POLYTECH
NICE SOPHIA

Une aiguille dans une botte de code

Rétro-Ingénierie, Maintenance et Évolution du Logiciel

Johann Mortara

05/12/2022

Mon parcours

2014-2016 : PeiP à Polytech Nice Sophia

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

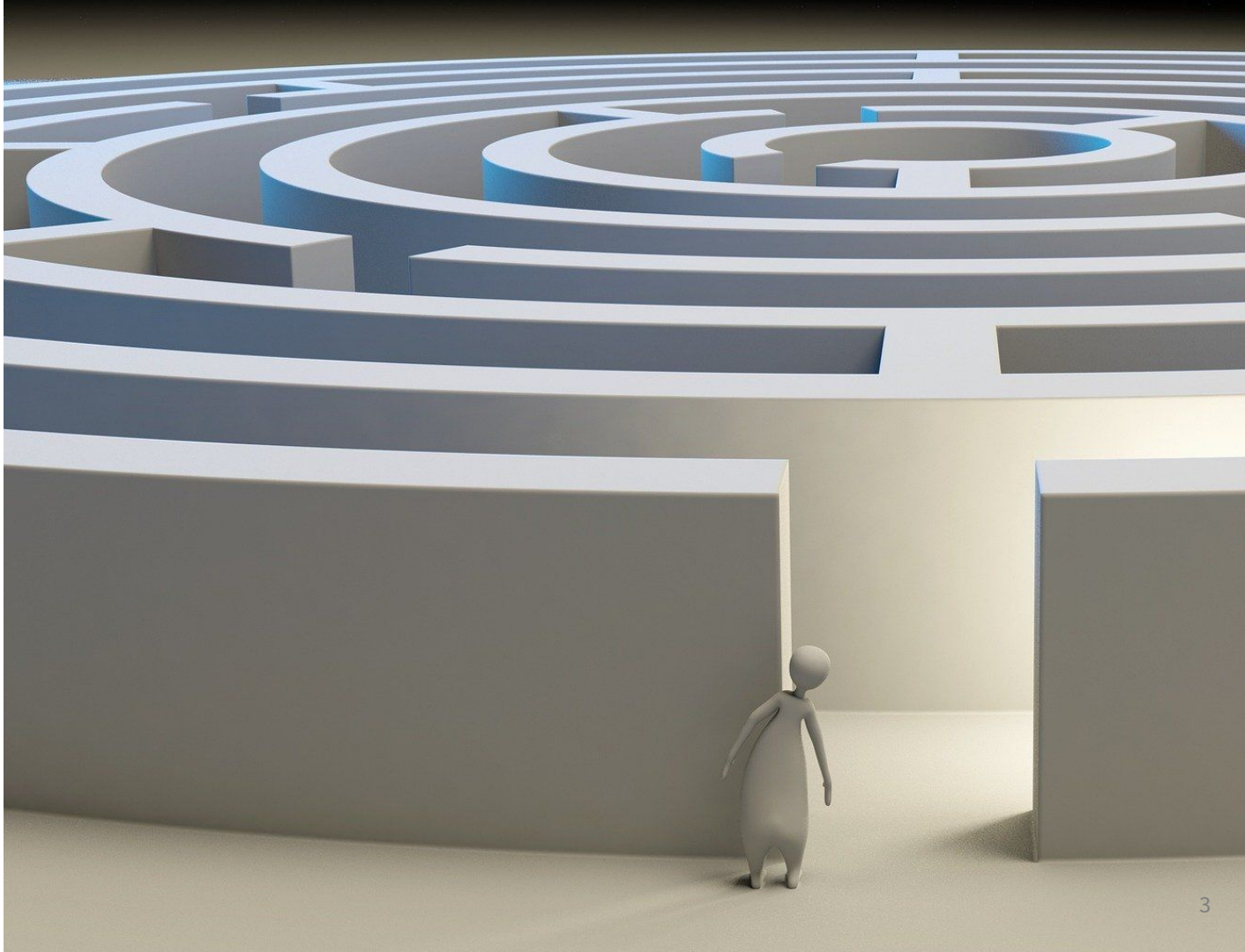
2019 : Doctorat au laboratoire I3S (directeur : Philippe Collet)

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

Pardon ?

1.

Quel est le problème ?



Source : Image by Arek Socha from Pixabay

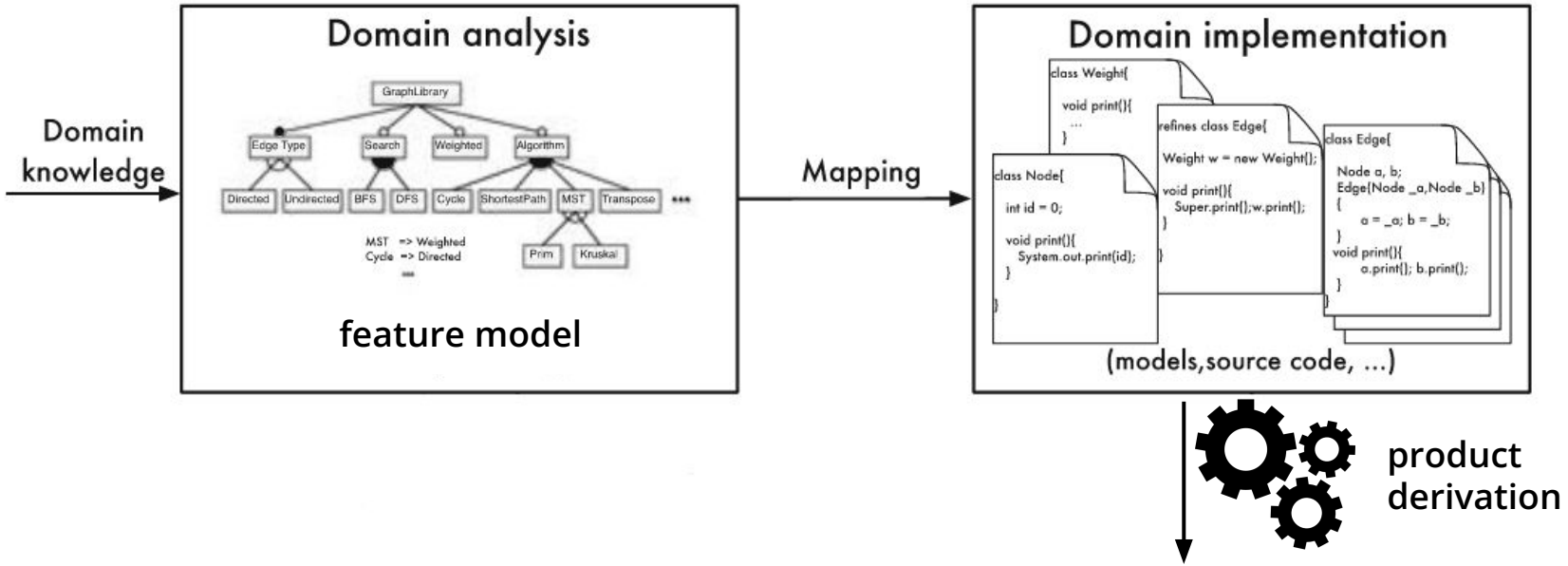
Vari-quoi ?

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
- ...

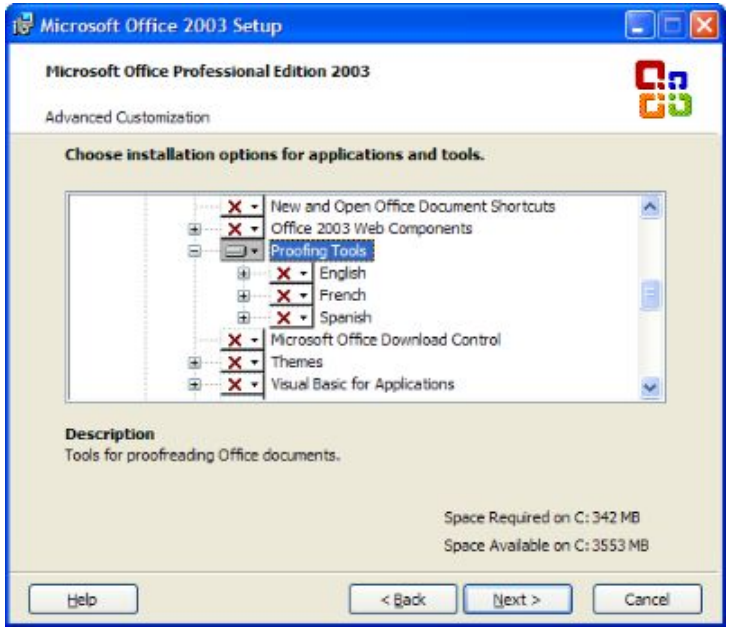
Les Lignes de Produits Logiciels (SPL)



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

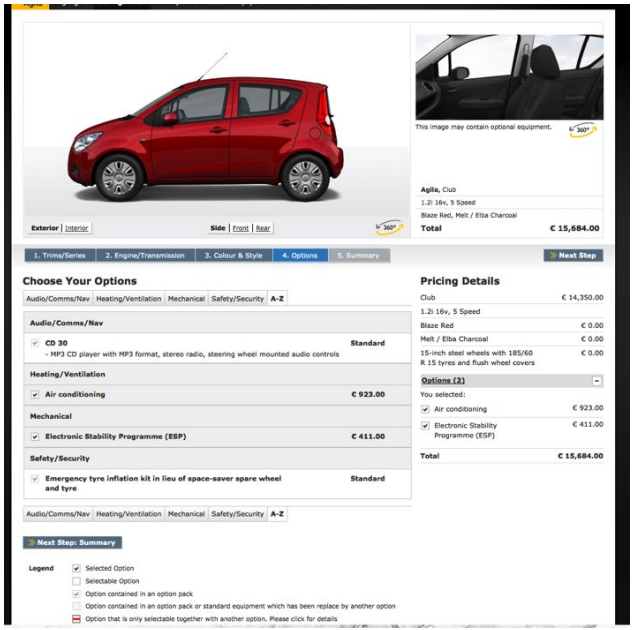
Exemples de lignes de produits

Des produits logiciels…



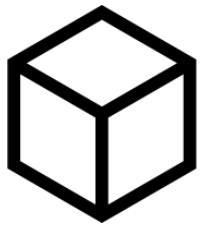
© Microsoft

… mais pas seulement !



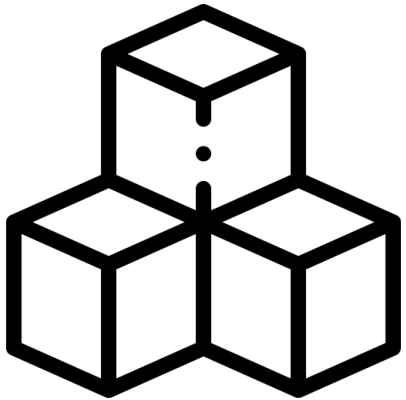
© Opel

Il était une fois...



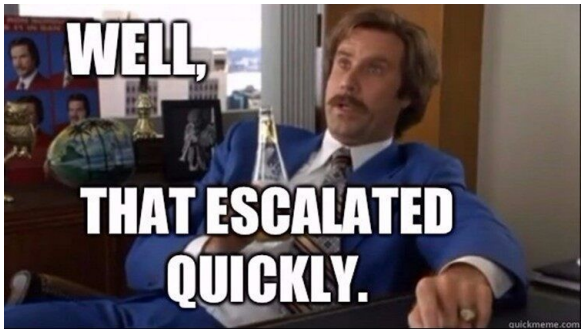
Votre petit projet

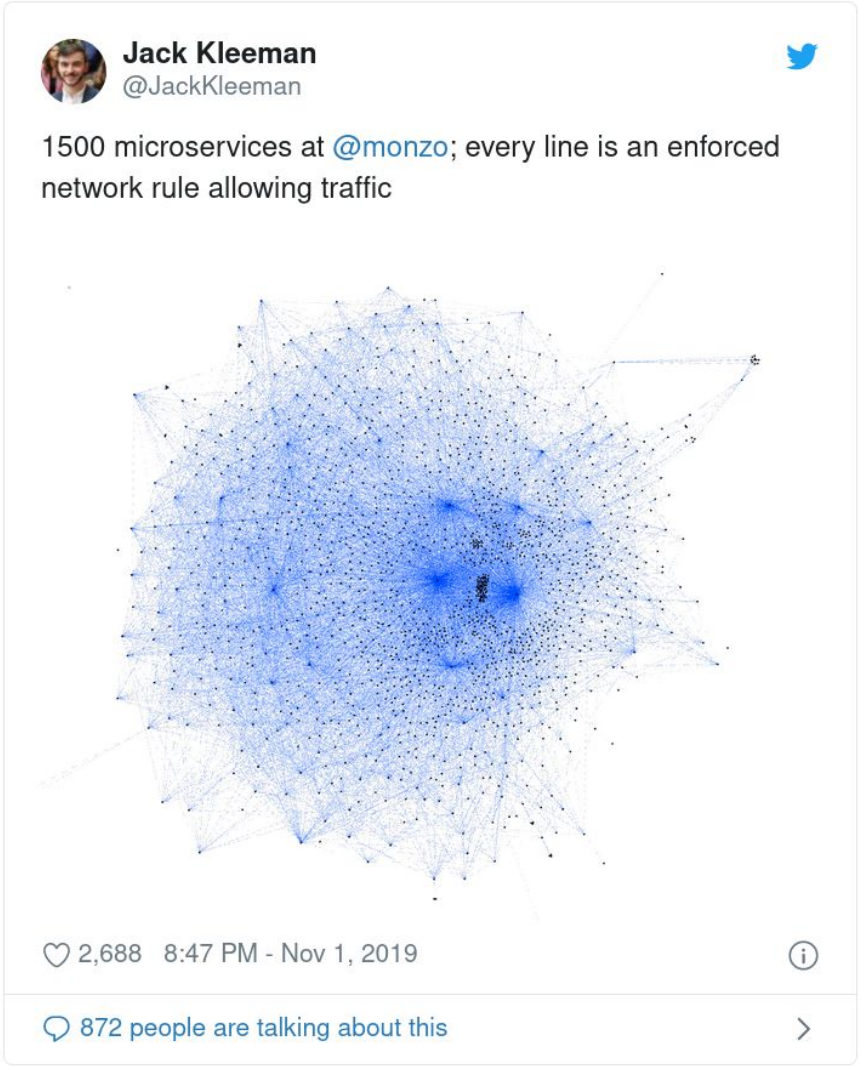
“Ah tiens et si on faisait ça aussi ?”



