

# Une aiguille dans une botte de code

**Rétro-Ingénierie, Maintenance et Évolution  
du Logiciel**

Johann Mortara

12/01/2022

# Mon parcours

2014-2016 : PeiP à Polytech Nice Sophia

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

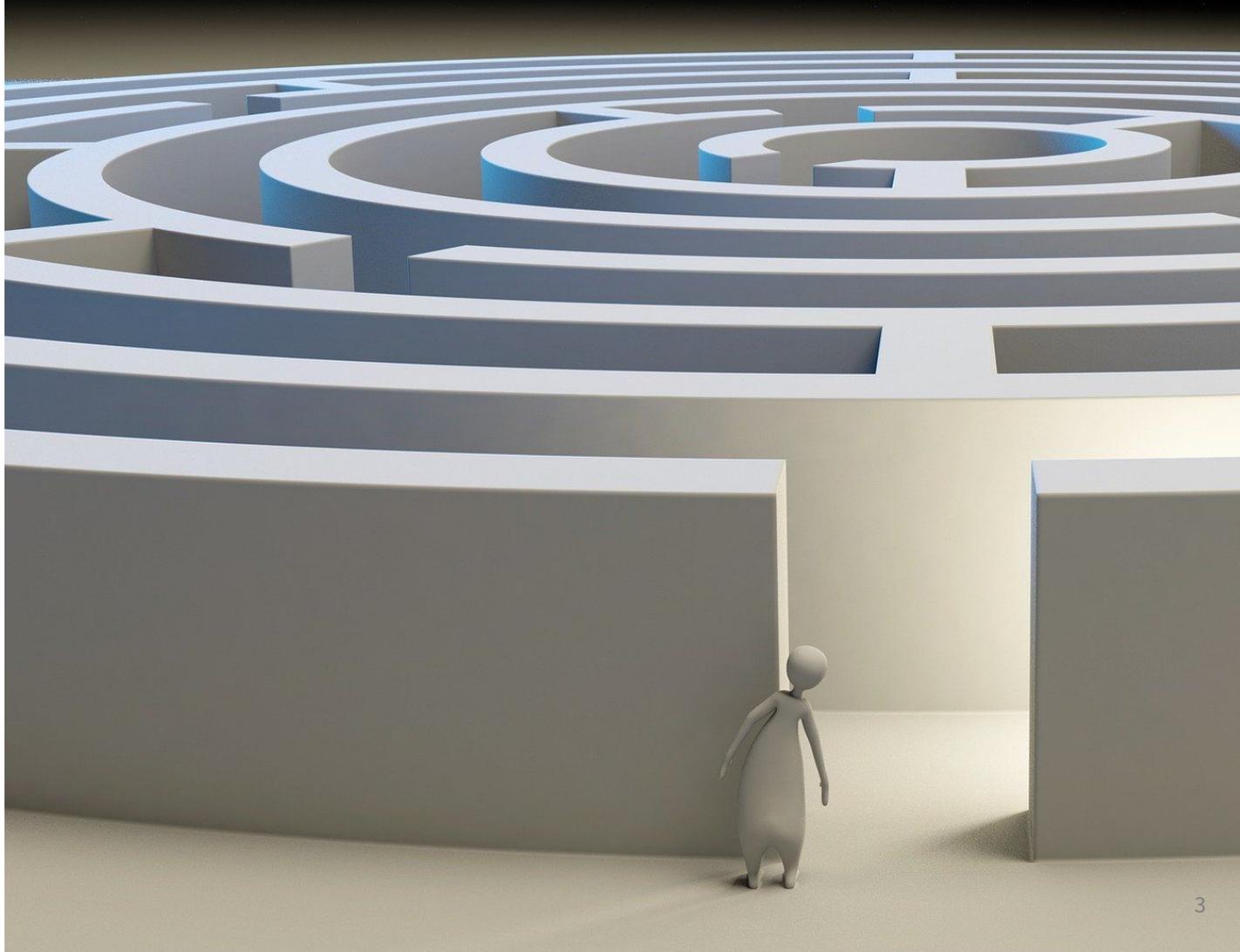
2019 : Thèse au laboratoire I3S (directeur : Philippe Collet)

**Identification, visualisation et gestion de variabilité au sein de  
grands systèmes orientés objets hautement variables**

**Pardon ?**

# 1.

## Quel est le problème ?



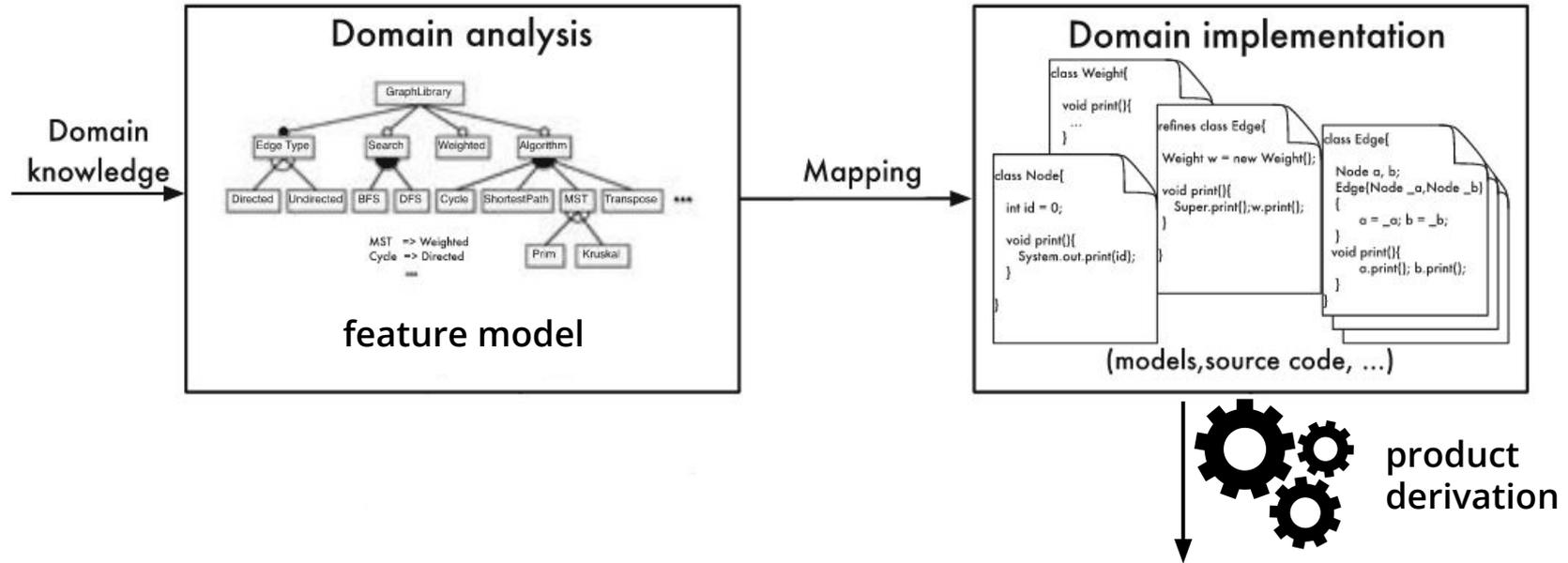
# Vari-quoi ?

**Variabilité** : Capacité d'un logiciel à être étendu ou configuré pour un contexte précis [Svahnberg2005]

Exemples :

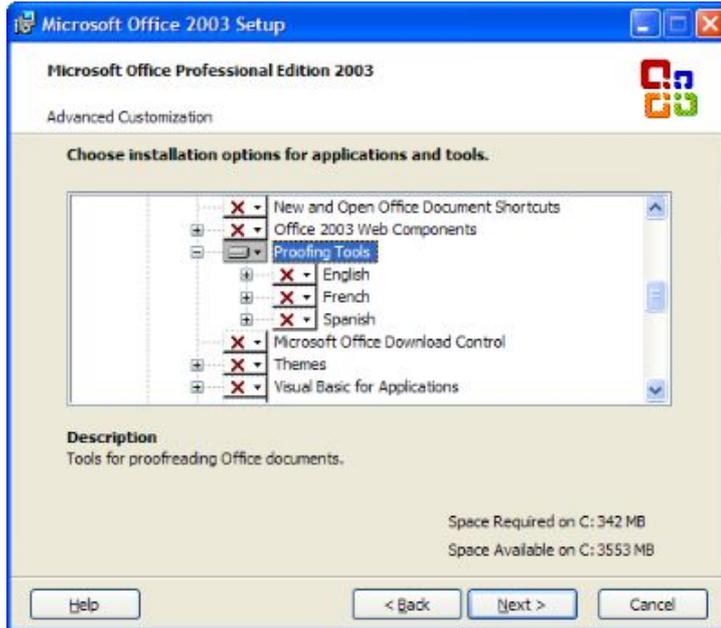
- support de plusieurs langues
- différentes connexions aux serveurs de mails d'un client mail (POP3, SMTP...)
- extensions des fonctionnalités avec des plugins
- ...

# Les Lignes de Produits Logiciels (SPL)



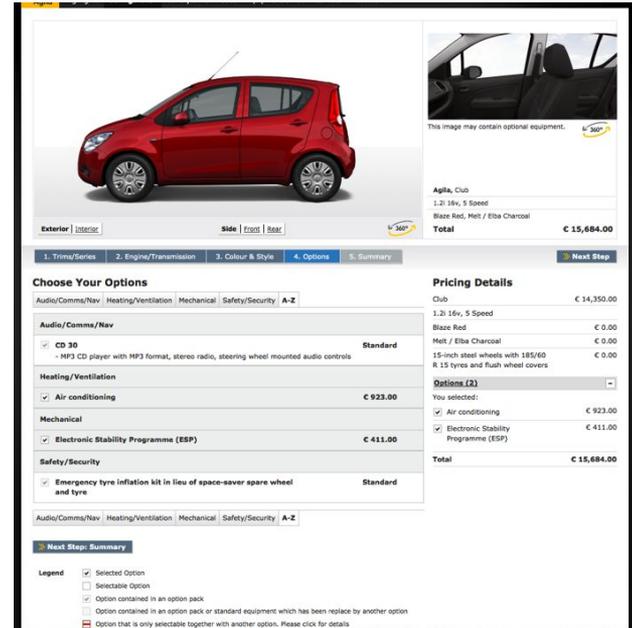
# Exemples de lignes de produits

Des produits logiciels...



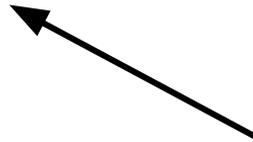
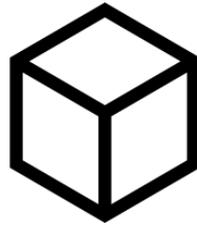
© Microsoft

... mais pas seulement !



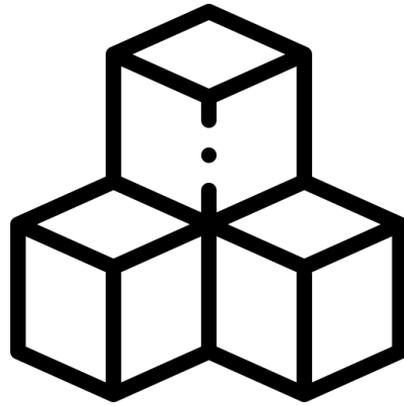
© Opel

# Il était une fois...



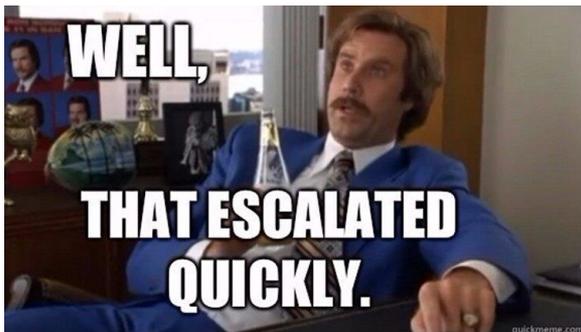
Votre petit projet

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



Votre *“petit”* projet

# Et là, c'est le drame !

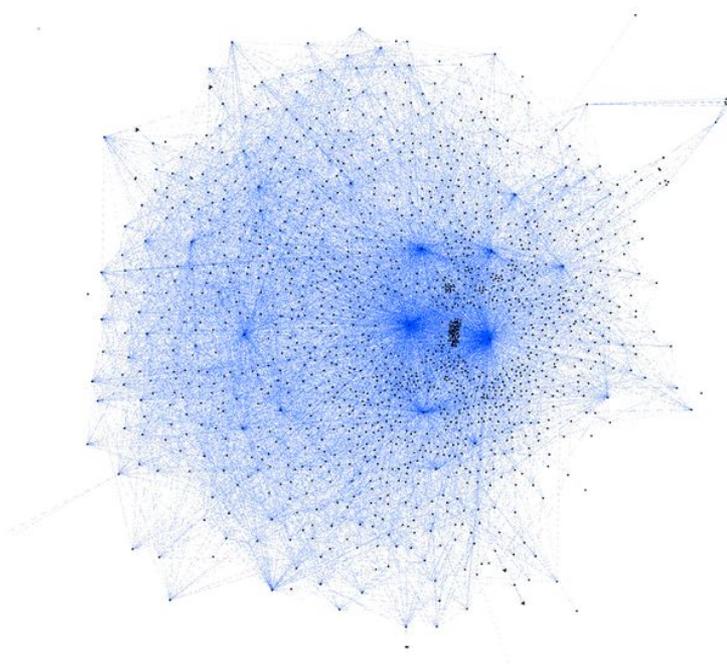


**Jack Kleeman**

@JackKleeman



1500 microservices at @monzo; every line is an enforced network rule allowing traffic



2,688 8:47 PM - Nov 1, 2019



872 people are talking about this



# Quelques systèmes hautement variables



16.000 options gérées  
dans 25M LoC  
[Acher2018]



24.000 différentes  
plateformes en 2015  
[Open2015]



2.000+ options générant des variantes  
pour différentes plateformes, niveaux de  
sécurité... [Acher2018]

⇒ pas de modèle formel ⇒ pas de SPL !

