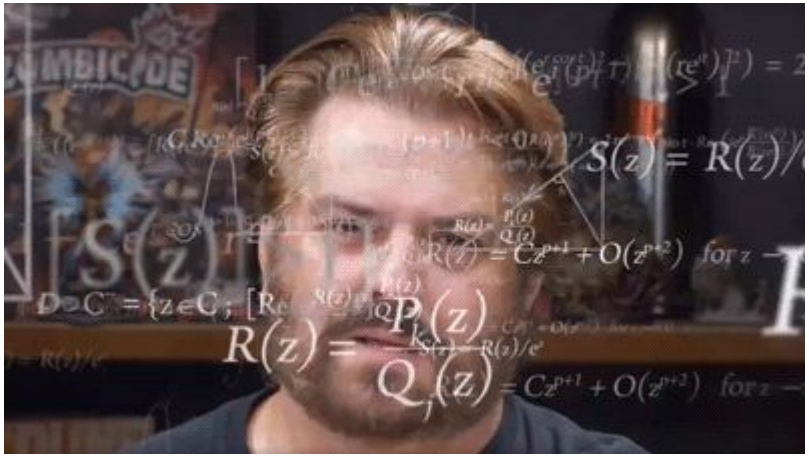
# Sélection des projets

Objectif : trouver des bons projets à analyser.

Qu'est-ce qu'un "bon" projet ?

- un projet qui nous permet de montrer que notre intuition est la bonne
- en somme, un projet variable !

Mais comment le sait-on ?



# Indices

- Métier du projet : est-ce que le métier comporte de la variabilité ou non ?
- La documentation : présence d'une liste de fonctionnalités ?
- L'intuition

# JFreeChart

Bibliothèque permettant de tracer différents types de graphiques

# JFreeChart

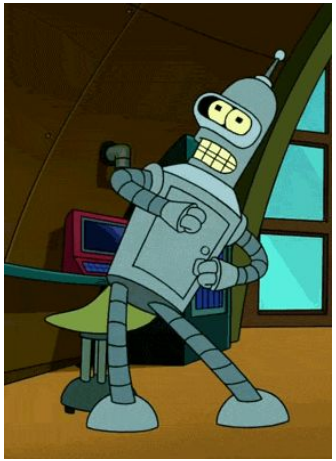
Bibliothèque permettant de tracer **différents types de graphiques**

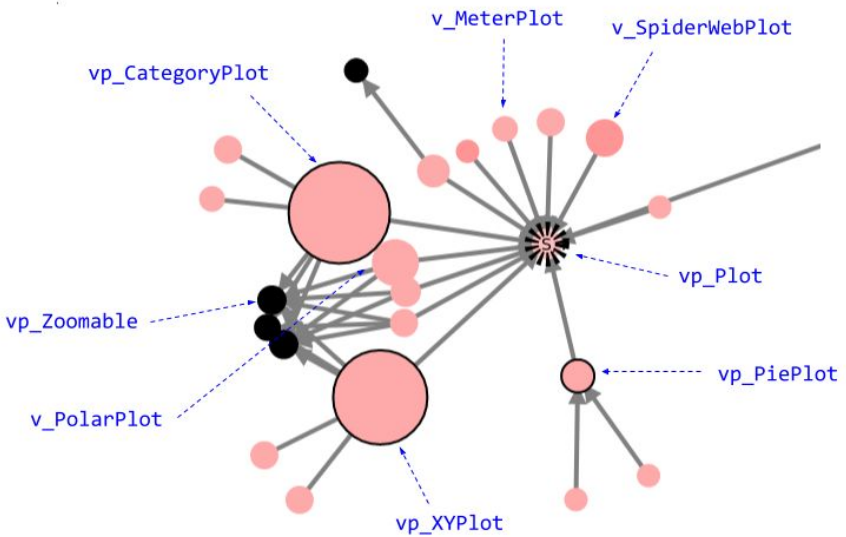
**variabilité ?**

# JFreeChart

variabilité !