Votre *“petit”* projet

Et là, c'est le drame !





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

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

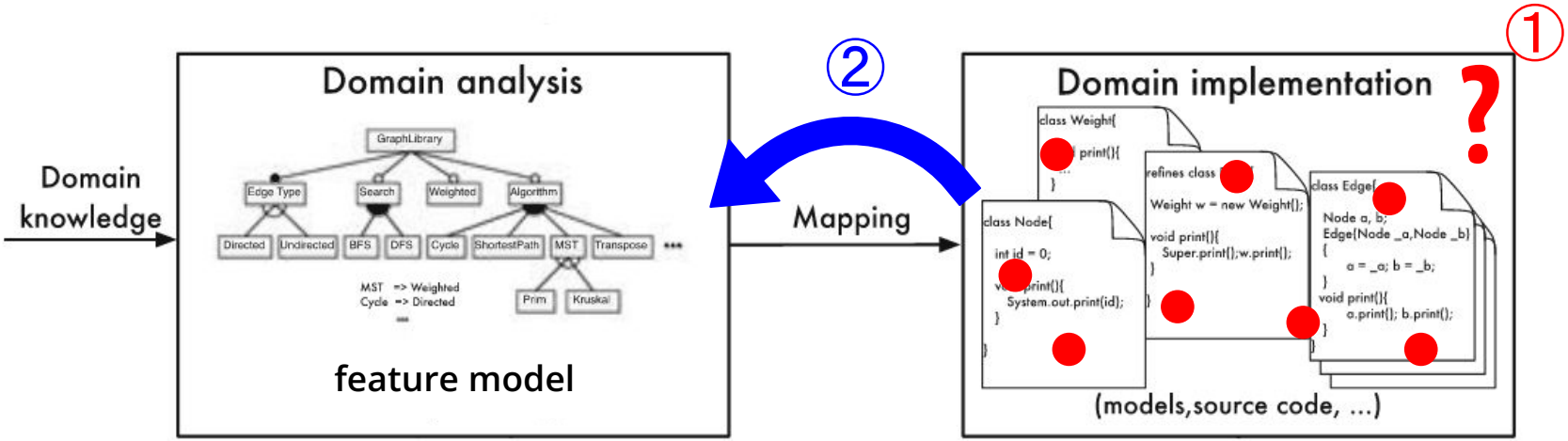
On identifie les implémentations de variabilité présentes pour reconstruire un *feature model* (i.e. features + contraintes entre elles).

Feature model ⟵ Implémentation

⇒ **rétro-ingénierie**

On identifie les implémentations de variabilité présentes

pour les relier à un feature model



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

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

⇒ 1ère 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 !

⇒ 2e 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**

⇒ 3e 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

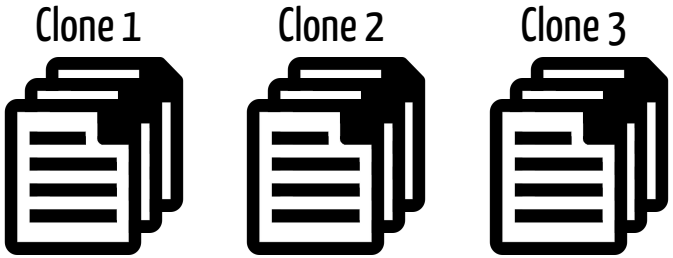
Source : Image by Mila Kusmenko from Pixabay

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 ?

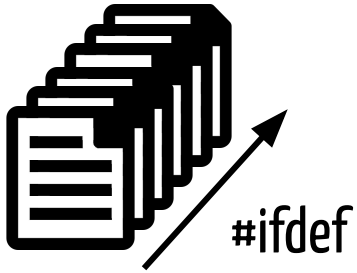
Non, car la variabilité d'un système OO est gérée la plupart du temps en une **unique base de code**.

Étape 1 : que sait-on faire ?

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

Technique 2 : directives de préprocesseur / 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 ?

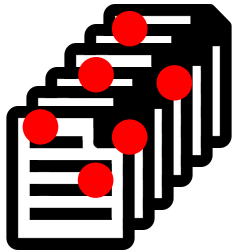
Pas toujours, car la majorité des systèmes OO **n'utilisent pas ces mécanismes**.

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 ?

On observe manuellement !

Points de variation et variantes

vp_shape

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

v_circle

```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;
  }
}
```

v_rectangle

```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);
  }
```

vp_draw

```java
  public void draw(int x, int y) {
  // rectangle at (x, y, width, length)
  }
  public void draw(Point p) {
  // rectangle at (p.x, p.y, width, length)
  }
}
```

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

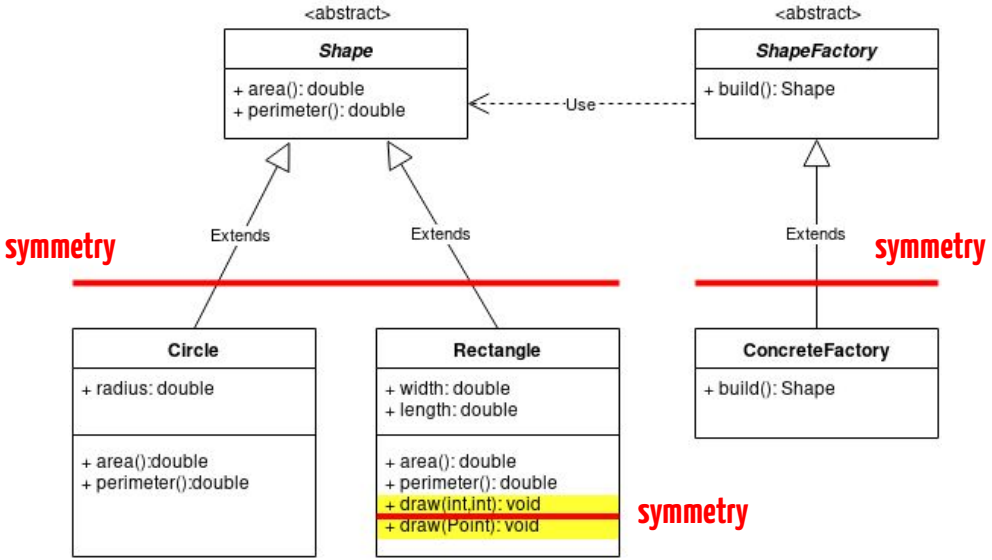
⇒ Nouvelle question : Comment identifier ces implémentations de variabilité ?

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 ?

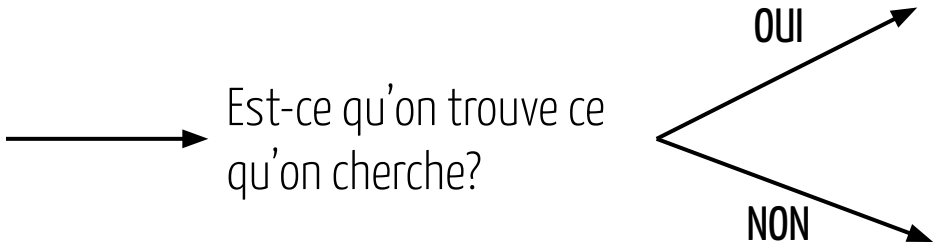


Xhevahire Tërnava, Johann Mortara, and Philippe Collet. 2019. Identifying and Visualizing Variability in Object-Oriented Variability-Rich Systems. In 23rd International Systems and Software Product Line Conference - Volume A (SPLC '19), September 9–13, 2019, Paris, France. ACM, New York, NY, USA, 12 pages.

Étape 3 : vérification de l'intuition

On commence petit !

Application de l'intuition sur un **petit** projet qui a les propriétés recherchées



Comment on choisit un projet ?

Caractéristiques recherchées :

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

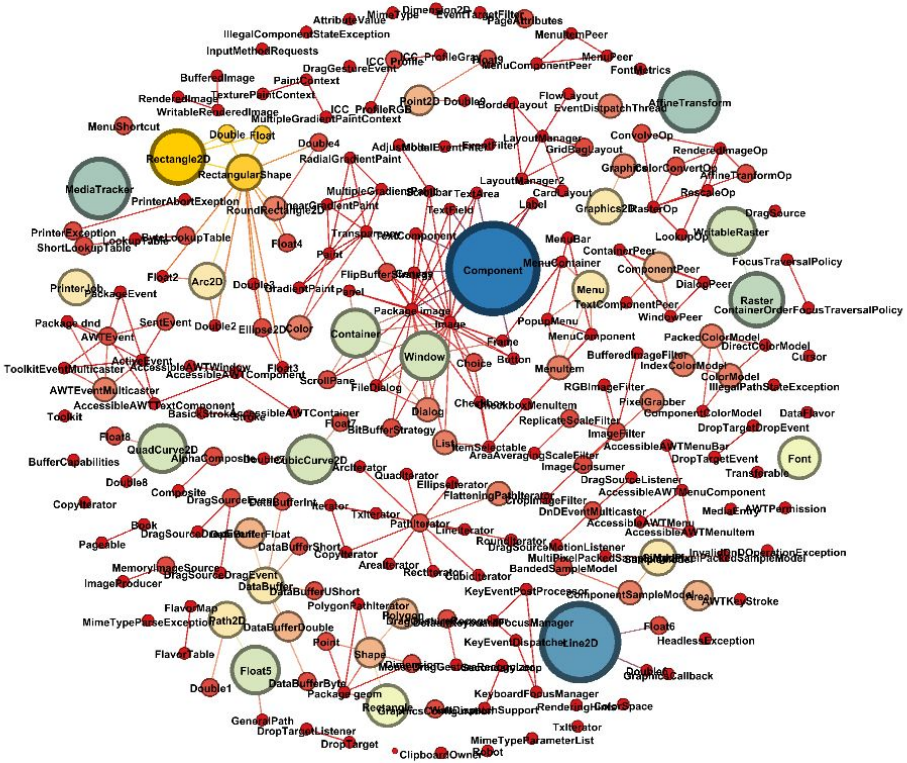
Candidat choisi : **Java AWT**

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

Premières expérimentations

- Identification manuelle des points de variation
- Première visualisation de la variabilité sous la forme d'un graphe **montre des zones de haute densité de variabilité**

C'est ce qu'on cherchait !



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.