# Que faire pour revenir à une SPL ?

On reprend tout à zéro :

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

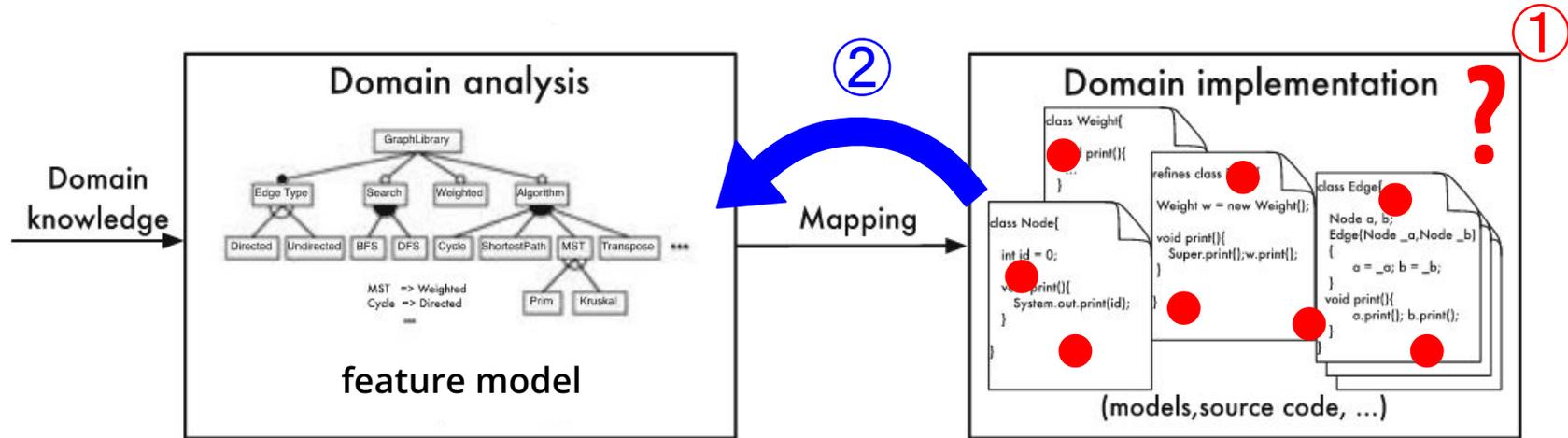
Feature model → Implémentation

On identifie les implémentations de variabilité présentes pour reconstruire un *feature model* (i.e. features + contraintes entre elles).

Feature model ← Implémentation

⇒ **rétro-ingénierie**

# On identifie les implémentations de variabilité présentes pour les relier à un feature model



# 2.

## Comment s'y prend-on ?

Découpons le  
problème !



Source : Image by [cpenzin](#) from [Pixabay](#)

## Identifier les implémentations de variabilité présentes pour les relier à un feature model

Les implémentations de variabilité sont **enfouies dans le code**, il nous faut donc **les identifier**.

Données à notre disposition : code source du système étudié

⇒ 1<sup>ère</sup> sous-question : Comment identifier des implémentations de variabilité d'un système en ayant pour seules données son code source ?

## Identifier les implémentations de variabilité présentes pour les relier à un feature model

Les implémentations de variabilité implémentent la variabilité (*c'est fou non ?*)

*i.e.* ces implémentations peuvent être reliées à des **fonctionnalités du domaine** (liées au métier), donc à des **fonctionnalités d'un feature model**

→ Besoin de s'assurer que les implémentations identifiées sont correctes !

⇒ 2<sup>e</sup> sous-question : Est-ce que les implémentations de variabilité identifiées correspondent vraiment à de la variabilité ?

# Identifier les implémentations de variabilité présentes pour les relier à un feature model

## Oui, mais pourquoi faire ?

Zones fortement variables représentent du code complexe

→ points d'intérêt du code qui doivent être connus des architectes / développeurs pour assurer leur qualité

⇒ 3<sup>e</sup> sous-question : Comment indiquer à un utilisateur les zones de forte densité d'un projet ?

# Nos sous-questions

1. **Comment identifier des implémentations de variabilité d'un système en ayant pour seules données son code source ?**
2. **Est-ce que les implémentations de variabilité identifiées correspondent vraiment à de la variabilité ?**
3. **Comment indiquer à un utilisateur les zones de forte densité d'un projet ?**

# 3.1

## À la recherche de la variabilité perdue



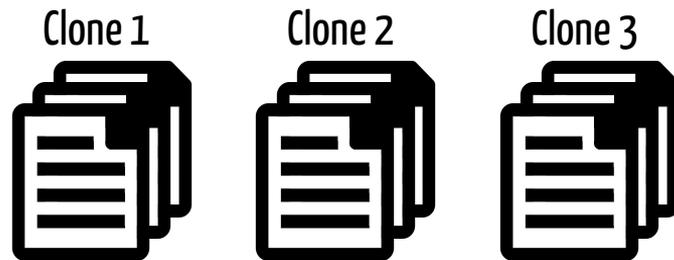
Source : Image by [Mila Kusmenko](#)  
from [Pixabay](#)

# Étape 1 : que sait-on faire ?

Rechercher dans l'état de l'art (articles scientifiques...) si des techniques existent déjà

## Technique 1 : comparaison de clones

Comparaison entre les clones et lien avec les fonctionnalités [Assunção2017]



## Peut-on s'en servir ?

Non, car la variabilité d'un système OO est gérée la plupart du temps en une unique base de code.

# Étape 1 : que sait-on faire ?

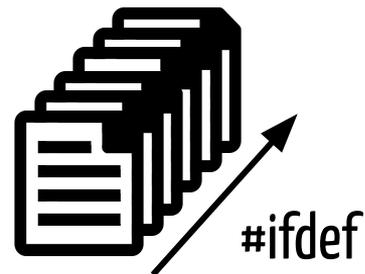
Rechercher dans l'état de l'art (articles scientifiques...) si des techniques existent déjà

## Technique 2 : directives de préprocesseurs / annotations dans une unique base de code

Identifier les directives ou annotations et les relier à des features [Liebig2010, Hunsen2016]

## Peut-on s'en servir ?

Pas toujours, car la majorité des systèmes OO n'ont pas ces informations.

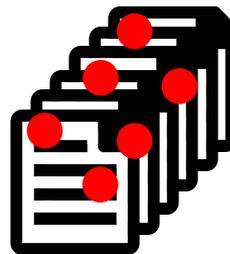


# Étape 1 : que sait-on faire ?

Rechercher dans l'état de l'art (articles scientifiques...) si des techniques existent déjà

## Notre contexte :

- Une unique base de code
- Pas d'annotations / directives



## Comment faire ?

Besoin de trouver une technique se reposant uniquement sur le code du système.

# Étape 2 : que cherche-t-on exactement ?

**Question : Comment les systèmes OO implémentent leur variabilité ?**

Exemples :

- paramètres
- propriétés
- annotations (ex : #ifdefs)

**Exemples spécifiques aux systèmes orientés objet :**

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

## Étape 2 : que cherche-t-on exactement ?

**Question : Comment les systèmes OO implémentent leur variabilité ?**

**On observe manuellement !**

# Points de variation et variantes

```
1 | public abstract class Shape {
2 |     public abstract double area();
3 |     public abstract double perimeter(); /*...*/
4 | }
```

vp\_shape

```
5 | public class Circle extends Shape {
6 |     private final double radius;
7 |     // Constructor omitted
8 |     public double area() {
9 |         return Math.PI * Math.pow(radius, 2);
10 |    }
11 |    public double perimeter() {
12 |        return 2 * Math.PI * radius;
13 |    }
14 | }
```

v\_circle

```
15 | public class Rectangle extends Shape {
16 |     private final double width, length;
17 |     // Constructor omitted
18 |     public double area() {
19 |         return width * length;
20 |    }
21 |     public double perimeter() {
22 |         return 2 * (width + length);
23 |    }
24 |     public void draw(int x, int y) {
25 |         // rectangle at (x, y, width, length)
26 |    }
27 |     public void draw(Point p) {
28 |         // rectangle at (p.x, p.y, width, length)
29 |    }
30 | }
```

v\_rectangle

vp\_draw

# Étape 2 : que cherche-t-on exactement ?

**Question : Comment les systèmes OO implémentent leur variabilité ?**

**Intuition d'après les observations : utilisation des mécanismes OO**

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

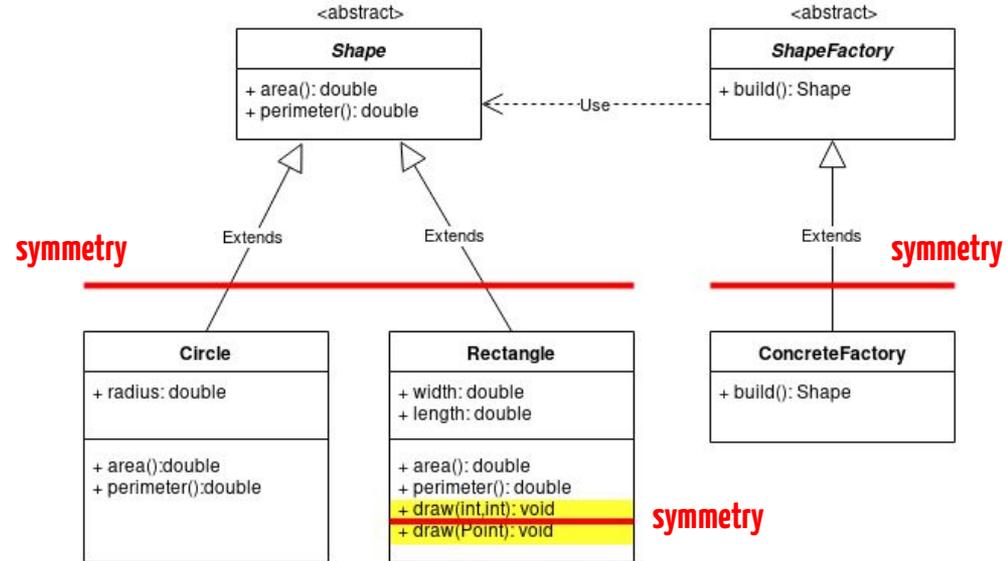
**⇒ Nouvelle question : Comment identifier ces implémentations de variabilité ?**

# 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 ?**

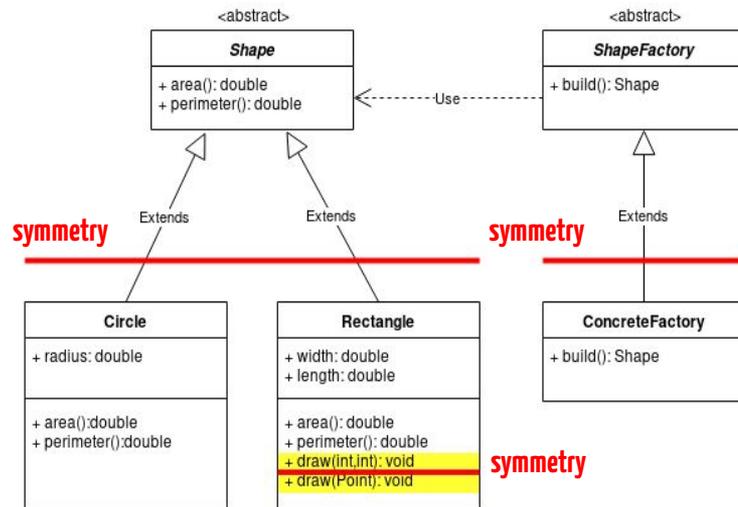
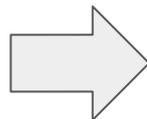


# Intuition

Code source

Identification des symétries

```
1 public abstract class Shape {
2     public abstract double area();
3     public abstract double perimeter(); /*...*/
4 }
5
6 public class Rectangle extends Shape {
7     private final double width, length;
8     // Constructor omitted
9     public double area() {
10         return width * length;
11     }
12     public double perimeter() {
13         return 2 * (width + length);
14     }
15     public void draw(int x, int y) {
16         // rectangle at (x, y, width, length)
17     }
18     public void draw(Point p) { // Point defined
19         // rectangle at (p.x, p.y, width, length)
20     }
21 }
22
23 public class Circle extends Shape {
24     private final double radius;
25     // Constructor omitted
26     public double area() {
27         return Math.PI * Math.pow(radius, 2);
28     }
29     public double perimeter() {
30         return 2 * Math.PI * radius;
31     }
32 }
```



# Étape 3 : vérification de l'intuition

**On commence petit !**

1. Prendre un petit projet qui a les propriétés recherchés,
2. Vérifier qu'on retrouve bien ces propriétés.

# Choix d'un projet

## Caractéristiques recherchées :

- implémenté dans un langage orienté objet (ex : Java)
- une seule base de code
- intuition de la présence de variabilité

## Candidat choisi : Java AWT

- Partie graphique du JRE
- Permet de créer des environnements graphiques avec différents types de composants ← **variabilité**



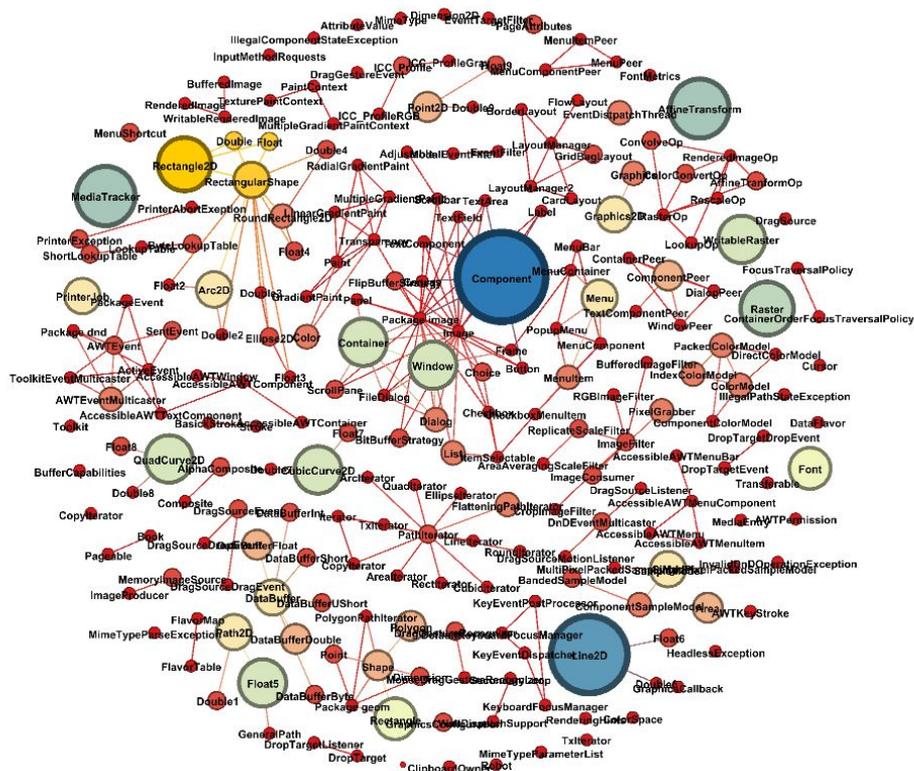
# Nos sous-questions

1. Comment identifier des implémentations de variabilité d'un système en ayant pour seules données son code source ? **Sur un projet, la densité de symétries semble être un moyen viable pour identifier des points de variation et leurs variantes.**
2. Est-ce que les implémentations de variabilité identifiées correspondent vraiment à de la variabilité ? **Bonne question...**
3. Comment indiquer à un utilisateur les zones de forte densité d'un projet ? **Sur un projet, un graphe semble approprié.**