Bibliothèque permettant de tracer **différents types de graphiques**





# JHipster

Outil de configuration de projets à partir d'un choix de pile technologique

Construit comme une ligne de produit

# JHipster

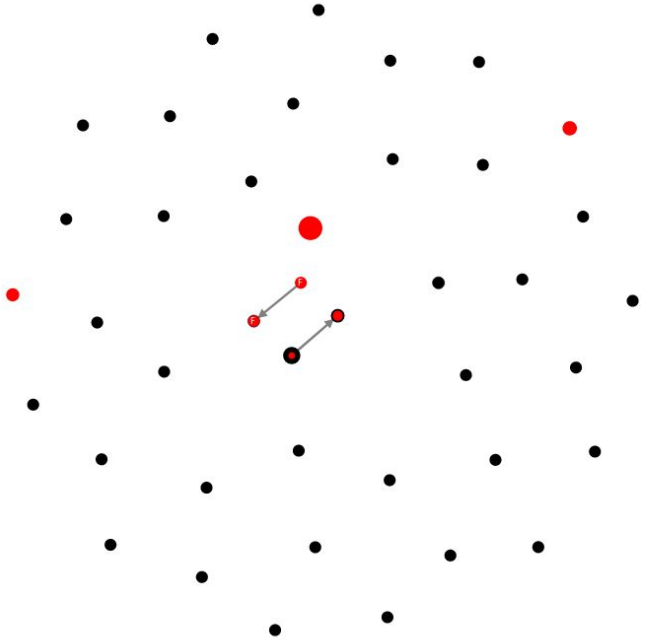
Outil de configuration de projets à partir d'un choix de pile technologique

**Construit comme une ligne de produit** ⟵ **variabilité ?**

# JHipster

Outil de configuration de projets à partir d'un choix de pile technologique

Construit comme une ligne de produit ← ~~variabilité~~



# À la recherche de la variabilité perdue !

Rétro-ingénierie à la main pour comprendre pourquoi ça n'a pas marché dans ce cas.

JHipster projet multi-langages et sur plusieurs dépôts ⟶ variabilité éparpillée

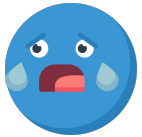
## Bilan néanmoins positif !

Prouve qu'on ne voit rien quand il n'y a rien à voir.

⟶ Pas de faux-positifs

# Niveau 3 : presque atteint

Identification et visualisation de variabilité au sein de **grands** systèmes hautement variables

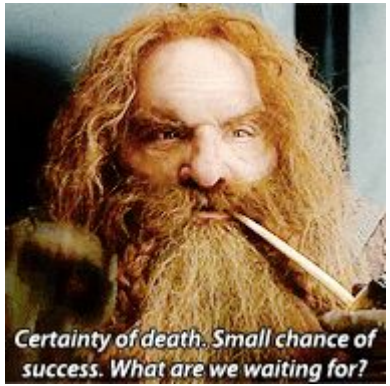


Identification et visualisation de variabilité au sein de ~~grands~~ systèmes ~~hautement~~ variables