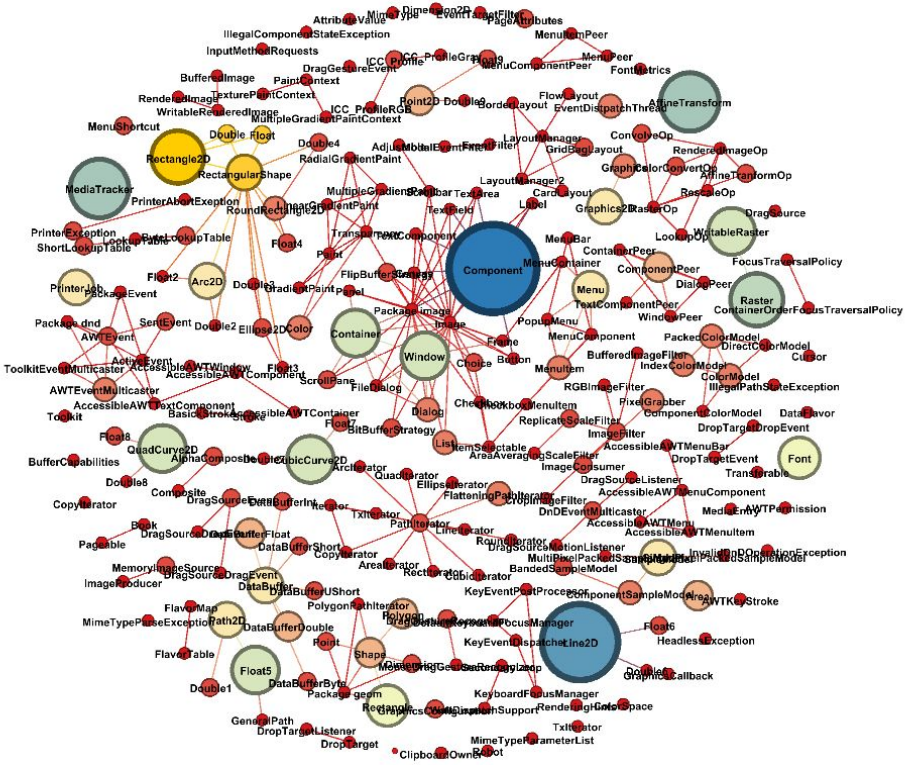
2. Est-ce que les implémentations de variabilité identifiées correspondent vraiment à de la variabilité ? **Bonne question…**

3. Comment indiquer à un utilisateur les zones de forte densité d'un projet ? Sur un projet, un graphe semble approprié.

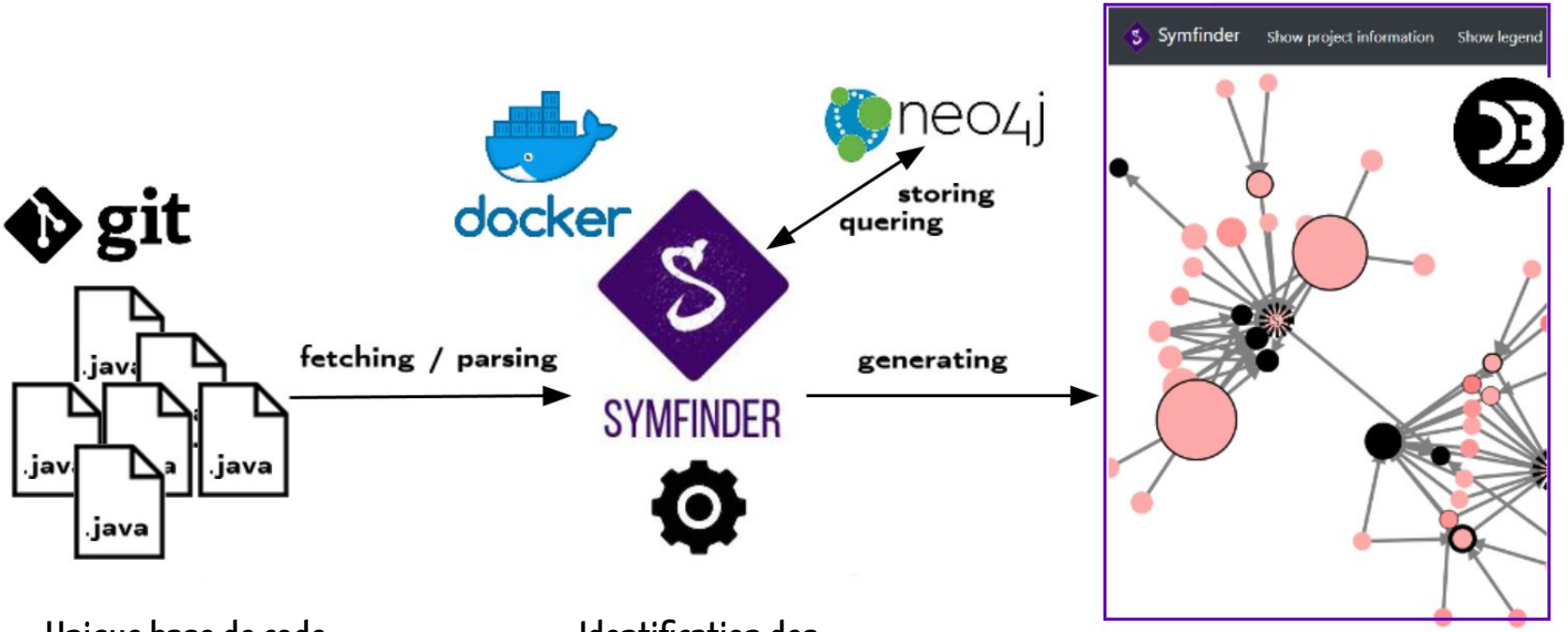
Premières expérimentations

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

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



symfinder



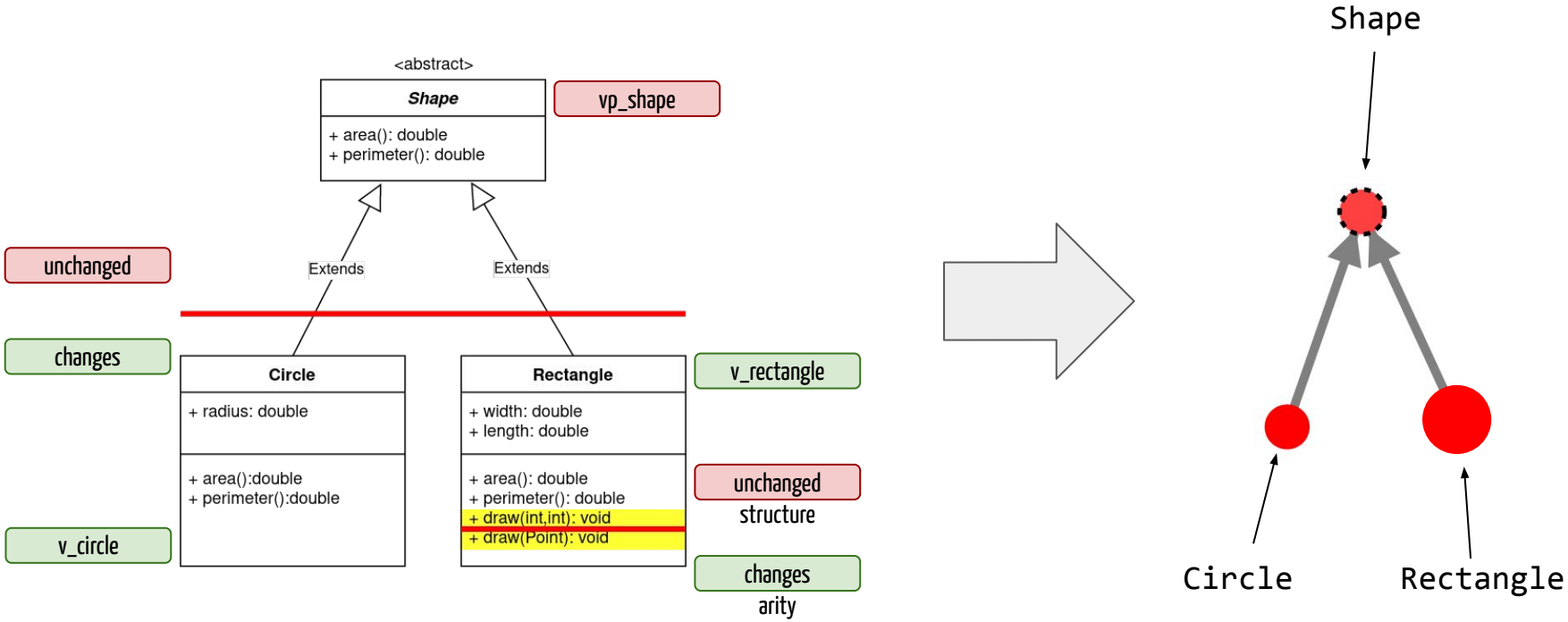
Unique base de code d'un système variable

Identification des symétries

Visualisation des *vp-s*

Johann Mortara, Xhevahire Tërnava, and Philippe Collet. 2019. symfinder: A Toolchain for the Identification and Visualization of Object-Oriented Variability Implementations. In 23rd International Systems and Software Product Line Conference - Volume B (SPLC '19), September 9–13, 2019, Paris, France. ACM, New York, NY, USA, 6 pages.

Visualisation d'un petit exemple



Johann Mortara, Xhevahire Tërnava, and Philippe Collet. 2019. symfinder: A Toolchain for the Identification and Visualization of Object-Oriented Variability Implementations. In 23rd International Systems and Software Product Line Conference - Volume B (SPLC '19), September 9–13, 2019, Paris, France. ACM, New York, NY, USA, 6 pages.

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

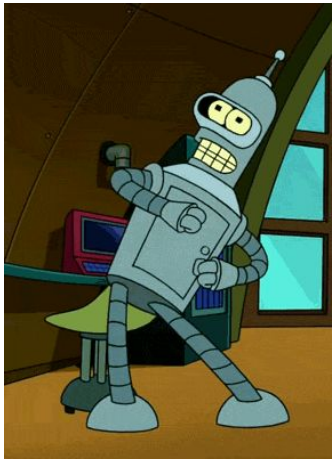
JFreeChart

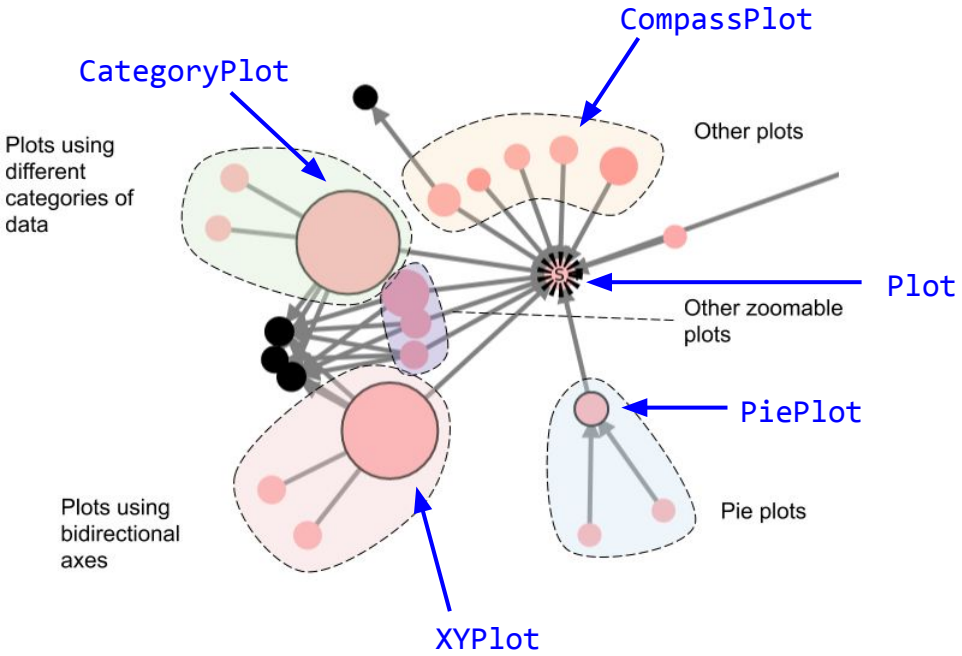
Bibliothèque permettant de tracer **différents types de graphiques**

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

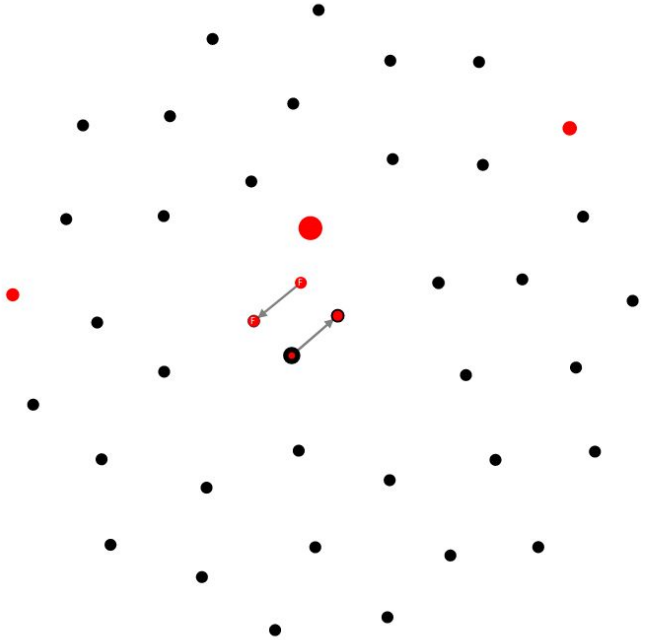
Outil de configuration de projets à partir d'un choix de pile technologique

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

Système	# LoC	# *vp*-s	# variantes
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

Système	# LoC	# *vp*-s	# variantes
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 ≠ # LoCs

Système	# LoC	# *vp*-s	# variantes
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

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	**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**

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 ?