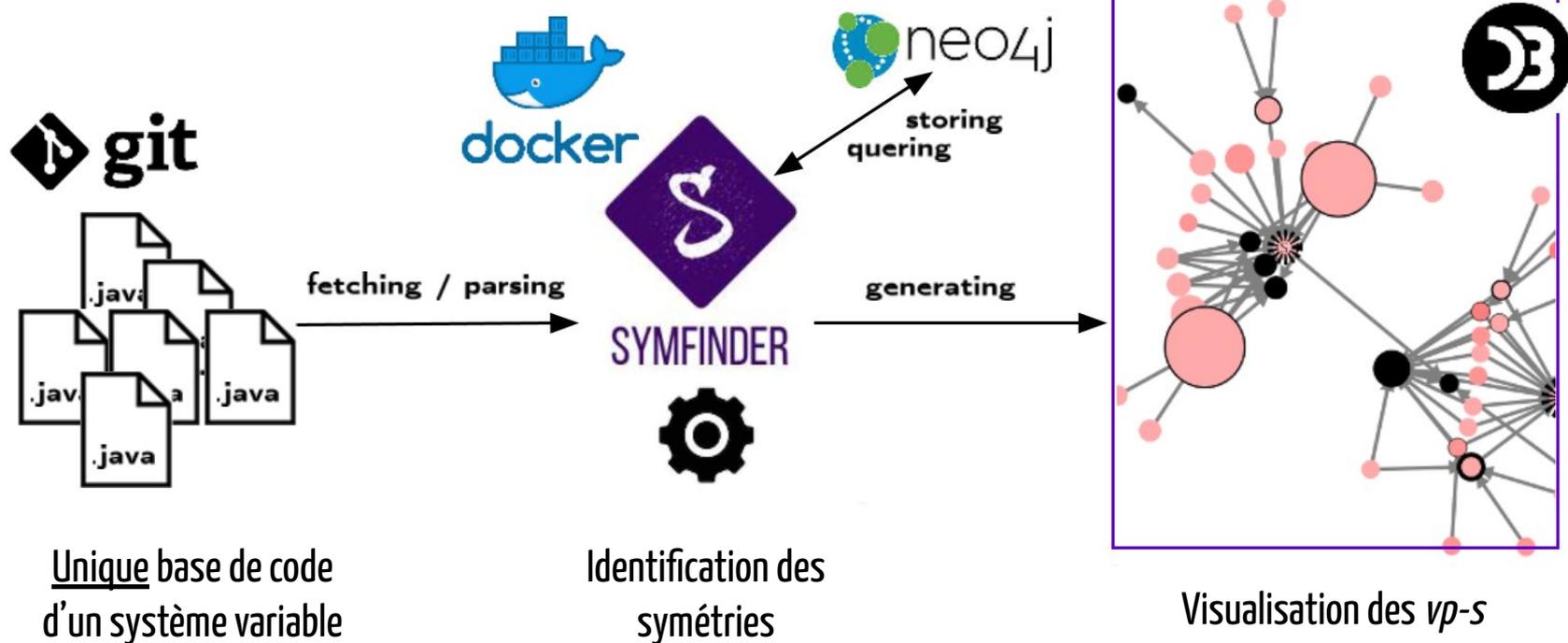
# Premières expérimentations

- Identification **manuelle** des points de variation de variation
- Première visualisation de la variabilité

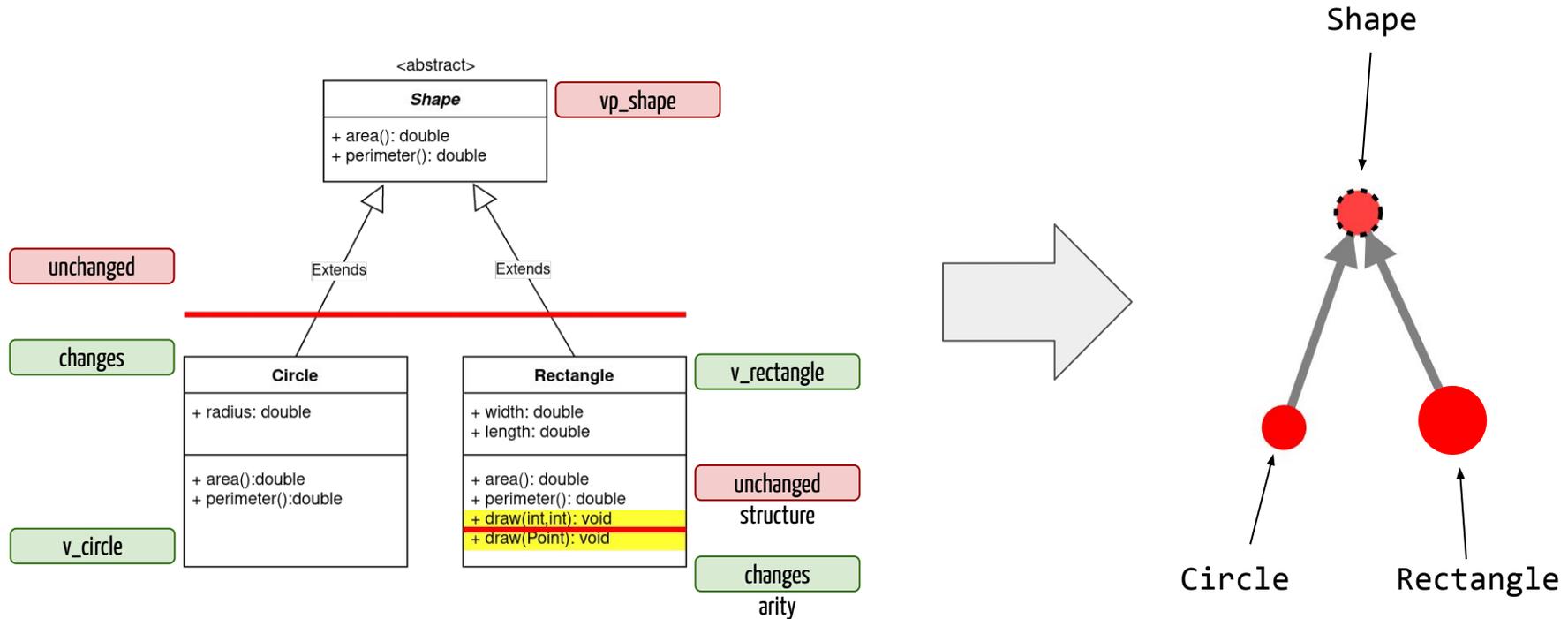
⇒ **besoin d'une solution automatisée pour valider l'intuition sur d'autres projets**



# symfinder



# Visualisation d'un petit exemple



# Vérification de l'approche symfinder

On exécute symfinder sur des projets qui ont (potentiellement) les caractéristiques recherchées !

# JFreeChart

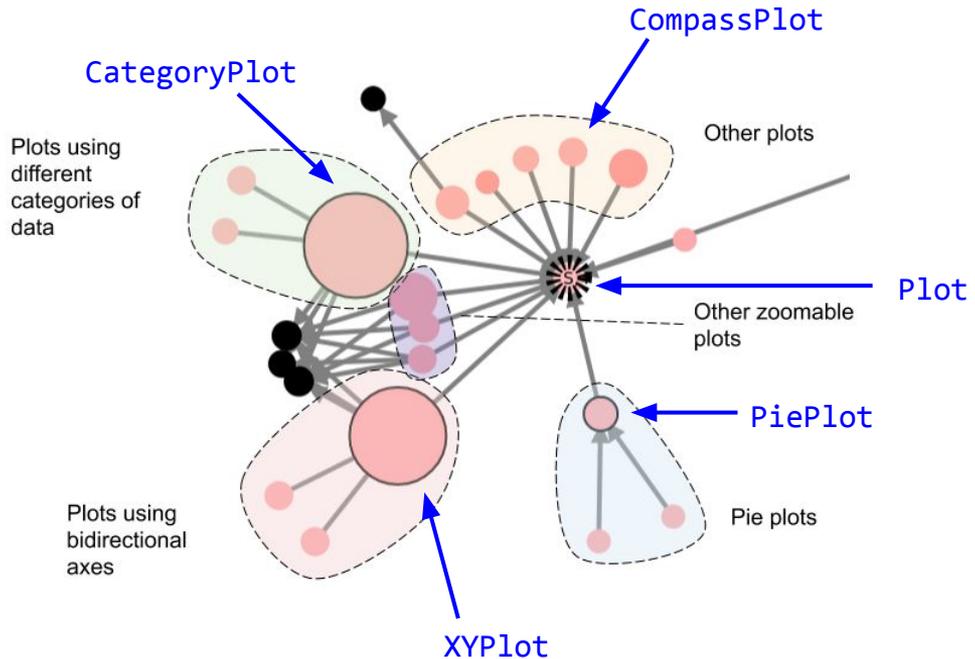
Bibliothèque permettant de tracer différents types de graphiques

# JFreeChart

Bibliothèque permettant de tracer **différents types de graphiques** ← **variabilité ?**

# JFreeChart

Bibliothèque permettant de tracer **différents types de graphiques** ← **variabilité!**



# JHipster

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

Construit comme une ligne de produit

# JHipster

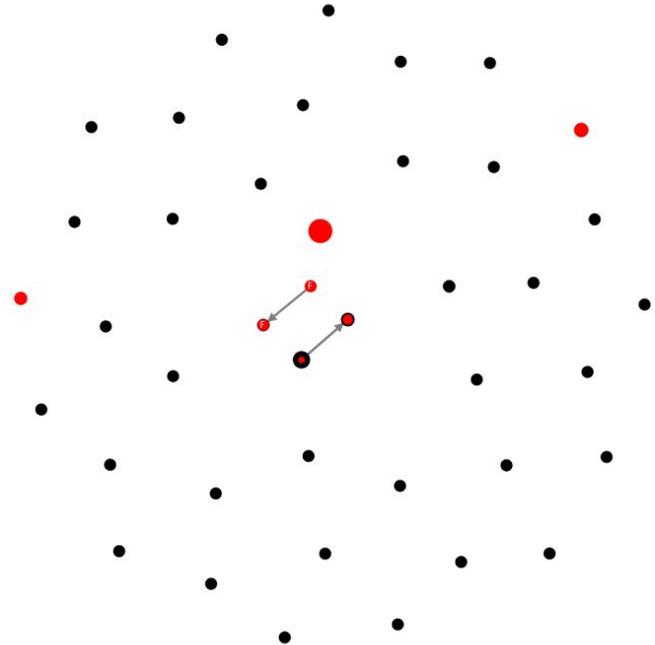
Outil de configuration de projets à partir d'un choix de pile technologique

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**



# Que s'est-il passé ?

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

Partie analysée : moteur qui orchestre les autres et qui est peu variable.

## Bilan néanmoins positif!

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

→ Pas de faux-positifs



# Des projets similaires mais variés

<b>Système</b>	<b># LoC</b>	<b># <i>vp-s</i></b>	<b># variantes</b>
Java AWT	69,974	1,221	1,808
Apache CXF 3.2.7	48,655	7,468	9,201
JUnit 4.12	9,317	253	319
Apache Maven 3.6.0	105,342	1,443	1,393
JHipster 2.0.28	2,535	140	115
JFreeChart 1.5.0	94,384	1,415	2,103
JavaGeom	32,755	720	919
ArgoUML	178,906	2,451	3,079

# Des projets similaires mais variés

# ≠ métiers

<b>Système</b>	<b># LoC</b>	<b># <i>vp</i>-s</b>	<b># variantes</b>
<b>Java AWT</b>	69,974	1,221	1,808
Apache CXF 3.2.7	48,655	7,468	9,201
<b>JUnit 4.12</b>	9,317	253	319
<b>Apache Maven 3.6.0</b>	105,342	1,443	1,393
JHipster 2.0.28	2,535	140	115
JFreeChart 1.5.0	94,384	1,415	2,103
JavaGeom	32,755	720	919
ArgoUML	178,906	2,451	3,079

# Des projets similaires mais variés

# ≠ # LoCs

<b>Système</b>	<b># LoC</b>	<b># <i>vp-s</i></b>	<b># variantes</b>
Java AWT	69,974	1,221	1,808
Apache CXF 3.2.7	48,655	7,468	9,201
JUnit 4.12	<b>9,317</b>	253	319
Apache Maven 3.6.0	105,342	1,443	1,393
JHipster 2.0.28	2,535	140	115
JFreeChart 1.5.0	94,384	1,415	2,103
JavaGeom	32,755	720	919
ArgoUML	<b>178,906</b>	2,451	3,079

# Une découverte ! Pas de corrélation #LoC / #vp-s / variantes

Système	# LoC	# vp-s	# variantes
Java AWT	69,974	1,221	1,808
Apache CXF 3.2.7	<b>48,655</b>	<b>7,468</b>	<b>9,201</b>
JUnit 4.12	9,317	253	319
Apache Maven 3.6.0	105,342	1,443	1,393
JHipster 2.0.28	2,535	140	115
JFreeChart 1.5.0	94,384	1,415	2,103
JavaGeom	32,755	720	919
ArgoUML	<b>178,906</b>	<b>2,451</b>	<b>3,079</b>

# Nos sous-questions

1. Comment identifier des implémentations de variabilité d'un système en ayant pour seules données son code source ? **La densité de symétries semble être un moyen viable pour identifier des points de variation et leurs variantes.**
2. Est-ce que les implémentations de variabilité identifiées correspondent vraiment à de la variabilité ? **Bonne question...**
3. Comment indiquer à un utilisateur les zones de forte densité d'un projet ? **Un graphe semble approprié.**

# 3.2

## Comment vérifier la pertinence de notre détection ?



## Question : Est-ce qu'on peut relier nos vp-s et variantes à des fonctionnalités ?

Besoin d'un projet pour lequel on a les informations que l'on cherche

- le rêve : un feature model
- au moins : apping implémentation  $\leftrightarrow$  fonctionnalité existant

Où trouver un tel projet ?