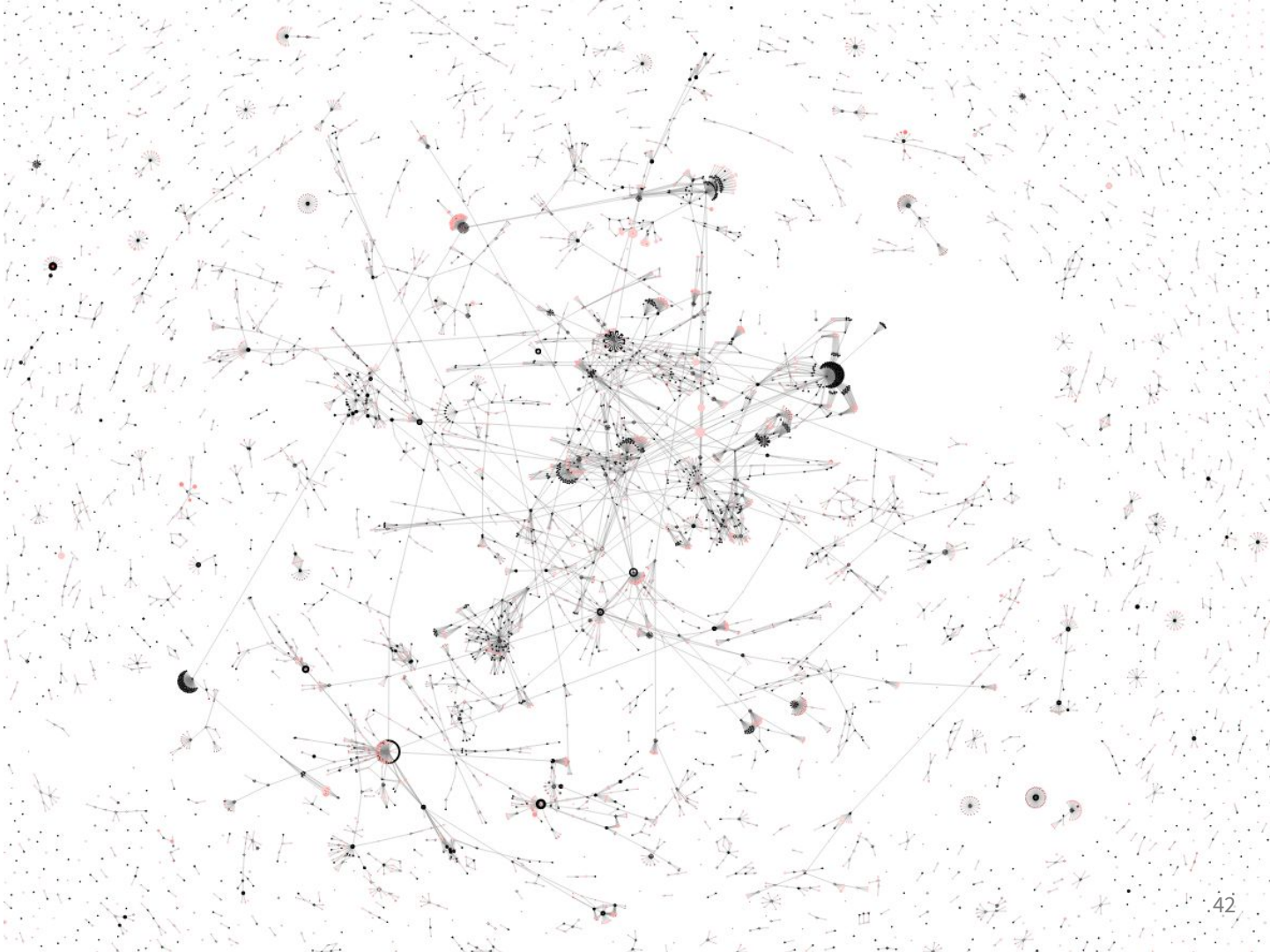
Identification ~~et visualisation~~ de variabilité au sein de ~~grands~~ systèmes ~~hautement~~ variables

# Apache NetBeans

- 4.5 M LoC
- Plus gros projet Java de la fondation Apache
- Présence de variabilité ? *Who knows ?*




# It works!



# Ce que cette aventure m'a apporté

Image par kropekk_pl de Pixabay

1. Suivre les pistes des intuitions ; on peut être (agréablement) surpris du résultat.

2. Ne pas trop en faire dès le départ ; une hypothèse, bien que simple, peut être suffisante.

3. Un échec n'en est jamais vraiment un.

# Que faire si ça ne marche pas ?

Réflexe : ~~pleurer~~ comprendre ce qu'il s'est passé.

⟶ on plonge dans le code pour chercher "à la main" ce que l'outil n'a pas trouvé.

## Ce n'est jamais une perte de temps !

On trouve quelque chose ?
⟶ On corrige l'outil.

On ne trouve rien ?
⟶ C'est qu'il n'y a rien à voir (pas de faux-positif) ⟶ tout va bien !

# Merci !