Source : Image by StockSnap from Pixabay

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 ?

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

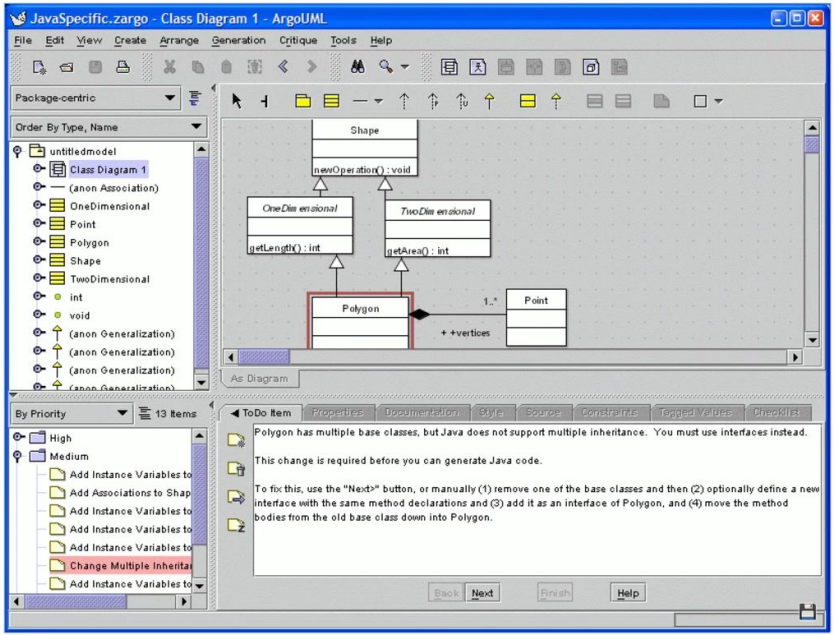
- le rêve : un feature model
- au moins : apping implémentation ⟷ fonctionnalité existant

Où trouver un tel projet ?

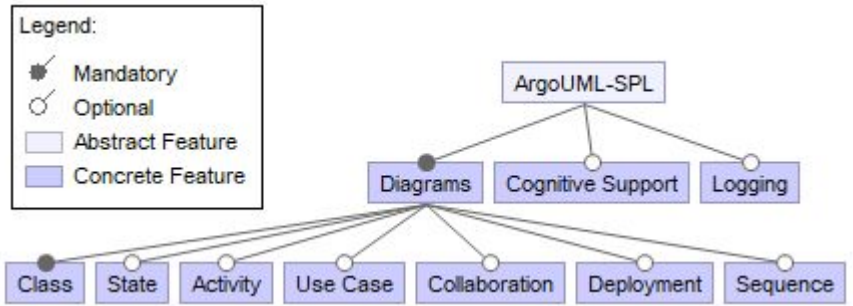
Rechercher dans l'état de l'art !

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

ArgoUML-SPL [Couto2011]

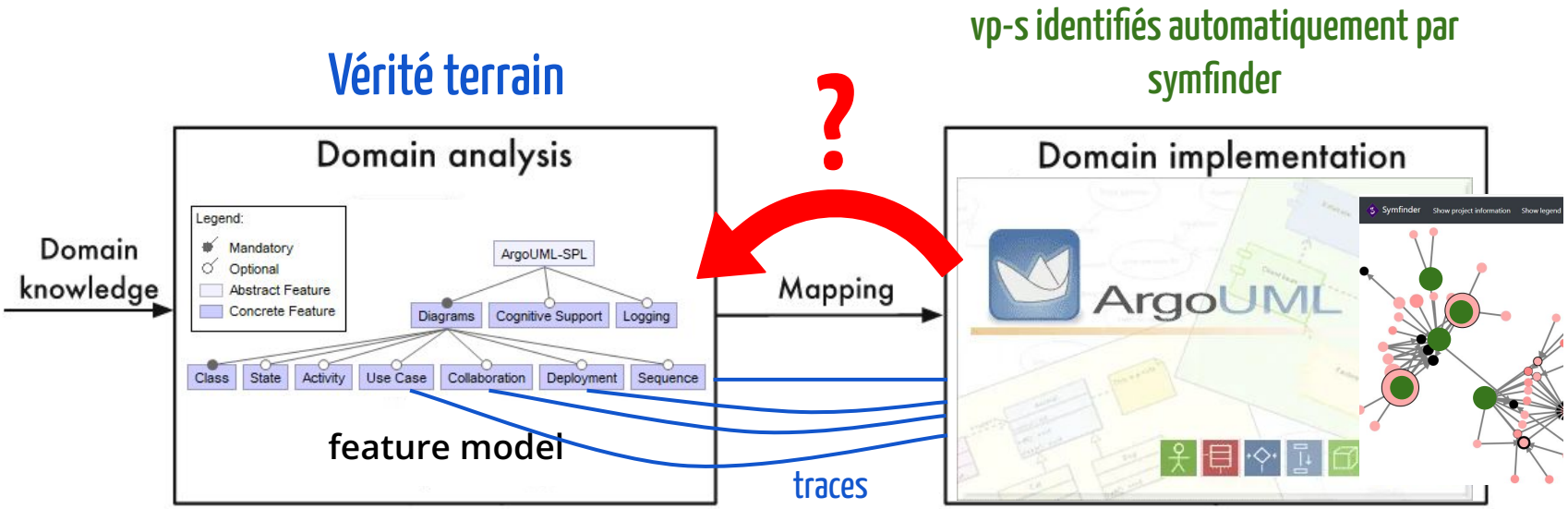


Éditeur d'ArgoUML



Feature model d'ArgoUML-SPL

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



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

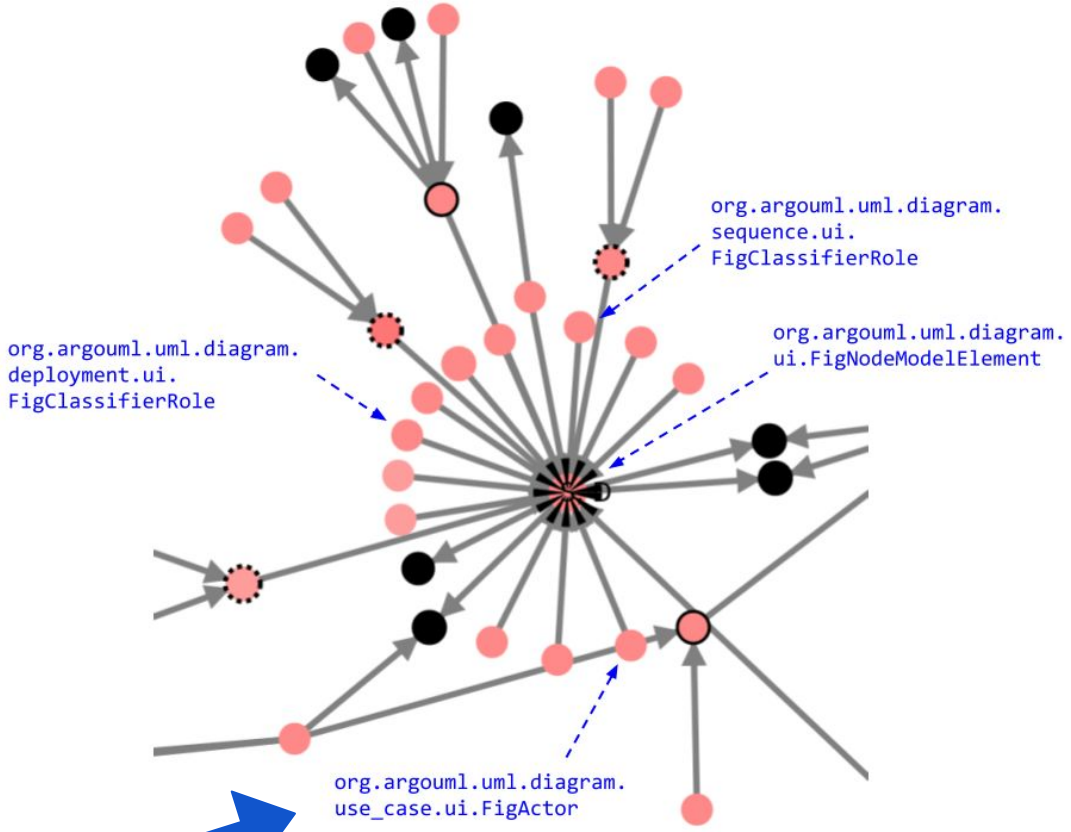
Que voit-on ?

Fonctionnalité : *Sequence*

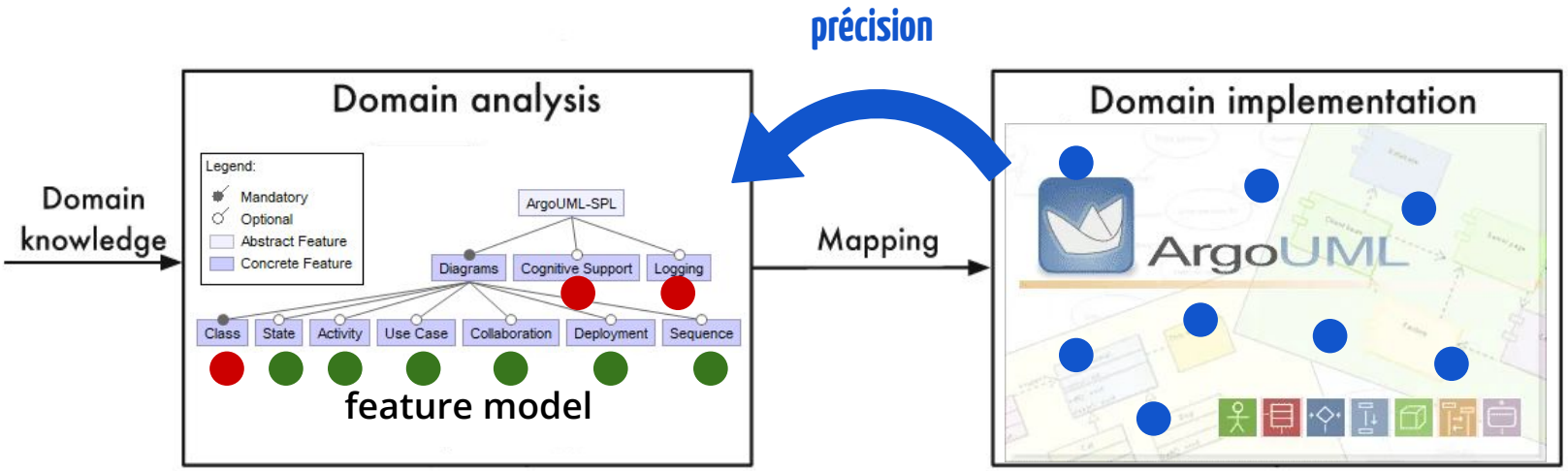
```
//#if defined(SEQUENCEDIAGRAM)
//@#$LPS-SEQUENCEDIAGRAM:GranularityType:Package
public class FigClassifierRole extends FigNodeModelElement
```

Fonctionnalité : *Use Case*

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



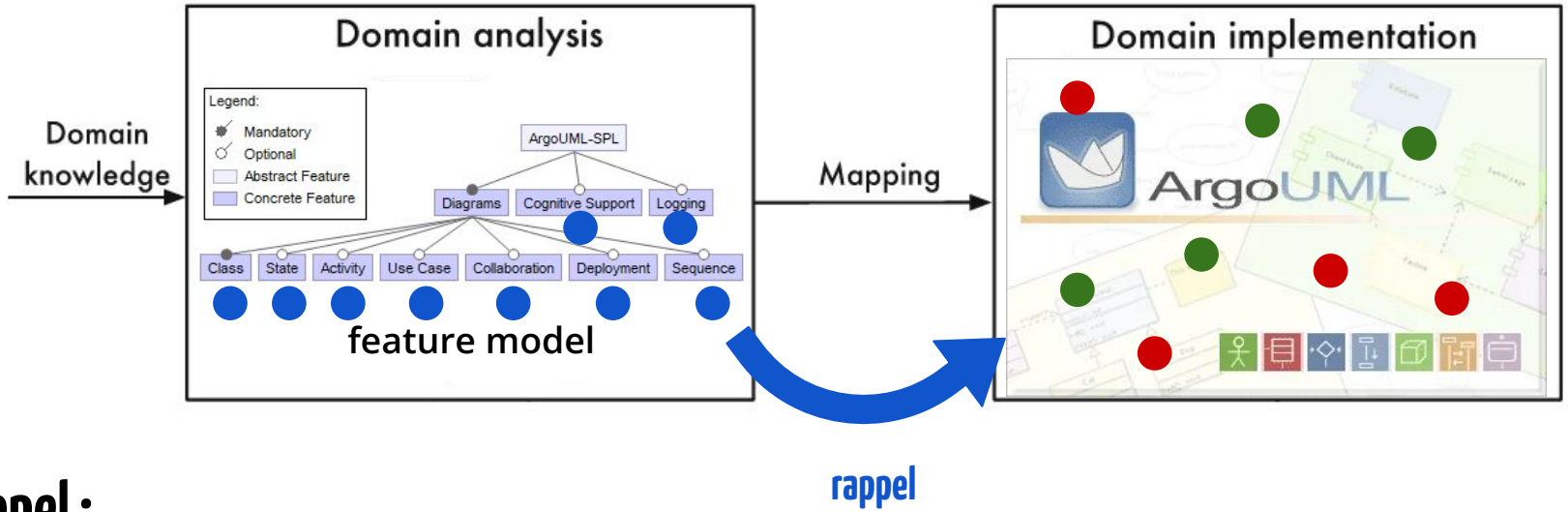
Pertinence des *vp-s*



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

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

Johann Mortara, Xhevahire Tërnava, and Philippe Collet. 2020. Mapping Features to Automatically Identified Object-Oriented Variability Implementations: The case of ArgoUML-SPL. In Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS '20), February 5–7, 2020, Magdeburg, Germany. ACM, New York, NY, USA, 9 pages.