## Question : Est-ce qu'on peut relier nos vp-s et variantes à des fonctionnalités ?

Besoin d'un projet pour lequel on a les informations que l'on cherche

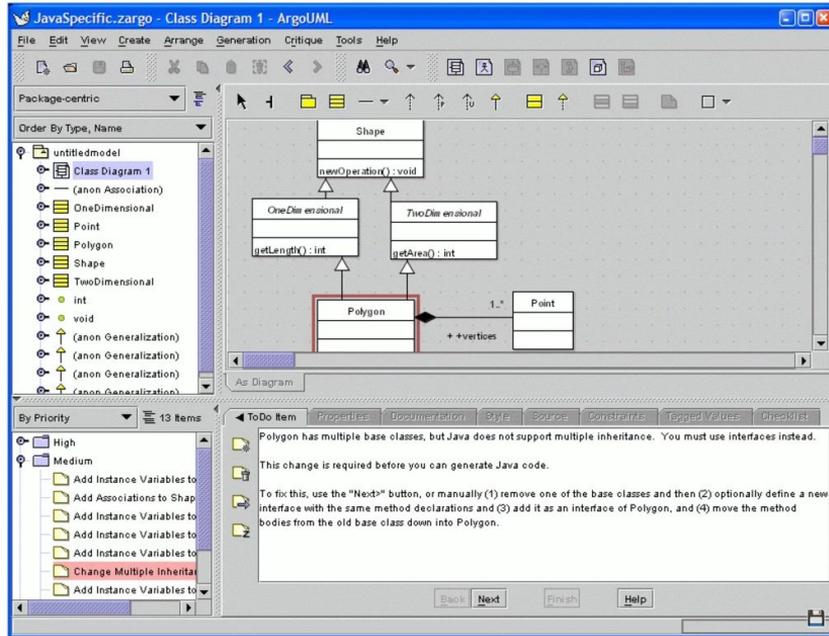
- le rêve : un feature model
- au moins : apping implémentation ↔ fonctionnalité existant

Où trouver un tel projet ?

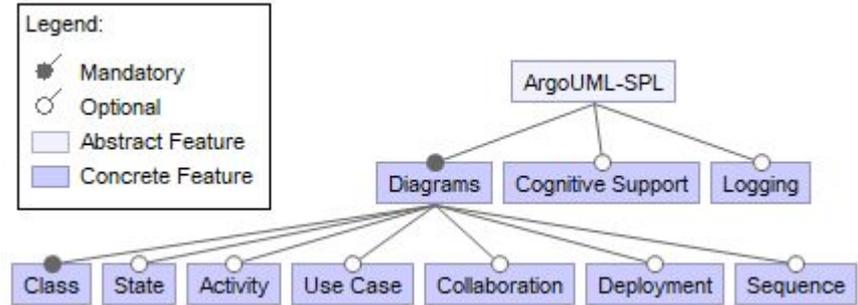
Rechercher dans l'état de l'art !

- indice pour un projet de qualité
- possibilité de comparaison avec d'autres techniques similaires

# ArgoUML-SPL [Couto2011]

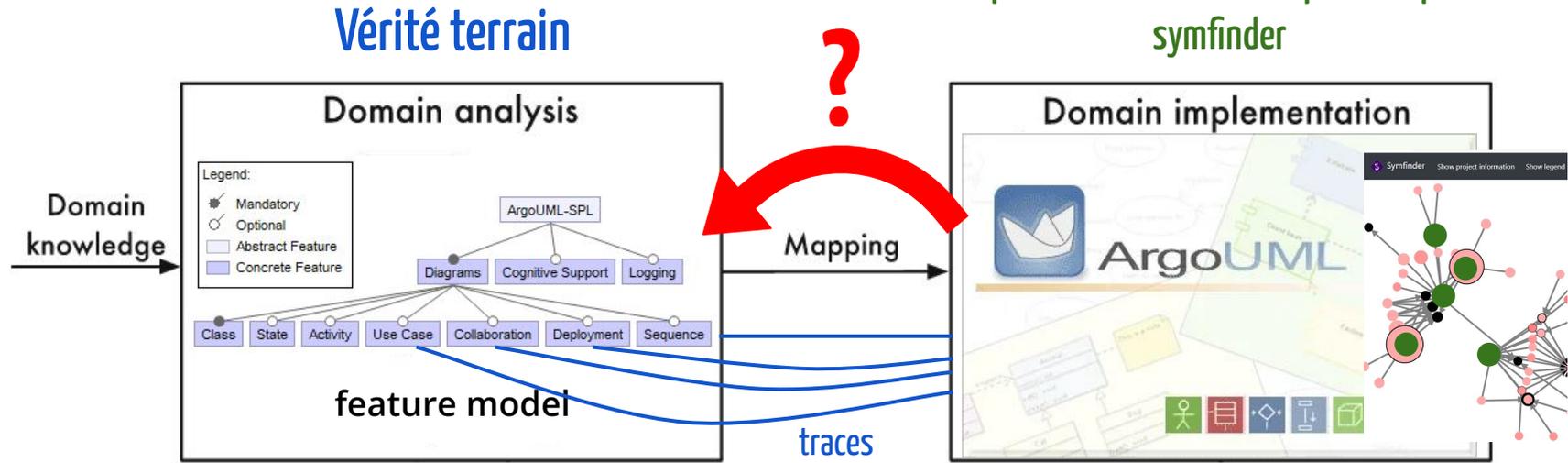


Éditeur d'ArgoUML



Feature model d'ArgoUML-SPL

# Question: Est-ce que les *vp-s* identifiés dans ArgoUML correspondent à des fonctionnalités du feature model ?



# Que voit-on ?

## Fonctionnalité : *Sequence*

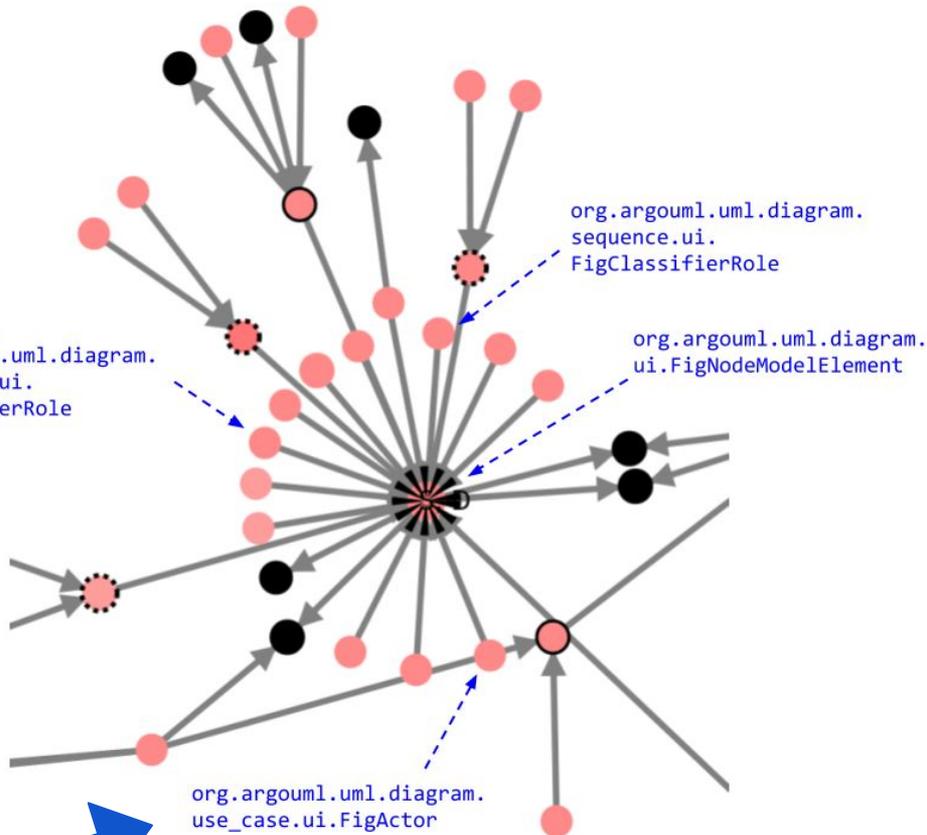
```
##if defined(SEQUENCEDIAGRAM)  
##@$LPS-SEQUENCEDIAGRAM:GranularityType:Package  
public class FigClassifierRole extends FigNodeModelElement
```



org.argouml.uml.diagram.  
deployment.ui.  
FigClassifierRole

org.argouml.uml.diagram.  
sequence.ui.  
FigClassifierRole

org.argouml.uml.diagram.  
ui.FigNodeModelElement



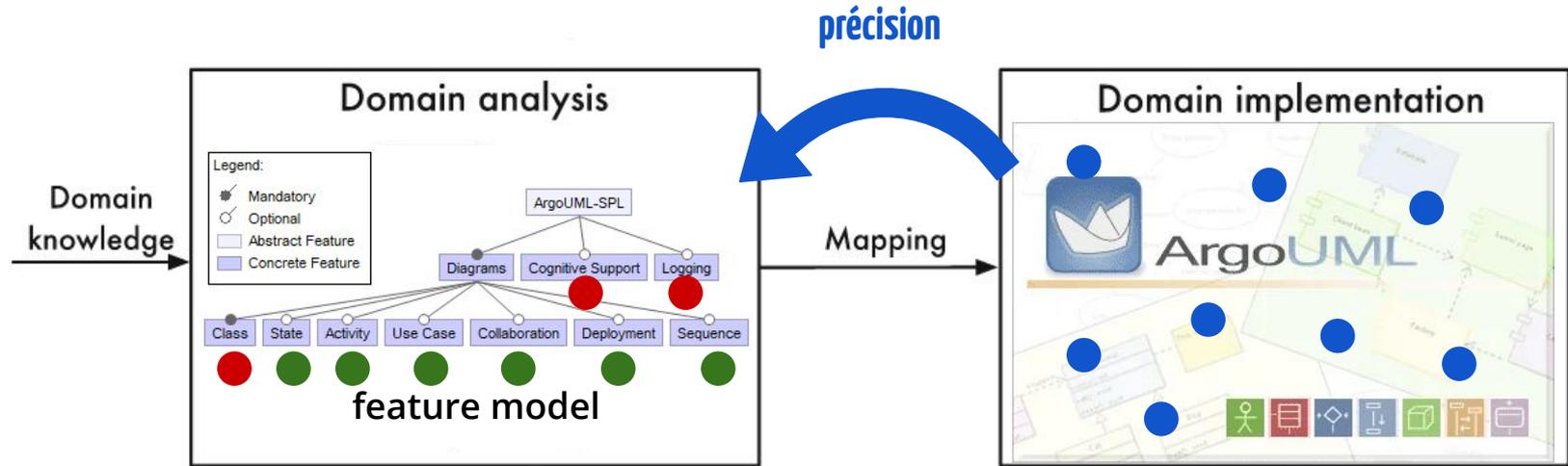
## Fonctionnalité : *Use Case*

```
##if defined(USECASEDIAGRAM)  
##@$LPS-USECASEDIAGRAM:GranularityType:Package  
public class FigActor extends FigNodeModelElement
```



org.argouml.uml.diagram.  
use\_case.ui.FigActor

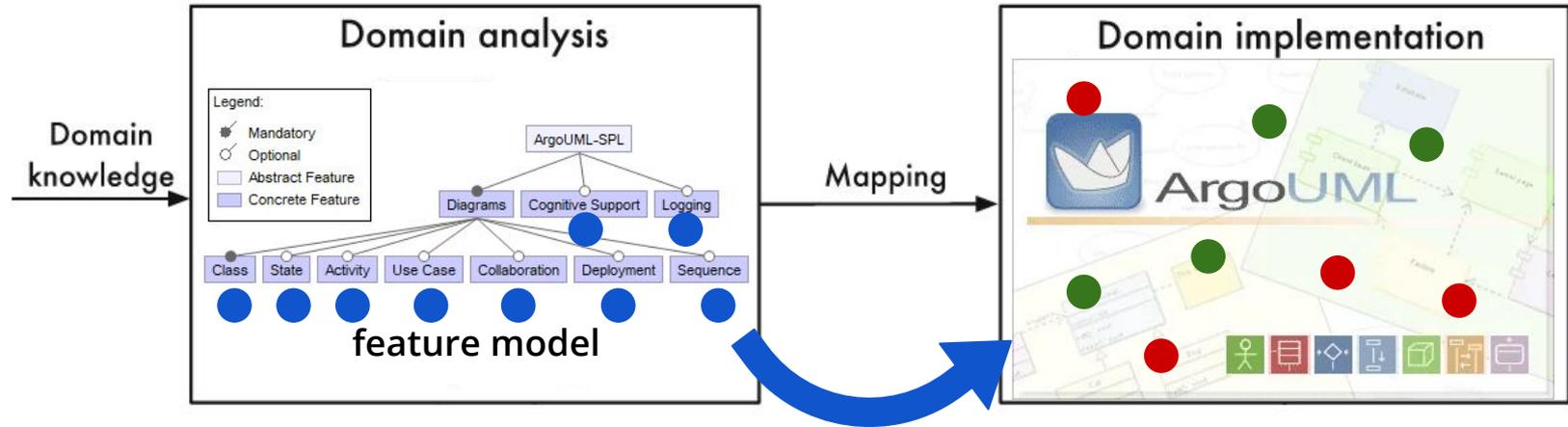
# Pertinence des *vp-s*



## Précision :

Pourcentage des *vp-s* et variantes identifiés qui peuvent être reliés à des fonctionnalités

# Pertinence des *vp-s*



## Rappel :

Pourcentage des traces de fonctionnalités qui peuvent être reliées à des *vp-s* et variantes identifiés

rappel

# Résultats obtenus

Précision :

**38 %**

Résultat faible, mais attendu :

- fonctionnalités à gros grain
- symétries pas uniquement liées à de la variabilité

Rappel :

**83 %**

Bon résultat !

**Analyse manuelle** des 17% de traces restantes  
→ ne sont pas liées à de la variabilité

# Nos sous-questions

1. Comment identifier des implémentations de variabilité d'un système en ayant pour seules données son code source ? **La densité de symétries semble être un moyen viable pour identifier des points de variation et leurs variantes.**
2. Est-ce que les implémentations de variabilité identifiées correspondent vraiment à de la variabilité ? **La majorité des implémentations de variabilité est identifiée, mais beaucoup de faux-positifs le sont également.**
3. Comment indiquer à un utilisateur les zones de forte densité d'un projet ? **Un graphe semble approprié.**

# Prenons un peu de recul...

Rappel du sujet initial :

**Identification, visualisation et gestion de variabilité au sein de  
grands systèmes orientés objets hautement variables**

# Prenons un peu de recul...

Rappel du sujet initial :

**Identification, visualisation et gestion de variabilité au sein de**  
**grands** systèmes orientés objets hautement variables

# Prenons un peu de recul...

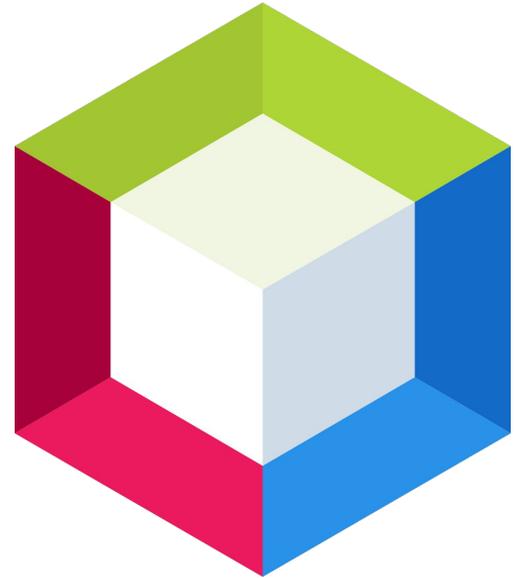
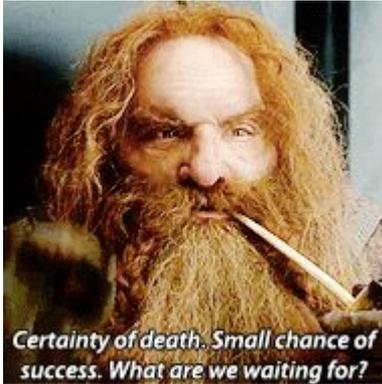
Rappel du sujet initial :

Identification, visualisation et gestion de variabilité au sein de  
**grands** systèmes orientés objets hautement variables

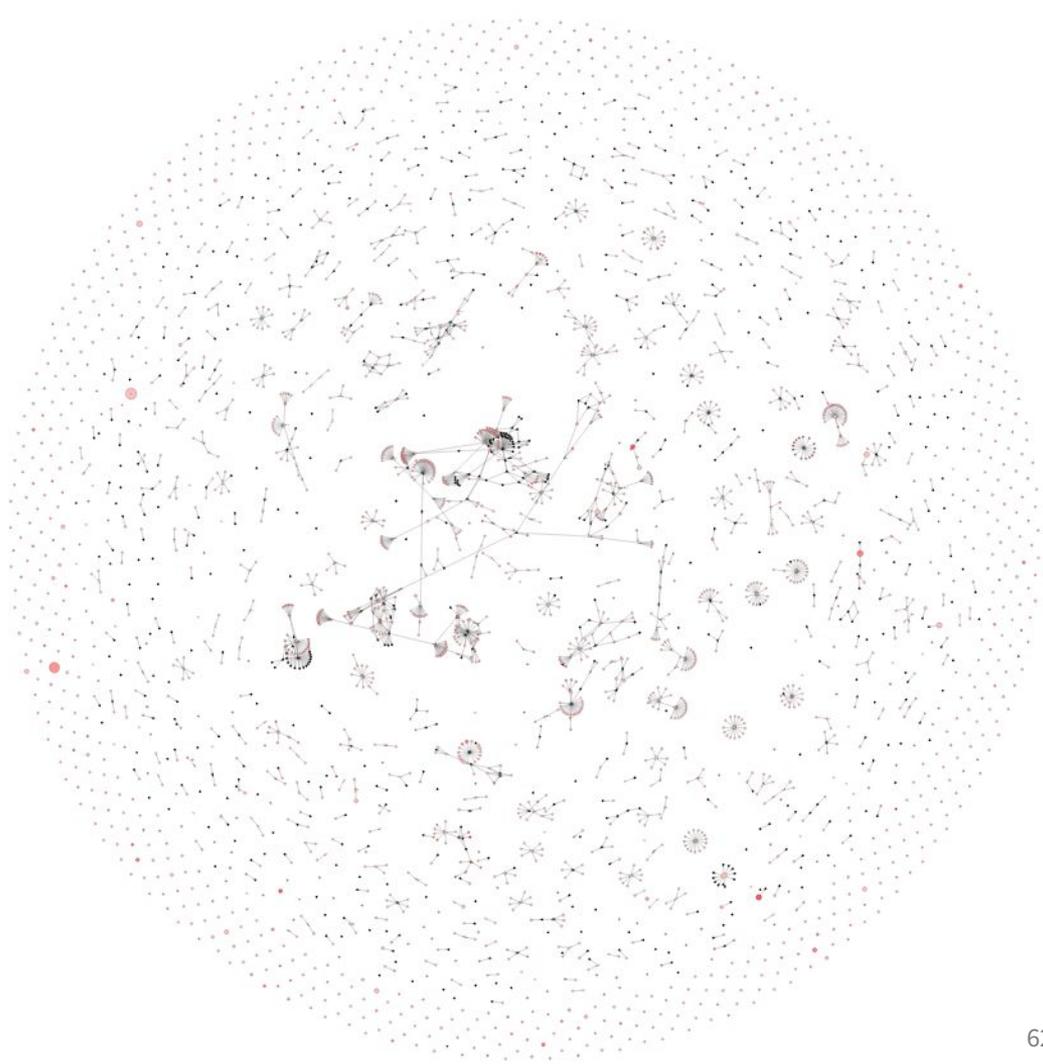
Plus grand système étudié : ArgoUML (179k LoC) → il nous faut plus grand !

# Apache NetBeans

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



**Passage à l'échelle  
de l'outil ✓**



**Passage à l'échelle  
de la visualisation...**

# Nos sous-questions

1. Comment identifier des implémentations de variabilité d'un système en ayant pour seules données son code source ? **La densité de symétries semble être un moyen viable pour identifier des points de variation et leurs variantes.**
2. Est-ce que les implémentations de variabilité identifiées correspondent vraiment à de la variabilité ? **La majorité des implémentations de variabilité est identifiée, mais beaucoup de faux-positifs le sont également.** **Au suivant !**
3. Comment indiquer à un utilisateur les zones de forte densité d'un projet ? **Un graphe semble approprié sur de petits systèmes mais est moins lisible sur de grands systèmes.**

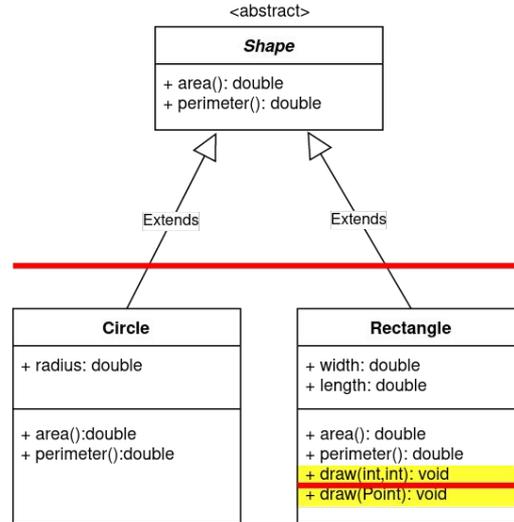
# 3.3

**Sommes-nous  
sur le bon  
chemin ?**

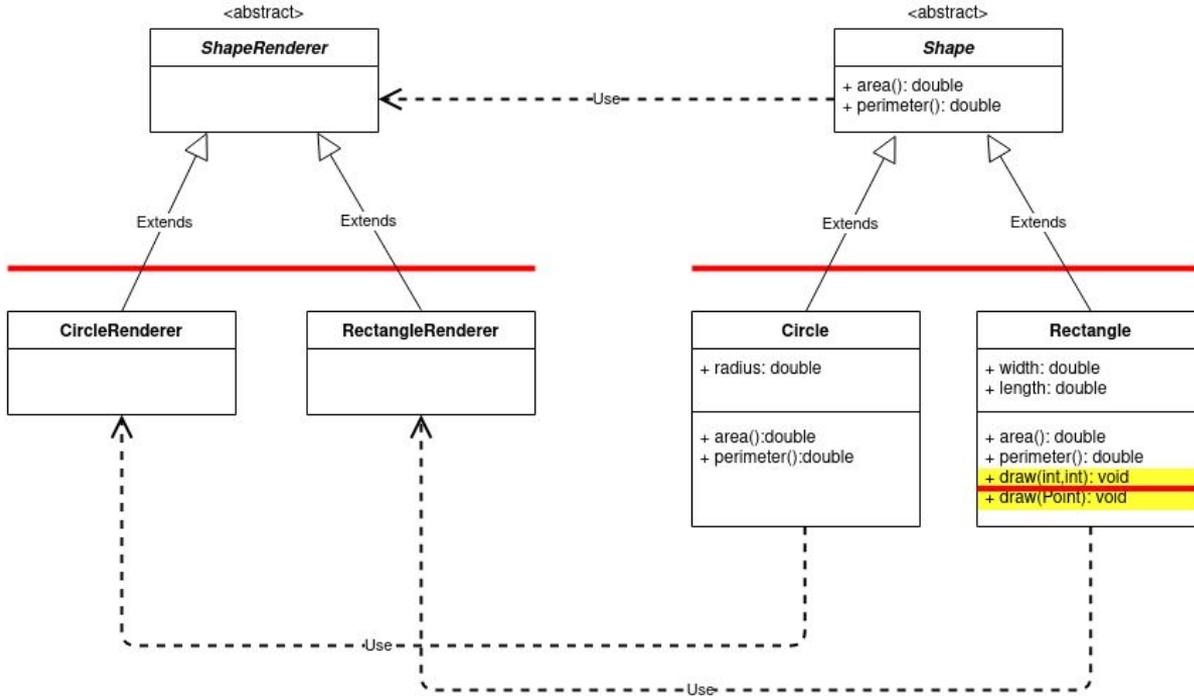


Source : Image by [ivalbak](#)  
from [Pixabay](#)

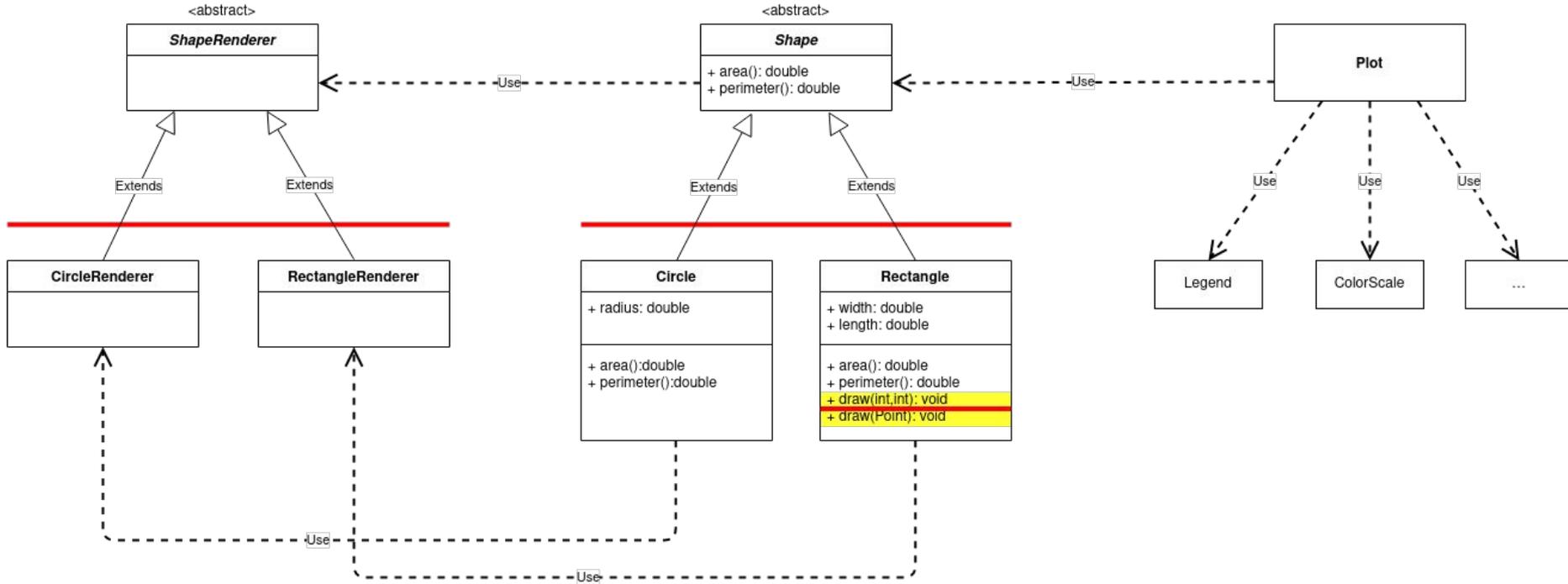
# Back to the origins...