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 :

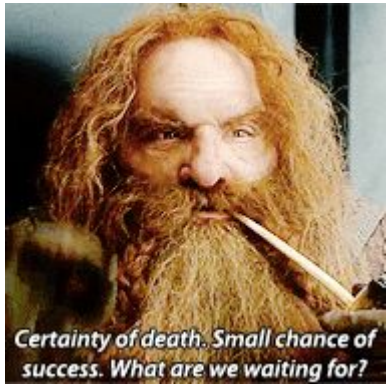
Identification, visualisation et gestion de variabilité au sein de

grands systèmes orientés objets hautement variables

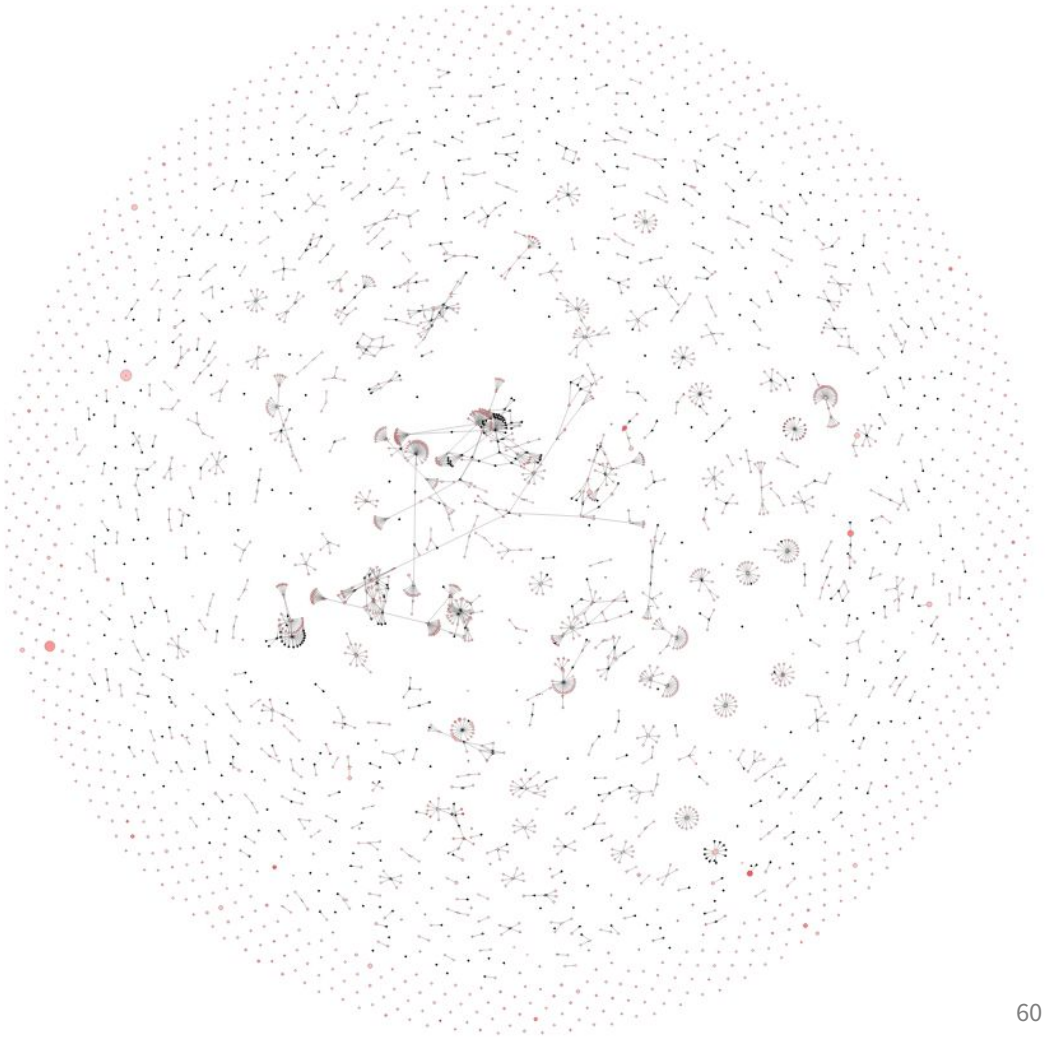
Plus grand système étudié : ArgoUML (179k LoC) ⟶ il nous faut plus grand !

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 !

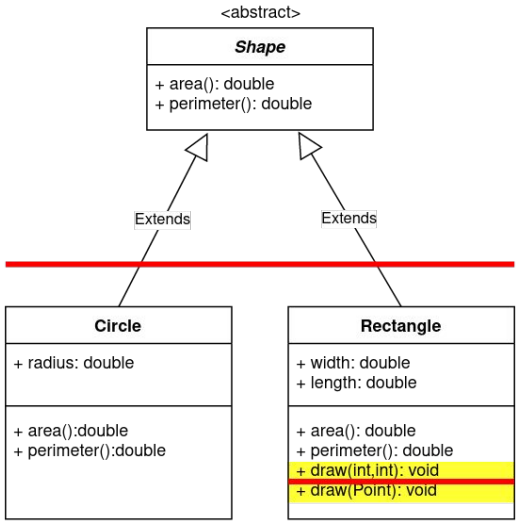
3. Comment indiquer à un utilisateur les zones de forte densité d'un projet ? Un graphe semble approprié sur de petits systèmes mais est moins lisible sur de grands systèmes.

3.3

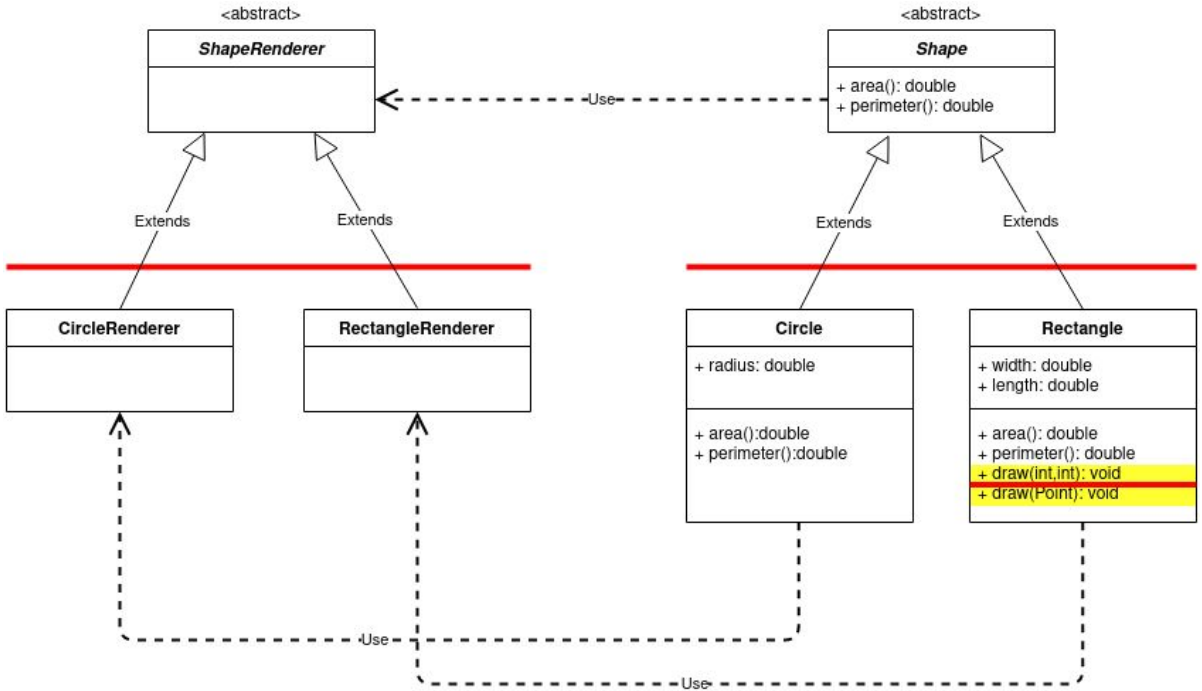
Sommes-nous sur le bon chemin ?

Source : Image by ivalbak from Pixabay

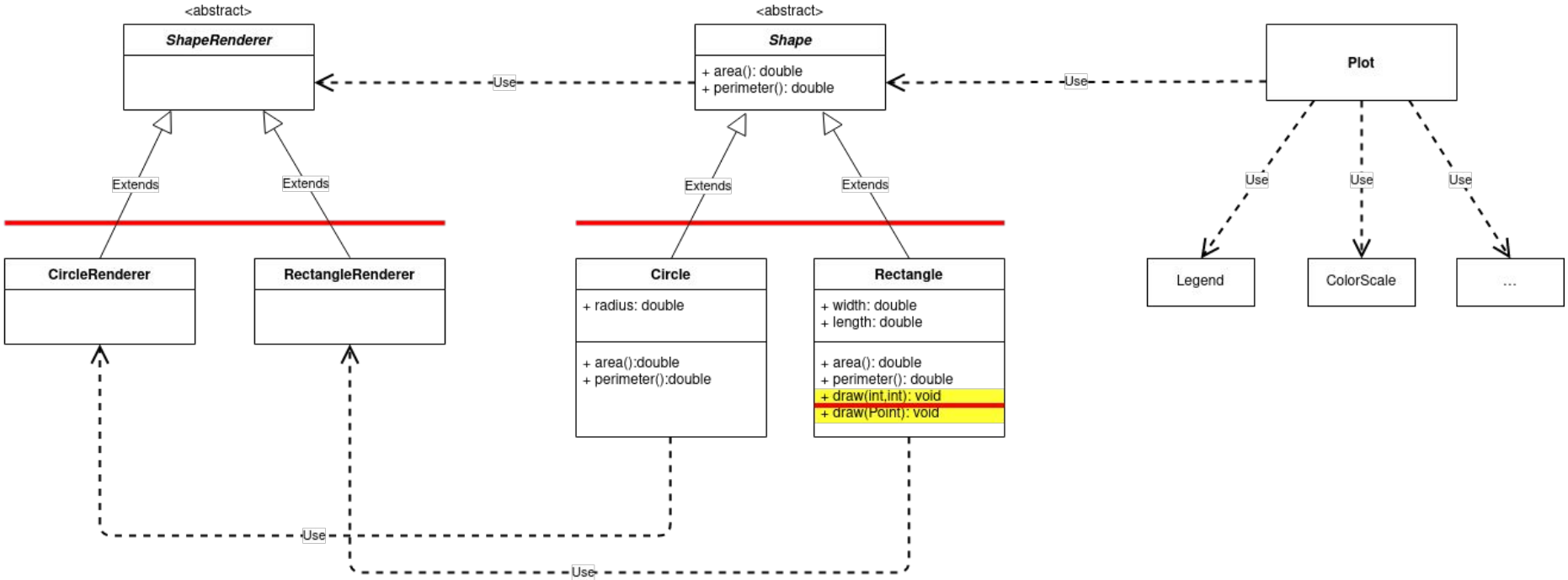
Back to the origins...

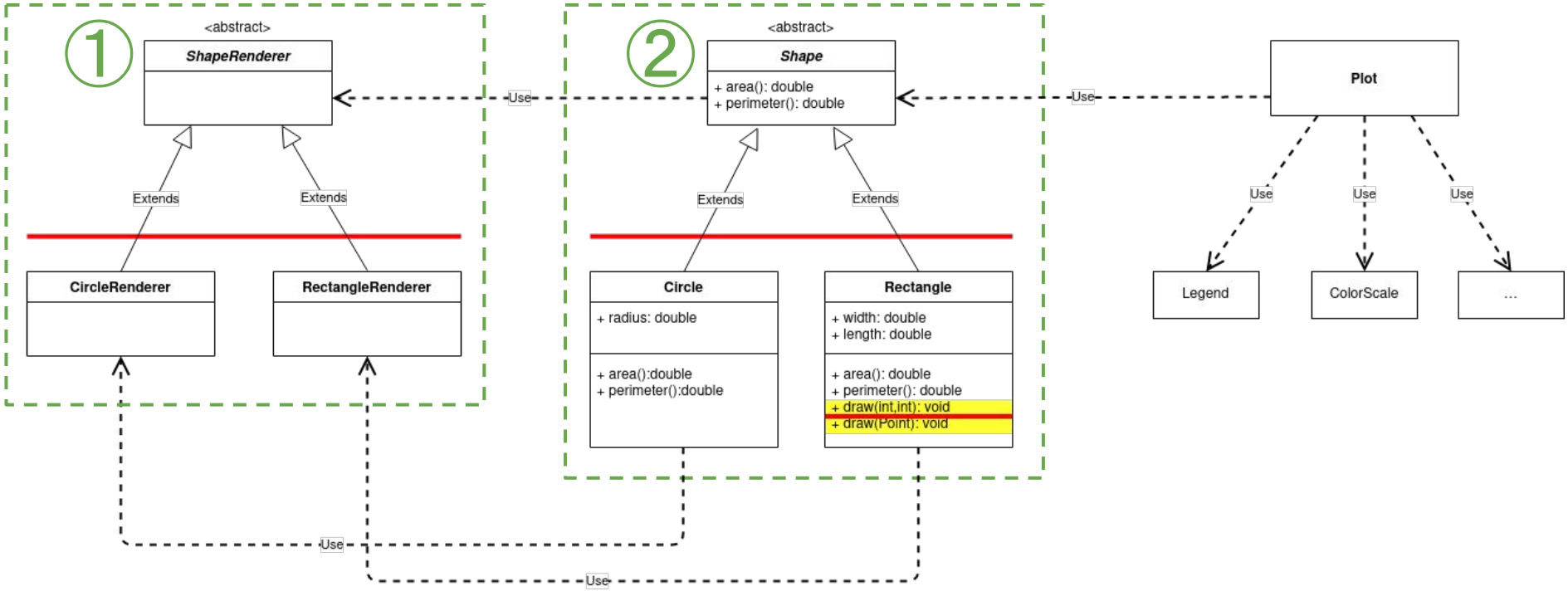


Variability implementations *in the wild*

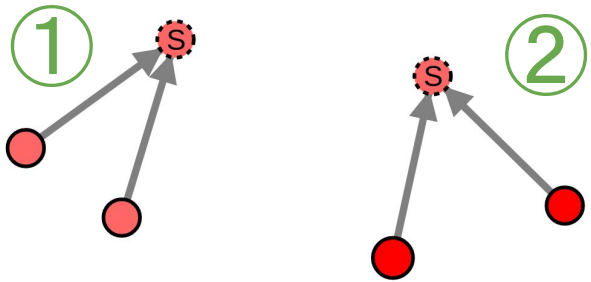


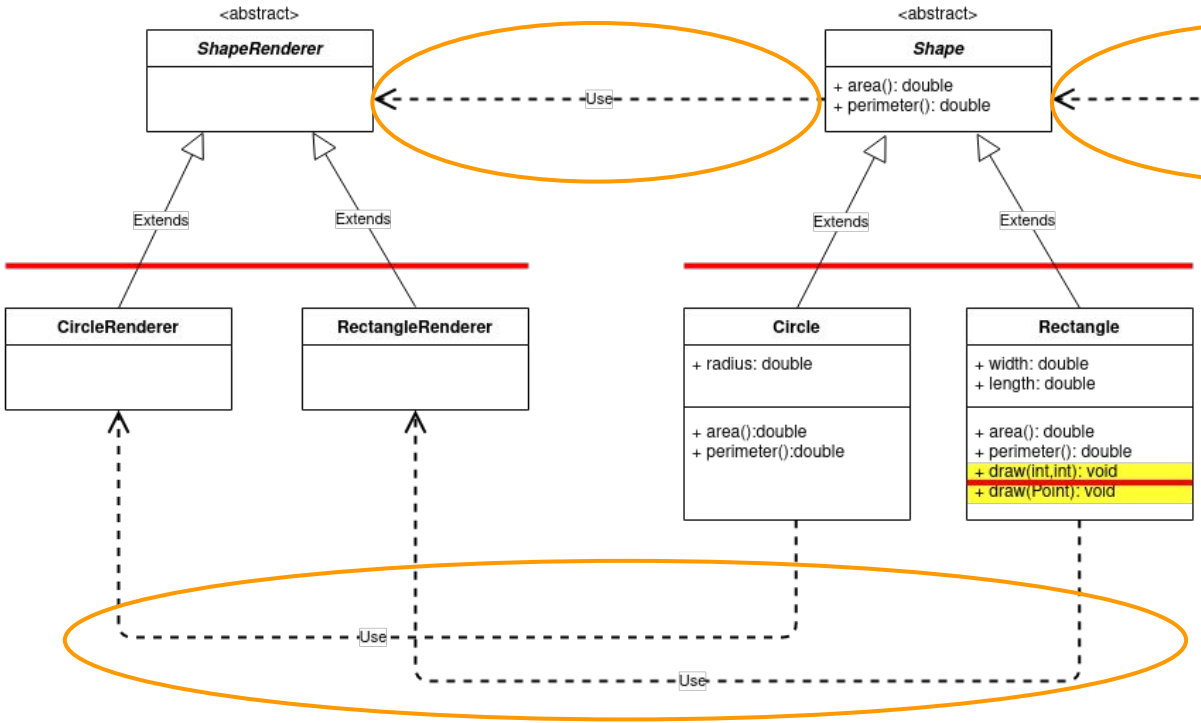
Variability implementations *in the wild*





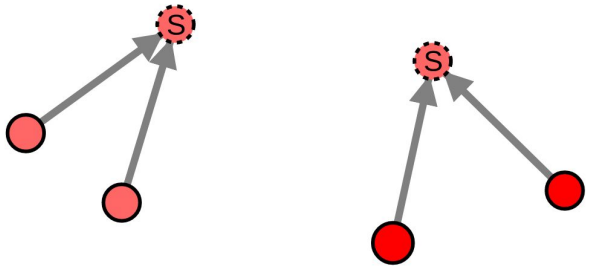
Visualisation générée par symfinder

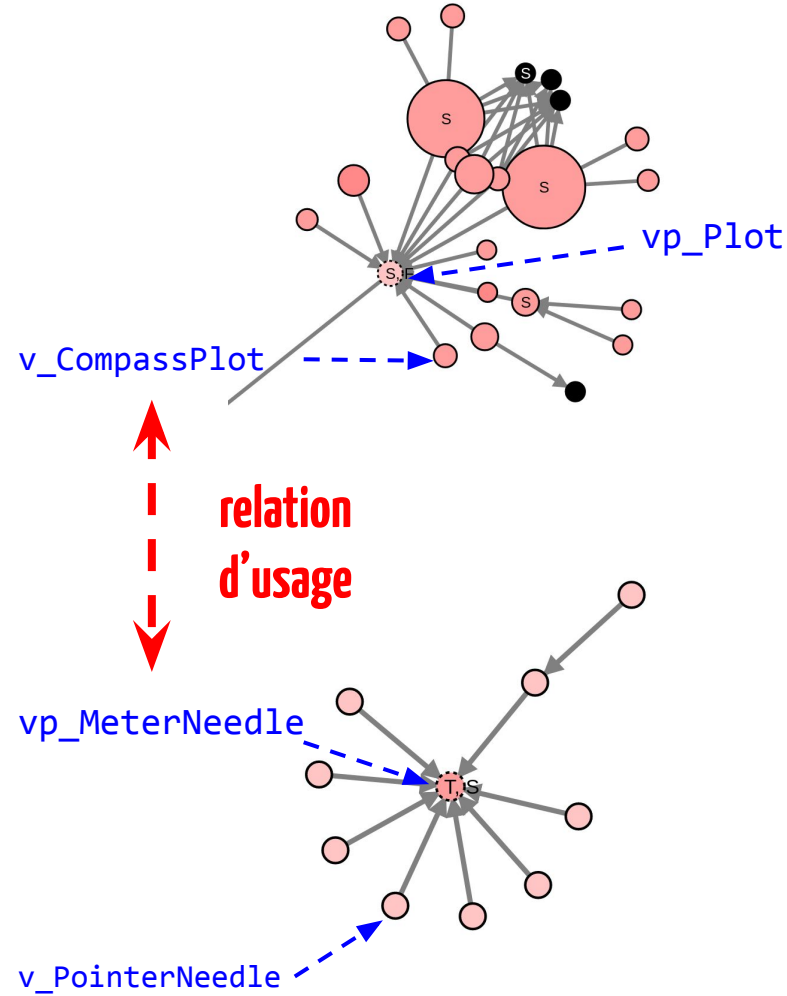




Pas utilisées par symfinder!