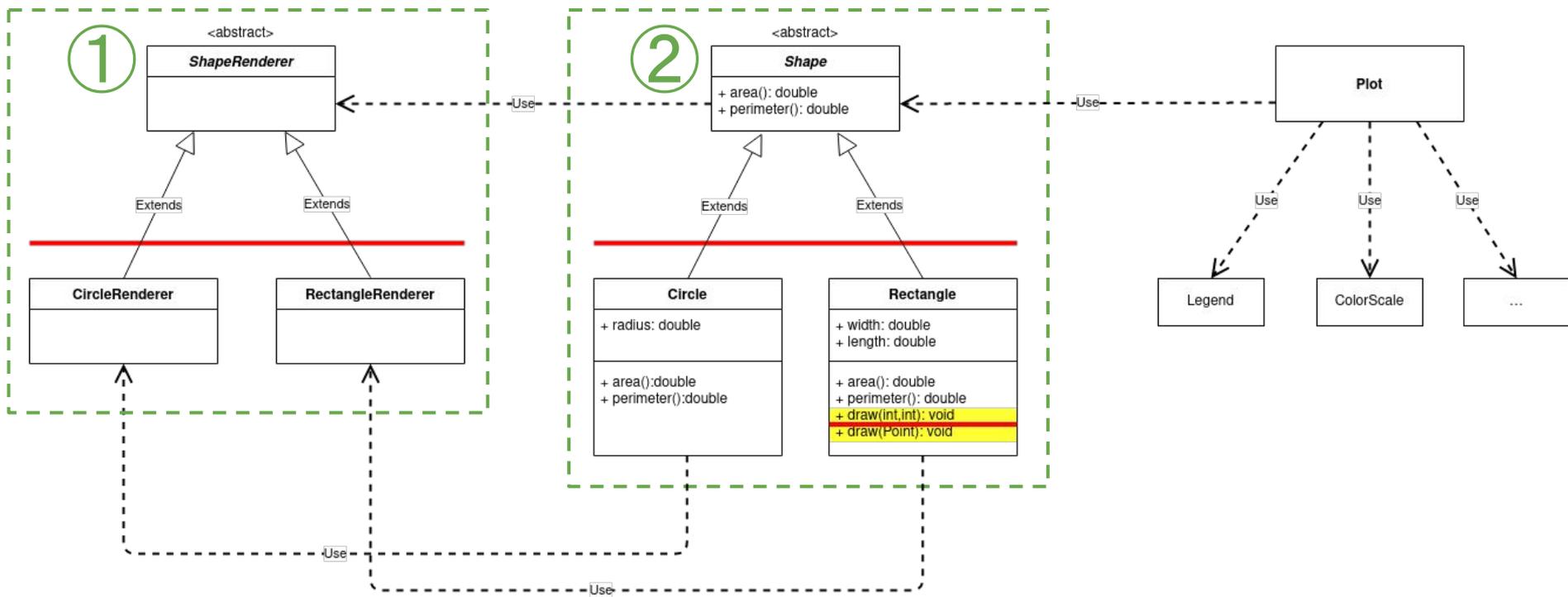


# Variability implementations *in the wild*

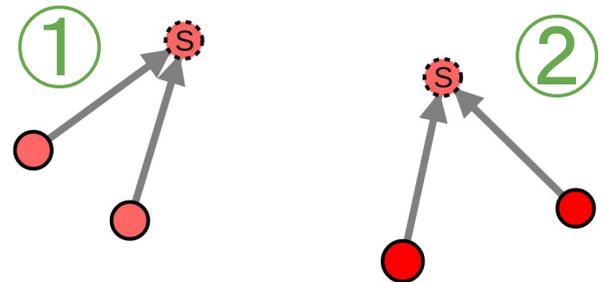


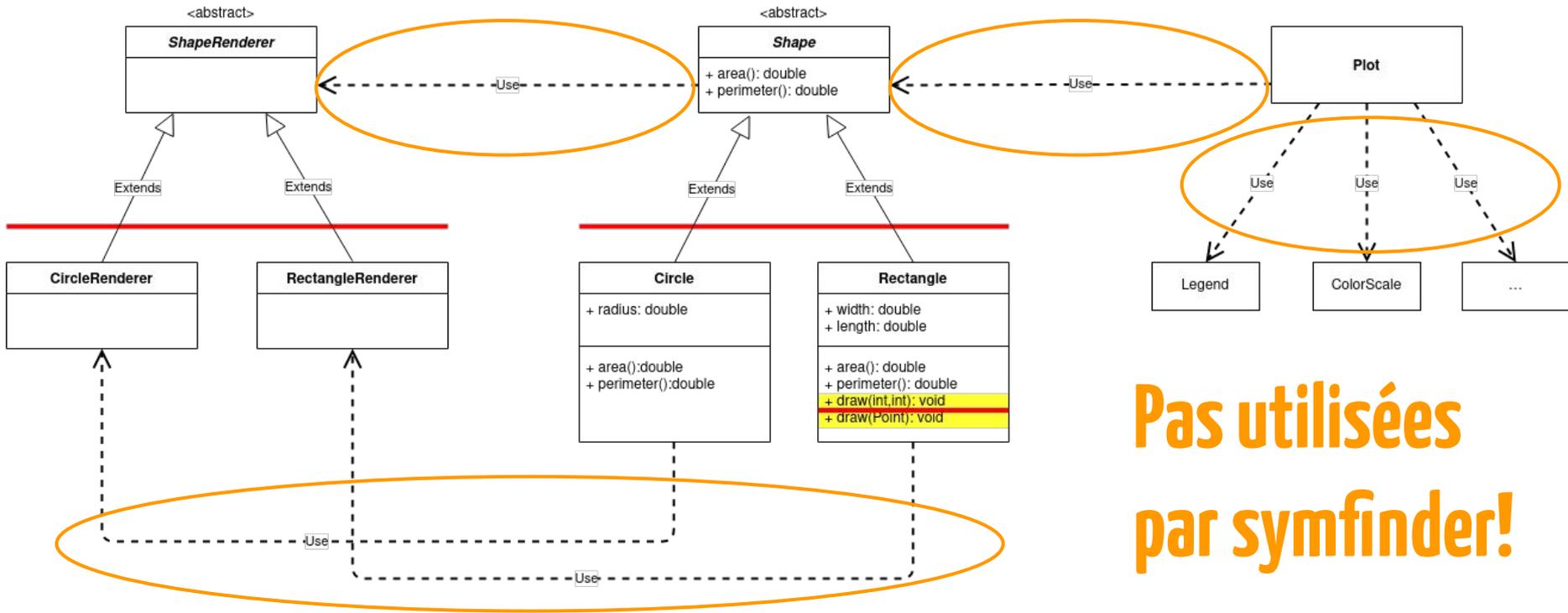
# Variability implementations *in the wild*





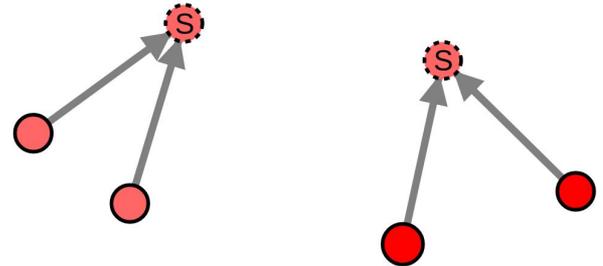
Visualisation générée par symfinder

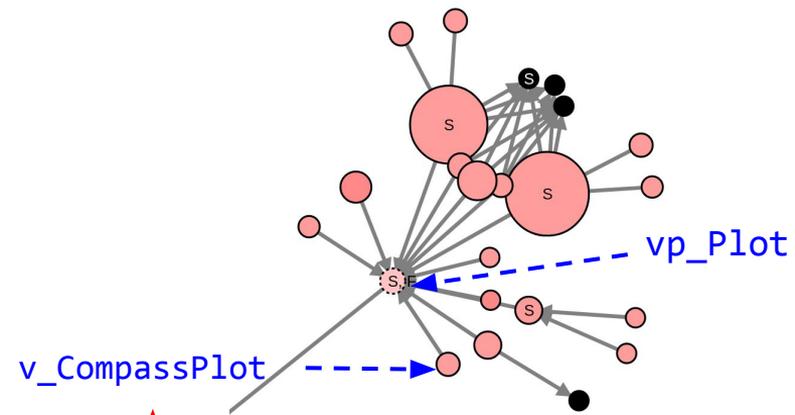




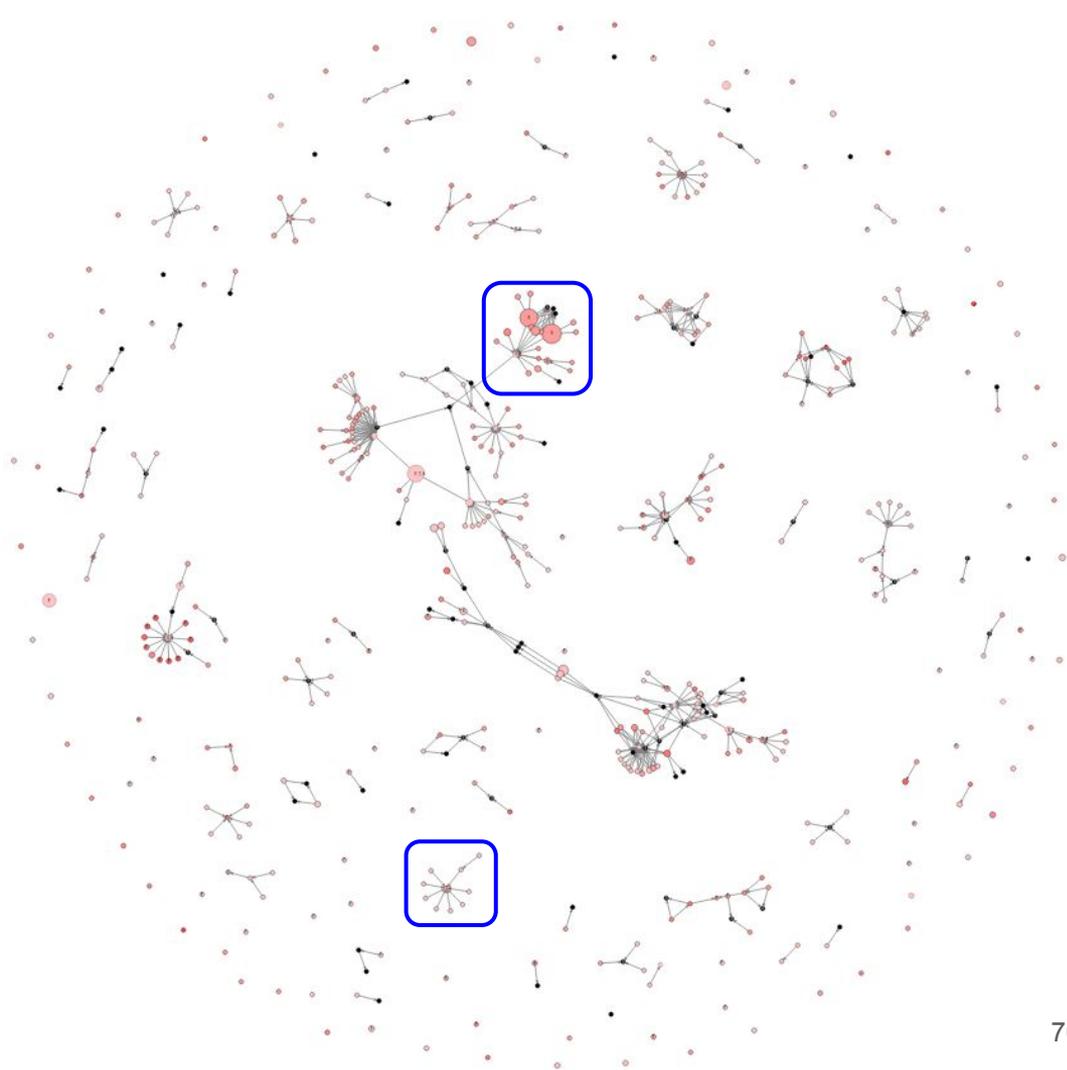
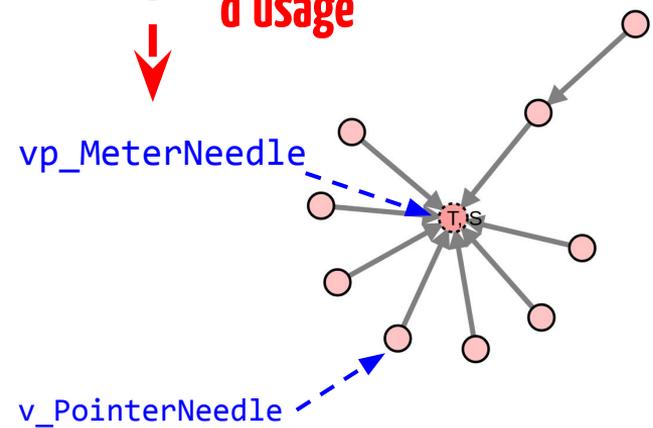
**Pas utilisées  
par symfinder!**

Visualisation générée par symfinder





**relation  
d'usage**



# symfinder-2

- Ajout des relations d'usage
- Redéfinition de la densité
- Nouvelle visualisation