Visualisation générée par symfinder

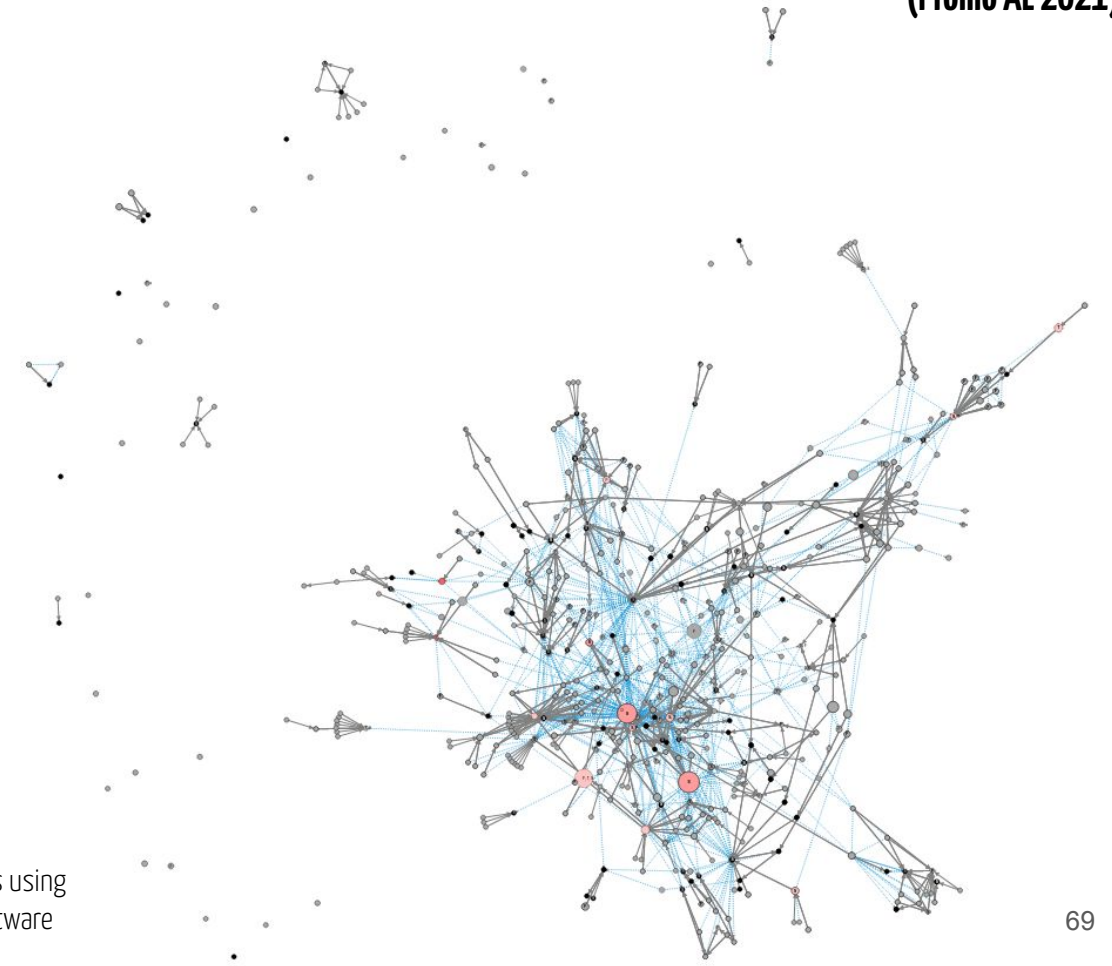



vp_Plot
v_CompassPlot
relation
d'usage
vp_MeterNeedle
v_PointerNeedle

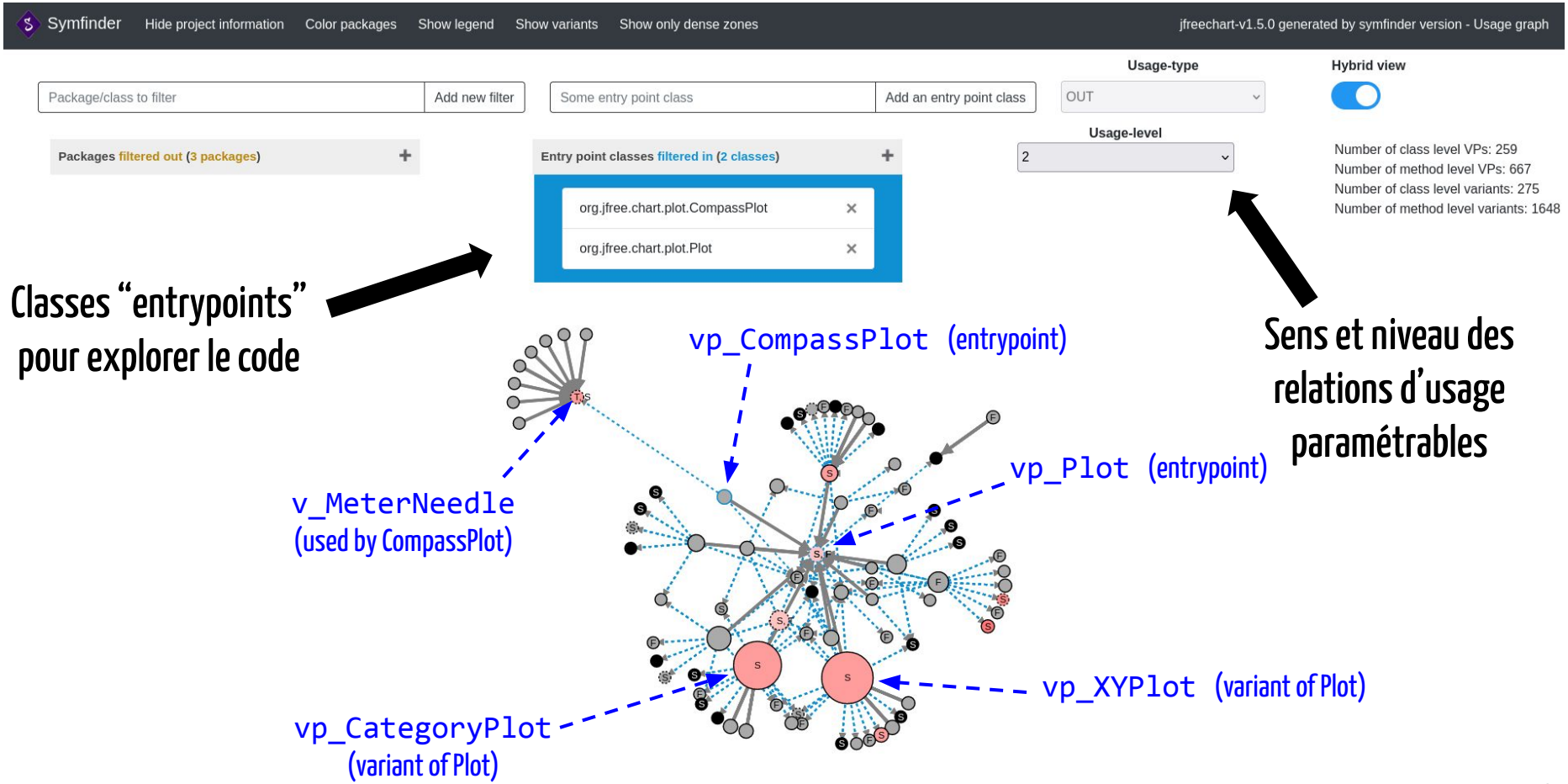
symfinder-2

Dédicace à Florian AINADOU, Paul-Marie DJEKINNOU et Djotiham NABAGOU !
(Promo AL 2021)

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



Johann Mortara, Xhevahire Tërnava, Philippe Collet, Anne-Marie Dery-Pinna. Extending the Identification of Object-Oriented Variability Implementations using Usage Relationships. SPLC 2021 - 25th ACM International Systems and Software Product Line Conference, Sep 2021, Leicester, United Kingdom. pp.1-8


Symfinder
Hide project information
Color packages
Show legend
Show variants
Show only dense zones
jfreechart-v1.5.0 generated by symfinder version - Usage graph
Package/class to filter
Add new filter
Some entry point class
Add an entry point class
Usage-type
OUT
Hybrid view
Usage-level
2
Packages filtered out (3 packages)
Entry point classes filtered in (2 classes)
org.jfree.chart.plot.CompassPlot
org.jfree.chart.plot.Plot
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
vp_CompassPlot (entrypoint)
vp_Plot (entrypoint)
v_MeterNeedle (used by CompassPlot)
vp_XYPlot (variant of Plot)
vp_CategoryPlot (variant of Plot)

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	*symfinder-2* ≥ 10 *v*-s ≤ 3 hops	*symfinder-2* ≥ 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

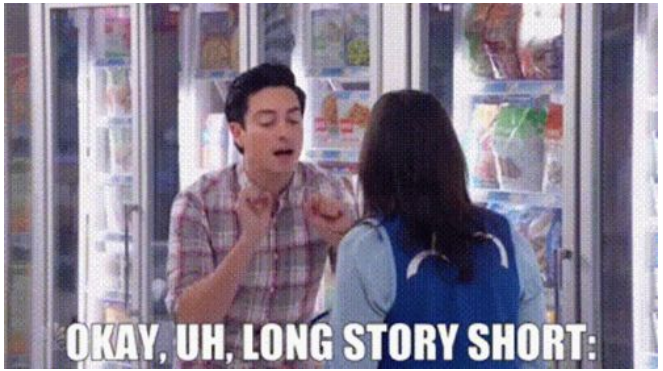
Source : Image by FunkyFocus from Pixabay

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 **utilisent** principalement **les features du système**

mais on ne les a pas :(

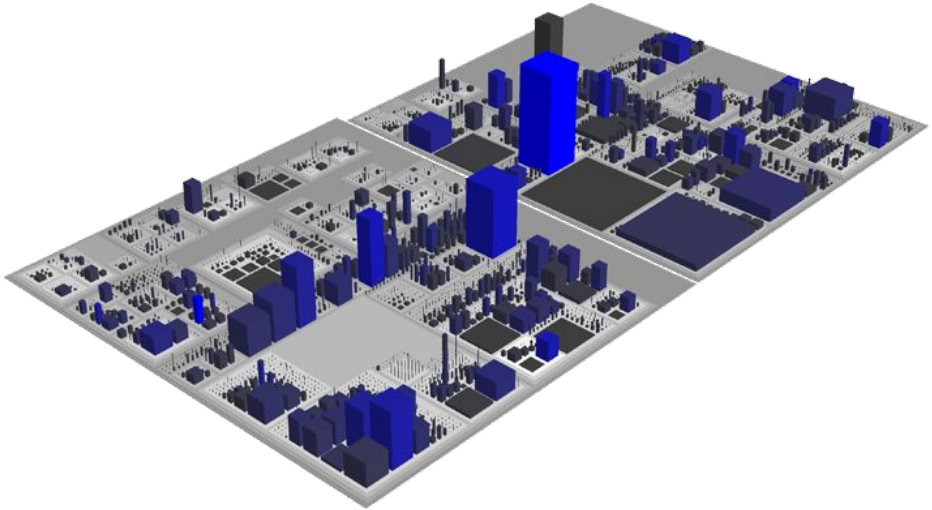
⇒ on ne peut rien réutiliser

mais… on peut le faire nous-mêmes !

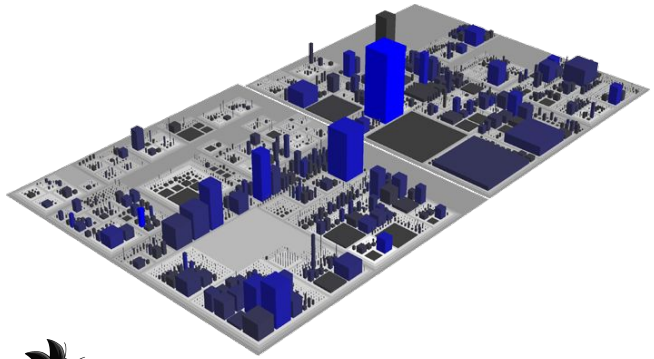
Q2: CodeCity [Wettel2007] et Evo-Street [Steinbrückner2013]

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

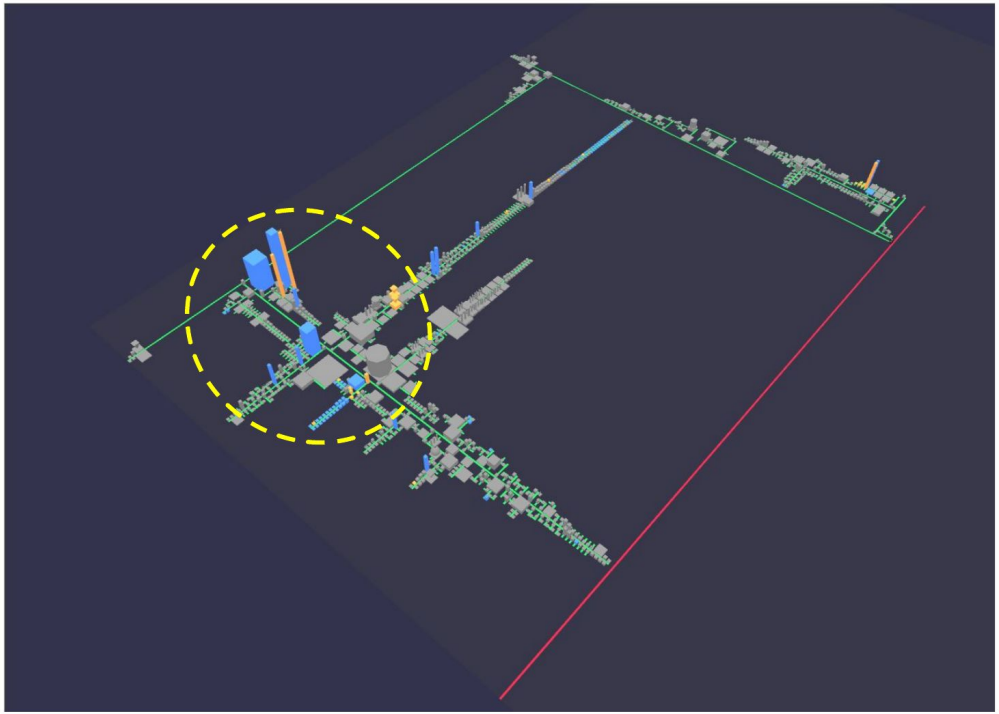
From **CodeCity** to **VariCity**

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**.

Dédicace à Paul-Marie DJEKINNOU, Florian FOCAS et François RIGAUT ! (Promo AL 2021)

Johann Mortara, Philippe Collet, Anne-Marie Dery-Pinna. Visualization of Object-Oriented Variability Implementations as Cities. 9th IEEE Working Conference on Software Visualization (VISSOFT 2021), Sep 2021, Luxembourg (virtual), Luxembourg. ff10.1109/VISSOFT52517.2021.00017ff. ffhal03312487

Passage à l'échelle de la visualisation ✔



Sous-partie du package java de NetBeans 12.2

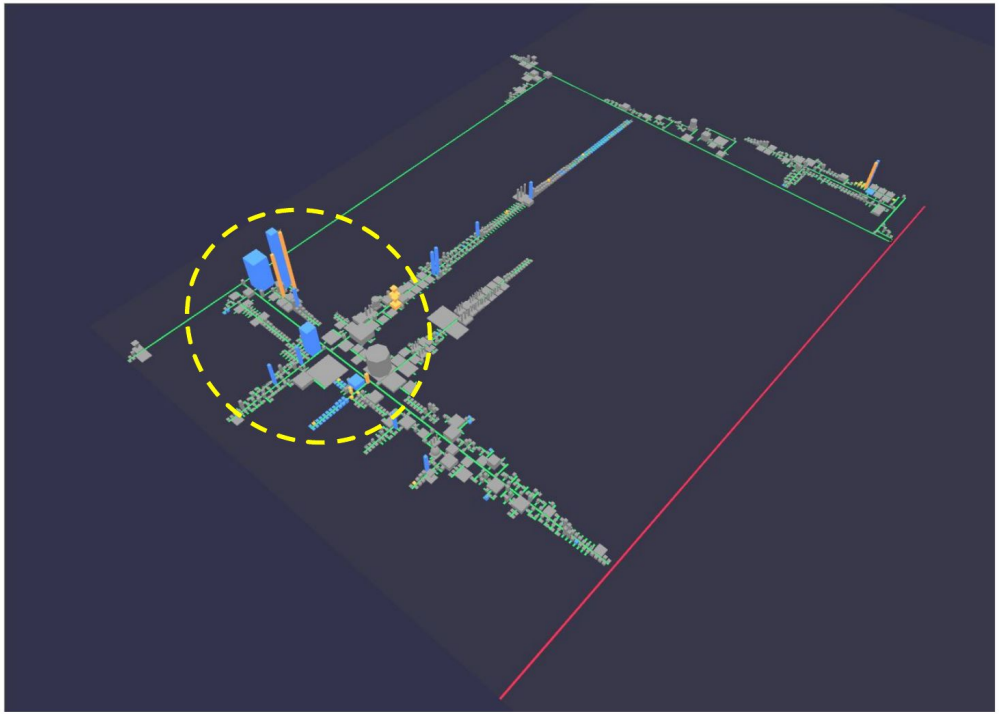
Nos sous-questions

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 ✔



Sous-partie du package java de NetBeans 12.2

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

Évaluation de VariCity

Est-ce que VariCity aide la compréhension de la variabilité OO comparé à des outils habituels ?