Package/class to filter  Add new filter

Some entry point class  Add an entry point class

Usage-type

Hybrid view

Packages filtered out (3 packages) +

Entry point classes filtered in (2 classes) +

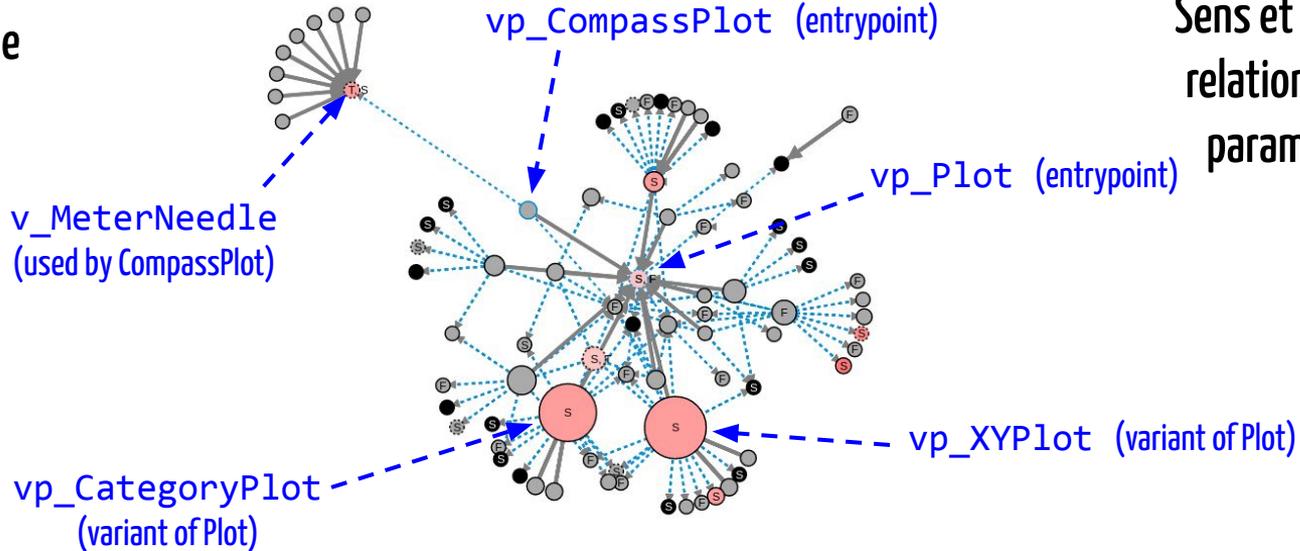
org.jfree.chart.plot.CompassPlot	x
org.jfree.chart.plot.Plot	x

Usage-level

Number of class level VPs: 259  
 Number of method level VPs: 667  
 Number of class level variants: 275  
 Number of method level variants: 1648

Classes "entrypoints"  
pour explorer le code

Sens et niveau des relations d'usage paramétrables



# Et la densité dans tout ça ?

**Détermination automatique des zones denses  
avec des seuils sur les variantes et leur proximité  
par les usages.**

↑ seuil #variantes et ↓ seuil proximité

⇒ ↓ #classes faisant partie de zones denses

Project	symfinder nodes	symfinder-2		
		≥ 5 v-s ≤ 3 hops	≥ 10 v-s ≤ 3 hops	≥ 30 v-s ≤ 2 hops
Java AWT	431	28	22	3
Apache CXF	3086	98	32	4
JUnit	118	5	0	0
Maven	616	8	1	0
JFreeChart	578	34	15	3
ArgoUML	1258	40	15	3
Cucumber	331	4	0	0
Logbook	117	0	0	0
Riptide	89	0	0	0
NetBeans	3494	58	22	2

⇒ on a des paramètres pour réduire le nombre de classes identifiées  
mais on doit les trouver manuellement

# Nos sous-questions

1. Comment identifier des implémentations de variabilité d'un système en ayant pour seules données son code source ? **La densité de symétries semble être un moyen viable pour identifier des points de variation et leurs variantes. Mais il faudrait la caractériser.**
2. Est-ce que les implémentations de variabilité identifiées correspondent vraiment à de la variabilité ? **La majorité des implémentations de variabilité est identifiée, mais beaucoup de faux-positifs le sont également. Cependant, on a une piste pour les limiter.**
3. Comment indiquer à un utilisateur les zones de forte densité d'un projet ? **Un graphe semble approprié sur de petits systèmes mais devient illisible sur de grands systèmes.**

# 3.4

## Vers une visualisation plus intuitive



Source : Image by [FunkyFocus](#)  
from [Pixabay](#)

# Question : Quelle nouvelle visualisation adopter ?

Rechercher dans l'état de l'art !

1. Quelles visualisations sont utilisées dans les analyses de systèmes variables ?
2. Quelles visualisations ont été développées pour aider la compréhension de grands systèmes ?

# Q1 in a nutshell

Visualisations de systèmes variables existantes ?

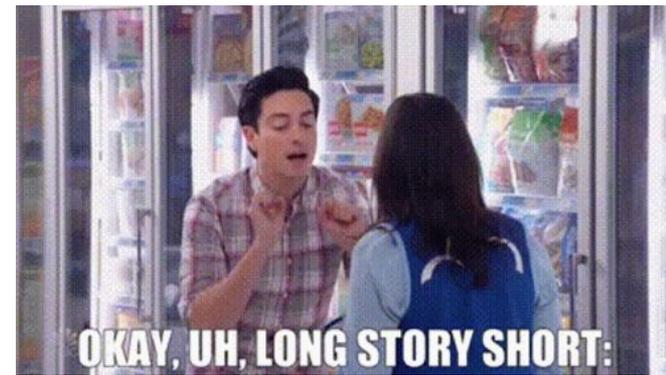
Recherche bibliographique !

Utilisent principalement les features du système

→ on ne les a pas :(

Bon ou mauvais signe pour nous ?

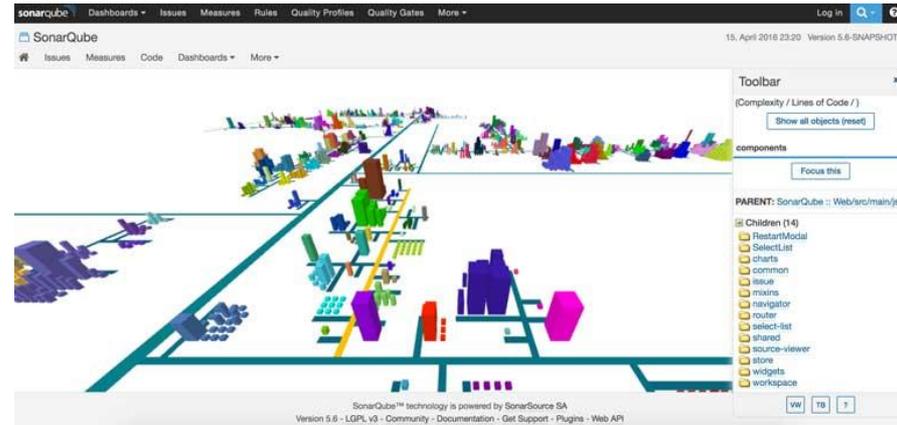
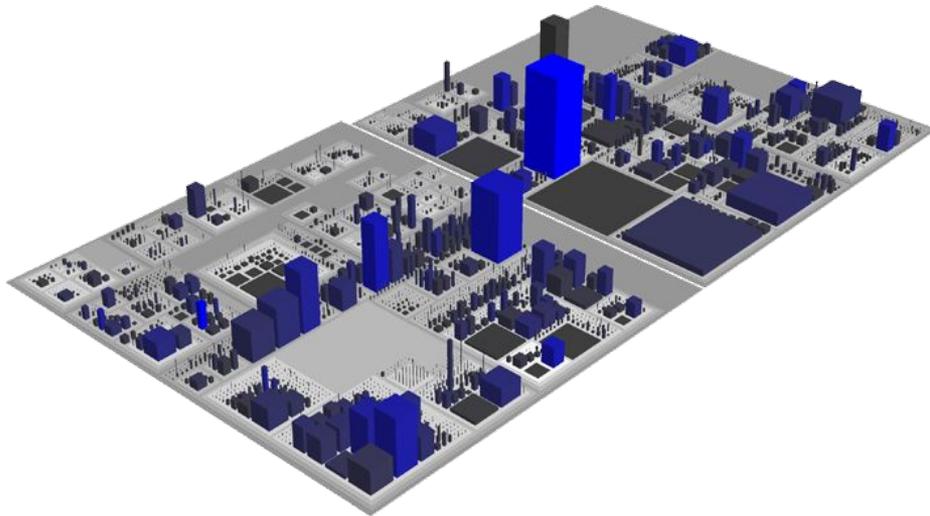
On ne peut rien réutiliser, mais... on peut le faire nous-mêmes !



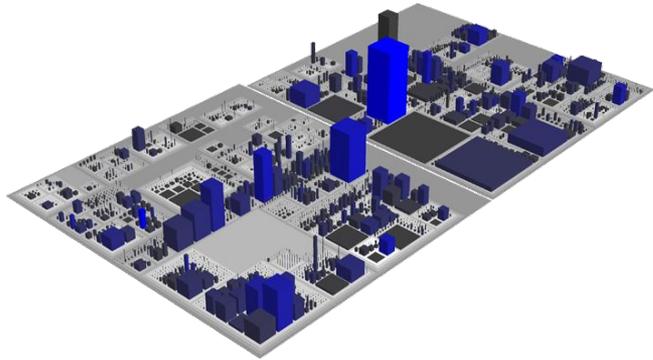
# CodeCity [Wettel2007] et Evo-Street [Steinbrückner2013]

Utilisation de la **métaphore de la ville** pour représenter un système orienté objet, en se reposant sur des **métriques de qualité du code**, pour **détecter des code smells**.

Visualisations intégrées à des systèmes de suivi de la qualité du code tels que SonarQube.



# Est-ce qu'on peut s'en servir ?



- Approche validée par la communauté



- Applicable sur des systèmes OO



- Montre des groupements de classes

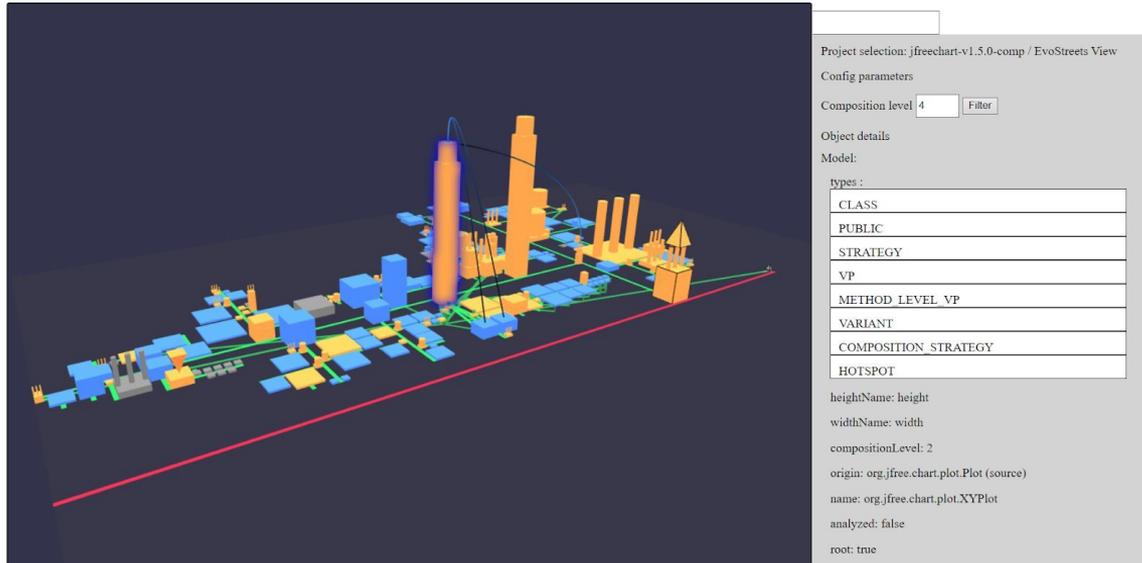


- Les classes sont groupées par package

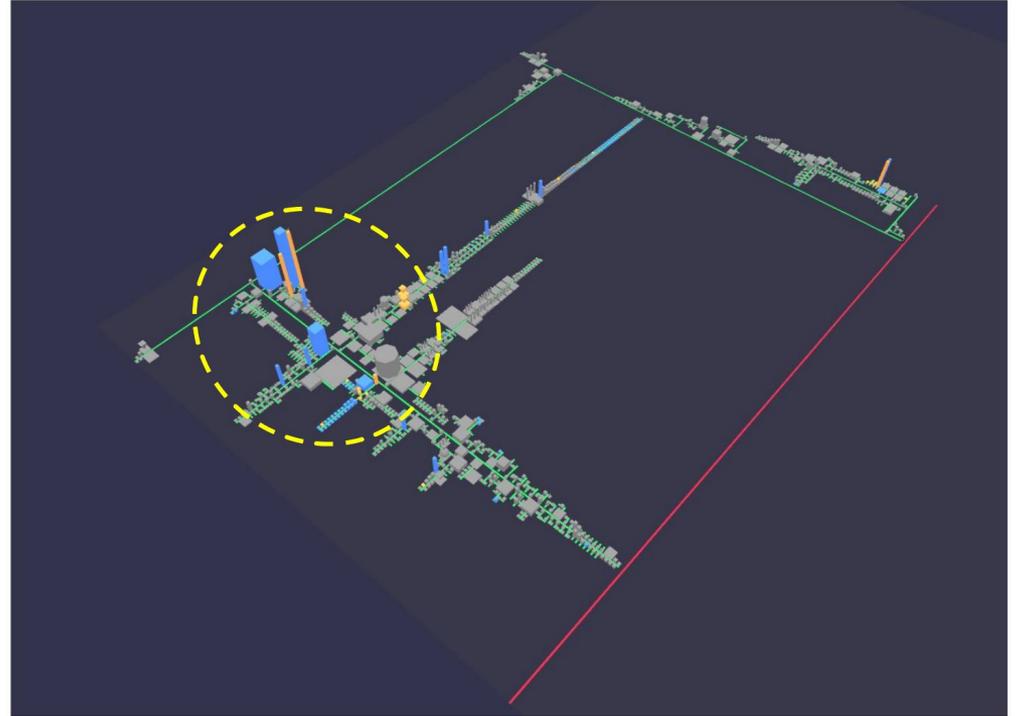


# From CodeCity to VariCity

Utilisation de la **métaphore de la ville** pour représenter un système orienté objet, en se reposant sur des **métriques de variabilité**, pour **identifier des zones de concentration de vp-s et variantes**.