Évaluation de VariCity

Est-ce que VariCity aide la compréhension de la variabilité OO comparé à des outils habituels ?

Est-il plus **rapide** et **facile** de résoudre des tâches de compréhension de la variabilité ?

Évaluation de VariCity

Est-ce que VariCity aide la compréhension de la variabilité OO comparé à des outils habituels ?

Est-il plus **rapide** et **facile** de résoudre des tâches de compréhension de la variabilité ?

IDE

Protocole expérimental

Format : Expérience en environnement contrôlé
Population : promo SI5 AL 2021 (49 étudiants)
Système objet : JFreeChart
Traitements : VariCity / IDE + CSV data

Résultats

Les sujets utilisant VariCity résolvent les tâches **plus rapidement** et **les trouvent plus faciles** que les sujets utilisant un IDE, **mais ils auraient préféré avoir accès au code**.

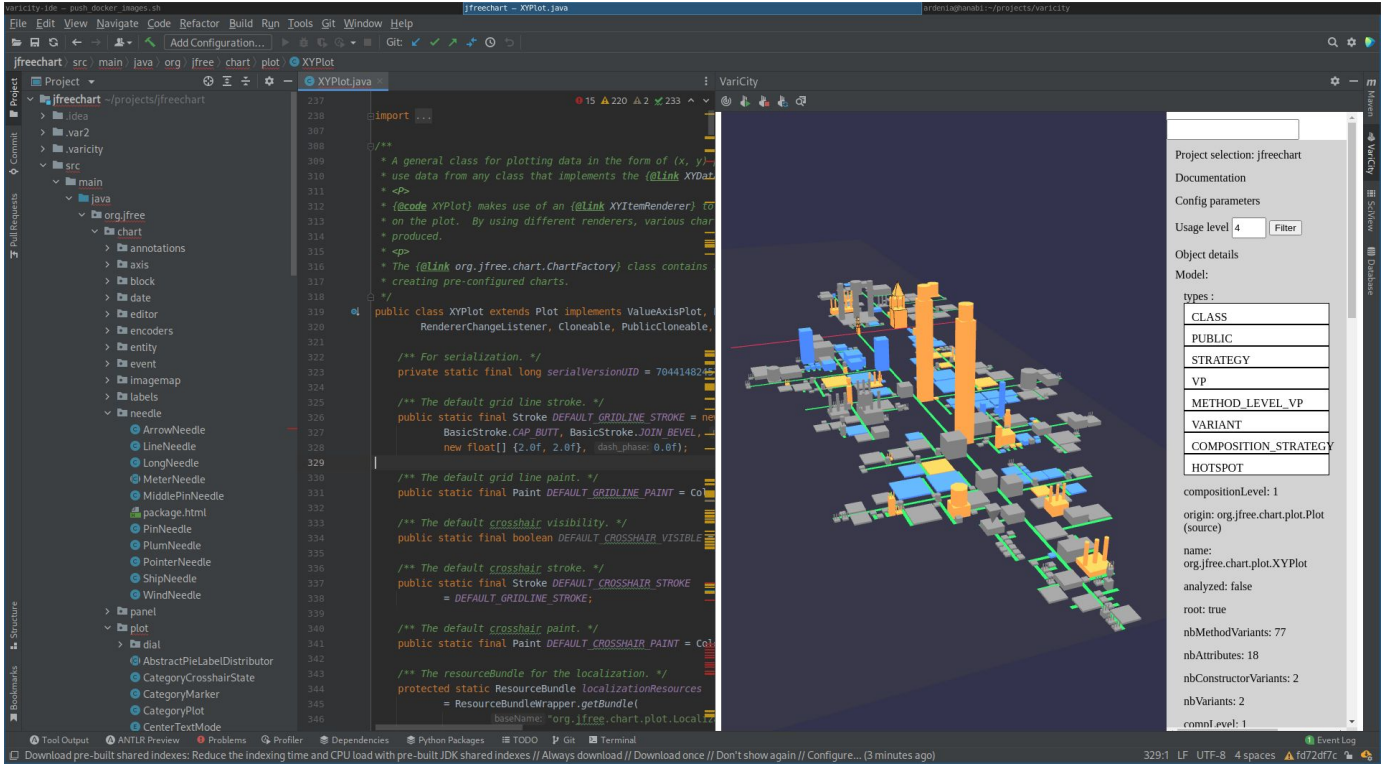
CodeCity et Evo-Streets utilisent des **métriques de qualité du code**

⇒ visualisations intégrées à **SonarQube**

⇒ **code et métriques au même endroit**

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

⇒ visualisation intégrée à **un IDE (IntelliJ)**



Dédicace à João BRILHANTE, Charly DUCROCQ et Ludovic MARTI (SI5 AL 2021) !

Et la qualité ?

Dédicace à Patrick ANAGONOU, Guillaume SAVORNIN et Anton VAN DER TUIJN (SI5 AL 2021) !

Hautement-Variable ✔
Critique ✘

Hautement-Variable ✔
Critique ✔

Hautement-Variable ✔
✔

Critique

1. On distingue les zones variables endettées
2. Pas de corrélation variabilité / qualité

Hautement-Variable ✘
Critique ✔

Johann Mortara, Philippe Collet, and Anne-Marie Dery-Pinna. “Customizable Visualization of Quality Metrics for Object-Oriented Variability Implementations”. In: Proceedings of the 26th ACM International Systems and Software Product Line Conference - Volume A. SPLC ’22. Graz, Austria: ACM, Sept. 2022, pp. 43–54.

Encore d'autres questions !

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), Martin BRUEL (SI5 AL 2021) et Yann BRAULT !

Causalité variabilité ↔ mauvaise qualité ?

Impacts de la variabilité sur d'autres propriétés du système ?

Gérer l'évolution de cette variabilité ?

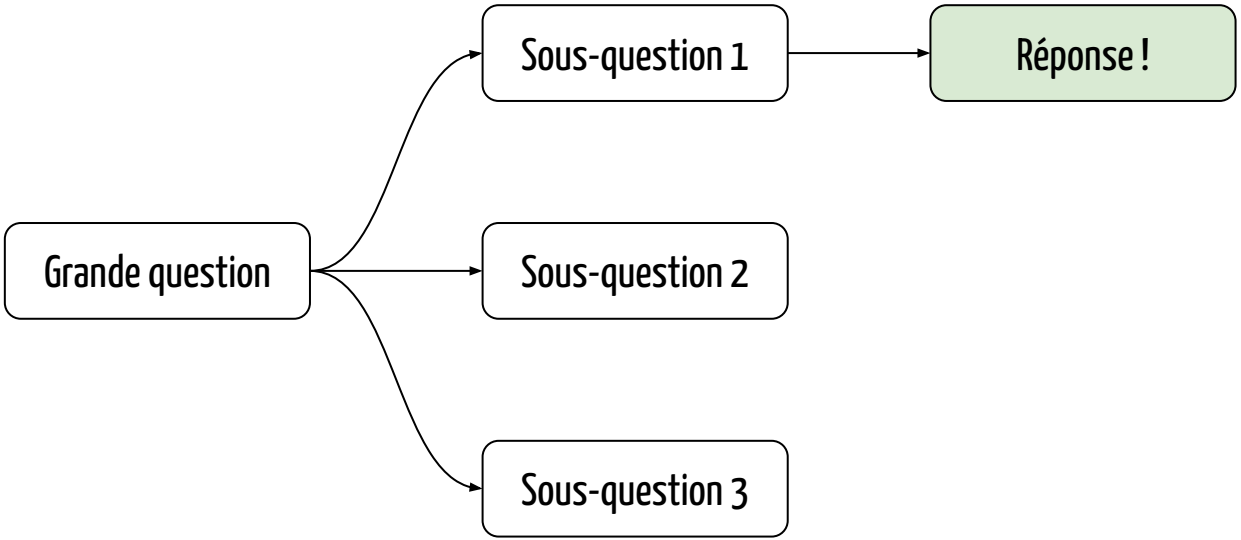
Noyau Linux v4.13 (09/2017) ⟶ v4.20 (12/2018)

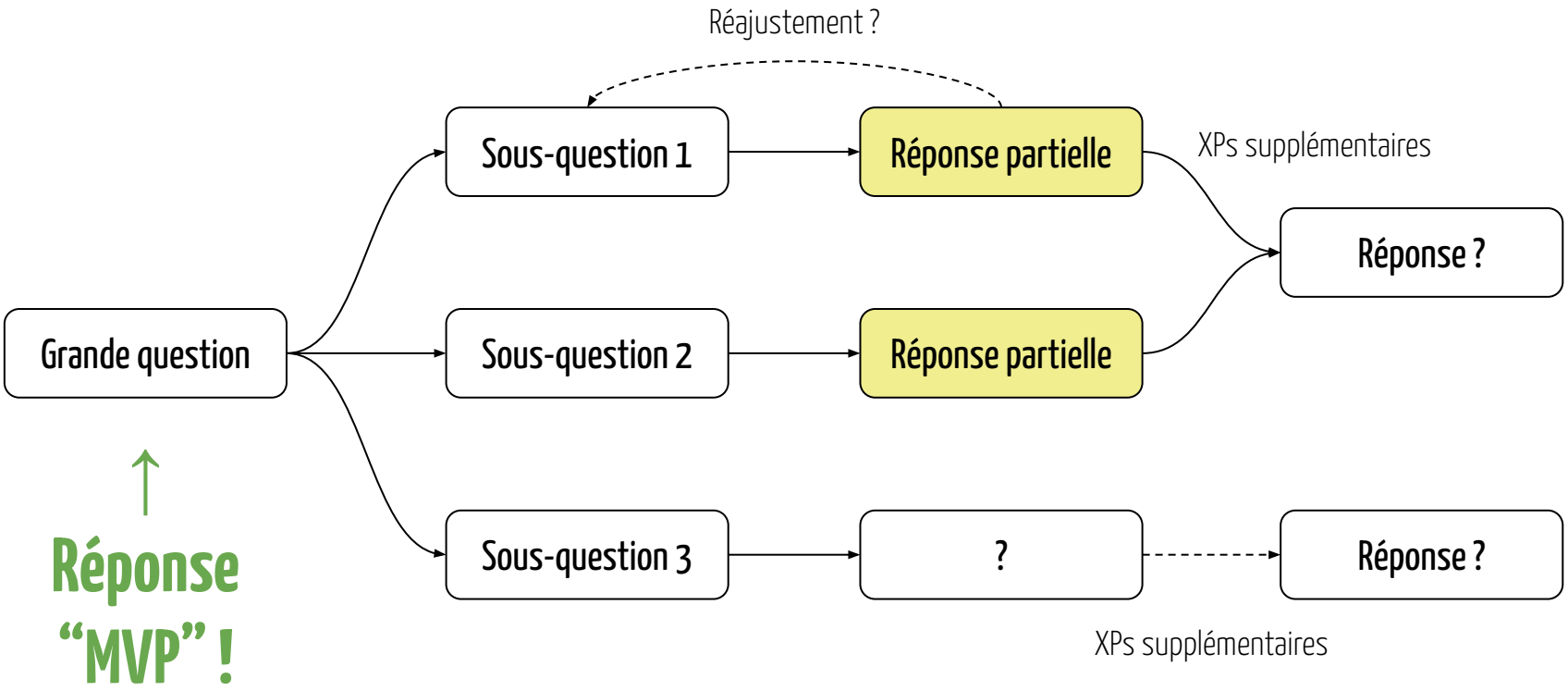
1,189+ 468- features [Martin2021]

4

En résumé

Source : Image by StockSnap from Pixabay


Grande question
Sous-question 1
Sous-question 2
Sous-question 3
Réponse !


Réajustement ?
Sous-question 1
Réponse partielle
XPs supplémentaires
Réponse ?
Grande question
Sous-question 2
Réponse partielle
Sous-question 3
?
Réponse ?
XPs supplémentaires

Réponse "MVP" !

La démarche scientifique est un questionnement permanent !

(82 "?" 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 ≠ 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 !