# Passage à l'échelle de la visualisation ✓



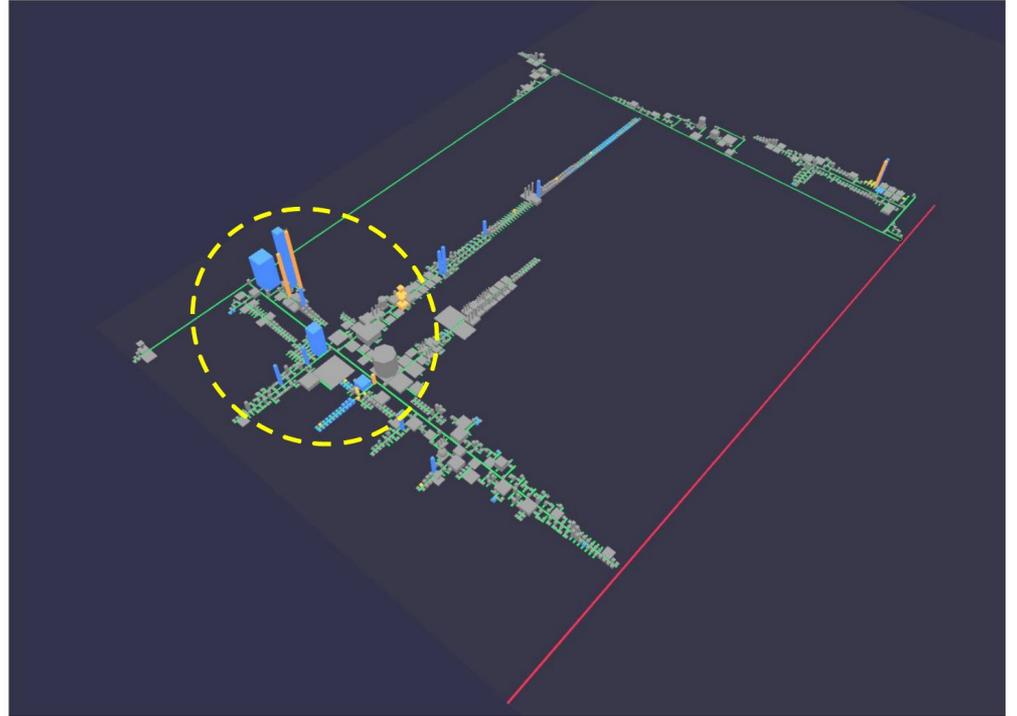
Sous-partie du package java de NetBeans 12.2

# Nos sous-questions

1. Comment identifier des implémentations de variabilité d'un système en ayant pour seules données son code source ? **La densité de symétries semble être un moyen viable pour identifier des points de variation et leurs variantes. Mais il faudrait la caractériser.**
2. Est-ce que les implémentations de variabilité identifiées correspondent vraiment à de la variabilité ? **La majorité des implémentations de variabilité est identifiée, mais beaucoup de faux-positifs le sont également. Cependant, on a une piste pour les limiter.**
3. Comment indiquer à un utilisateur les zones de forte densité d'un projet ? **Une visualisation sous forme de ville semble donner de bons résultats.**

Passage à l'échelle  
de la visualisation ✓

Mais est-ce que c'est  
pratique à utiliser ?



Sous-partie du package java de NetBeans 12.2

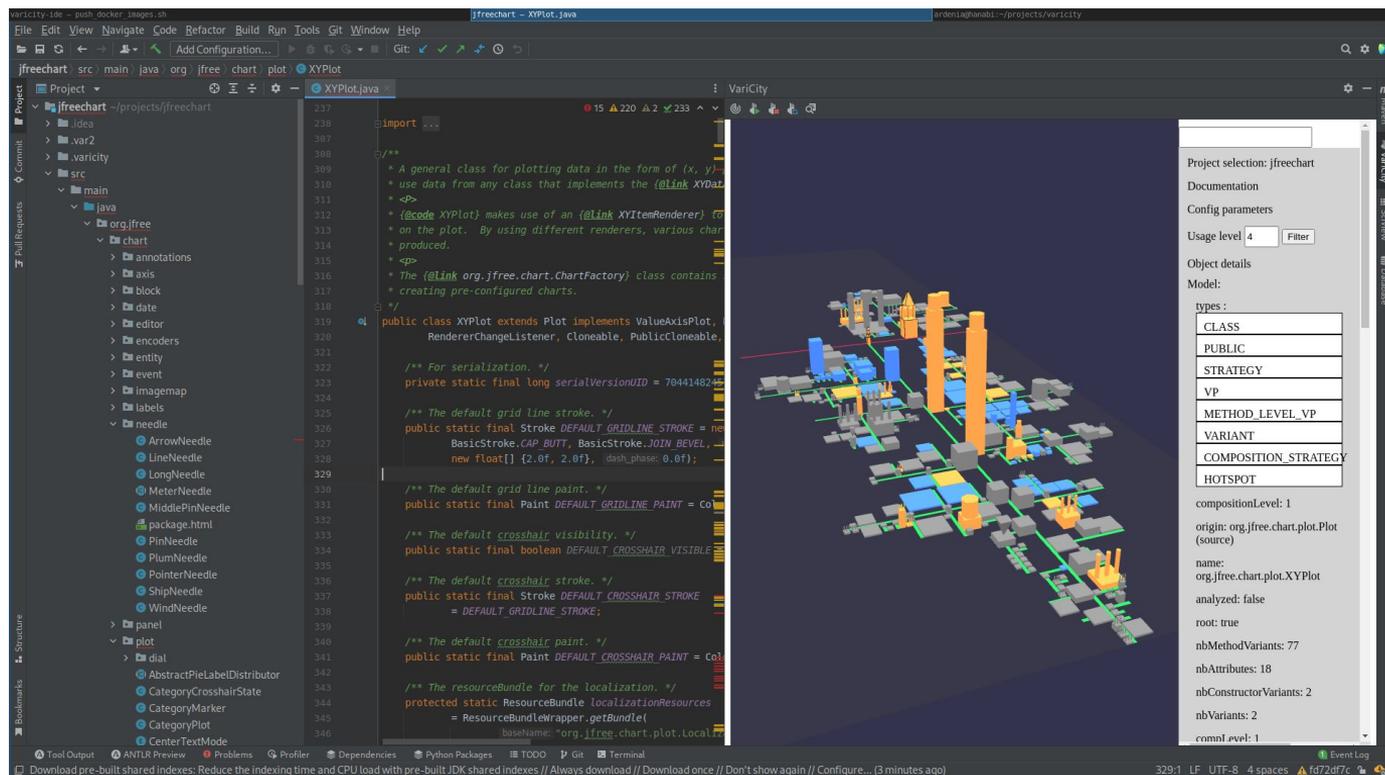
**CodeCity et Evo-Streets** utilisent des **métriques de qualité du code**

**VariCity** utilise des **métriques de variabilité du code**

⇒ visualisations intégrées à **SonarQube**

⇒ visualisation intégrée à **IntelliJ**

⇒ **code et métriques au même endroit**



**Dédicace à João BRILHANTE, Charly DUCROcq et Ludovic MARTI !**

# Encore d'autres questions !

Est-ce qu'il existe un lien  
variabilité  $\leftrightarrow$  qualité du code ?



Dédicace à Patrick ANAGONOU, Guillaume SAVORNIN et Anton VAN DER TUIJN !

Est-ce qu'on gère de la variabilité dans  
d'autres langages OO comme en Java ?

Exploration de bases de code en C++ et JavaScript

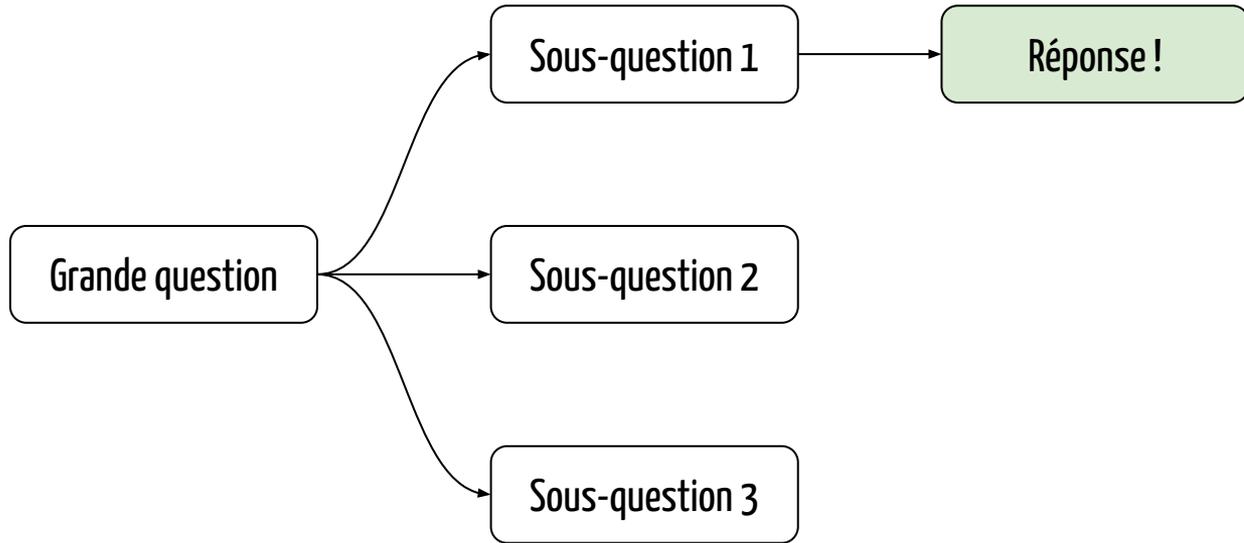
Dédicace à Théo FORAY, Grégoire PELTIER et Nathan STROBBE (SI5 AL 2020), et à Martin BRUEL !

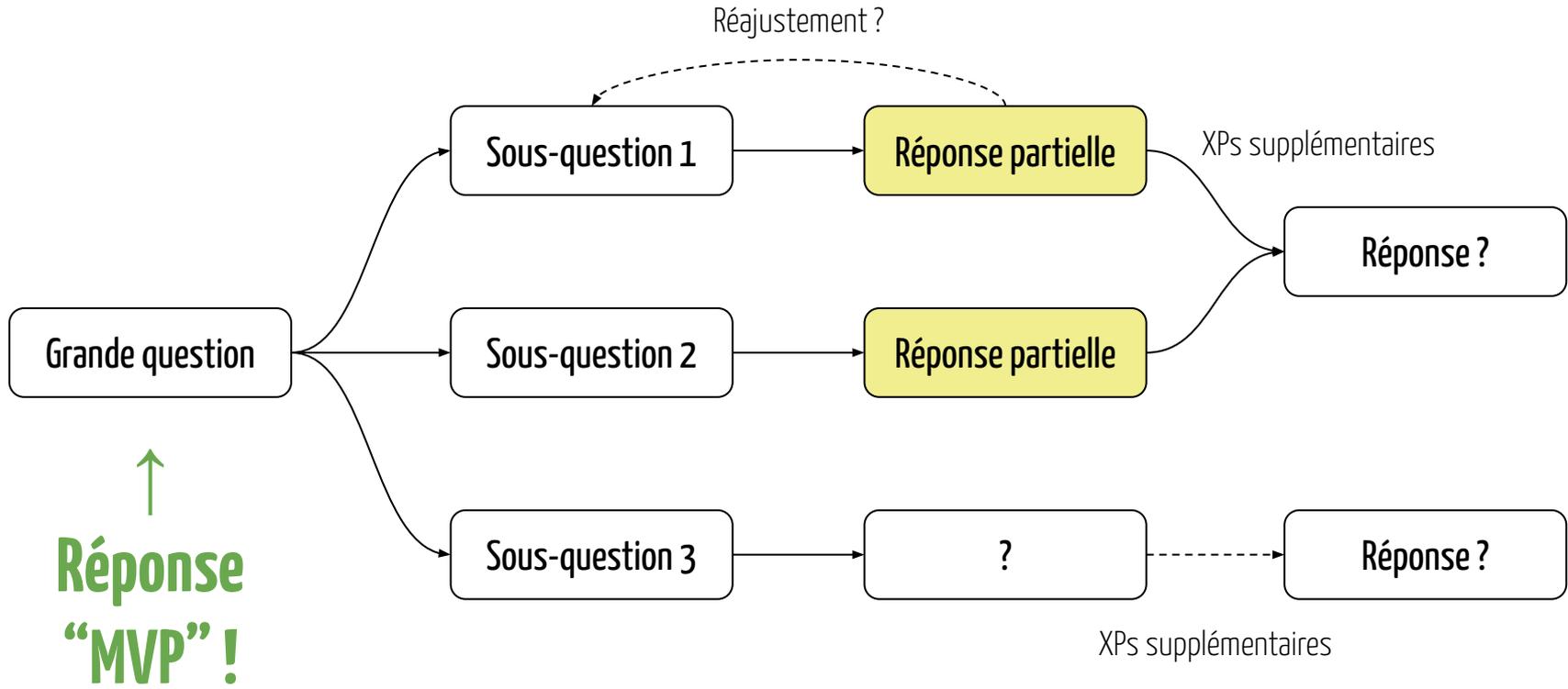
# 4

## En résumé



Source : Image by [StockSnap](#)  
from [Pixabay](#)





# La démarche scientifique est un questionnement permanent !

(77 “?” jusqu’ici...)

Chaque étape est source de questions permettant :

- **de délimiter précisément ce qu’on cherche** → **approche et dataset adaptés au problème**  
*À quelle question souhaite-t-on répondre ? De quelles informations ai-je besoin ? Où les trouver ? Comment les calculer ? ...*
- **d’analyser et comprendre les résultats des expériences**  
*À quoi correspond ce 68% ? Que sont les 32% restants ? Comment expliquer ça ? L’outil ? Le dataset ? ...*
- **de trouver les limitations de l’approche**  
*Est-ce lié à mon dataset ? De quelles informations supplémentaires aurais-je besoin ? ...*

# Par où commencer ?



1. Ne pas foncer tête baissée : commencer à explorer à la main pour défendre l'hypothèse.
2. Vous pourrez être amenés à revoir vos hypothèses de travail à tout moment, et c'est normal !
3. Un résultat contre-intuitif n'est pas forcément faux, mais un résultat attendu n'est pas forcément trivial.
4. Si résultat obtenu  $\neq$  résultat attendu, alors rétro-ingénierie à la main !
5. Si résultat obtenu = résultat attendu, alors on s'assure que ça ne tombe pas en marche !



# En route pour l'aventure !

Merci pour votre attention !