Visualization of Object-Oriented Variability Implementations as Cities

Johann Mortara – Philippe Collet – Anne-Marie Dery-Pinna

Université Côte d'Azur, CNRS, I3S, France

GT GLIHM — Virtual

November 25, 2021

Highly-variable Systems with a Single Code Base

16.000 options managed in 25M LoC

#ifdef

24.000 different platforms in 2015

Object-orientation

2.000+ options generating variants for platforms, security levels…

#ifdef / object-orientation

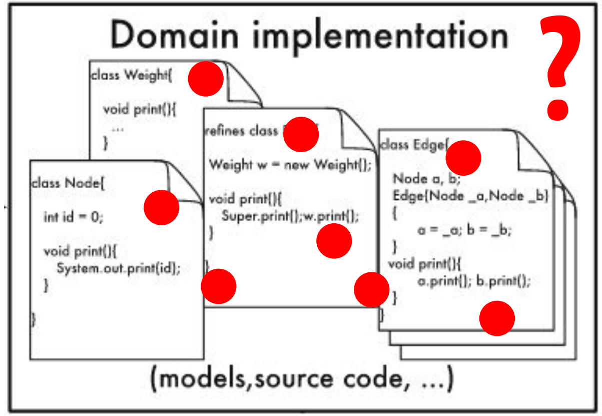
and multiple management techniques…

OO codebases use OO mechanisms to implement variability in a single codebase

- inheritance
- overloading of methods and constructors
- design patterns

Creation of **complex zones** in the system

Undocumented OO variability implementations



```
class Weight{

  void print(){
    ...
  }
```

```
class Node{

  int id = 0;

  void print(){
    System.out.print(id);
  }

}
```

```
refines class

 Weight w = new Weight();

 void print(){
   Super.print();w.print();
 }

}
```

```
class Edge{

  Node a, b;
  Edge(Node _a,Node _b)
  {
       a = _a; b = _b;
  }
 void print(){
       a.print(); b.print();
  }
}
```

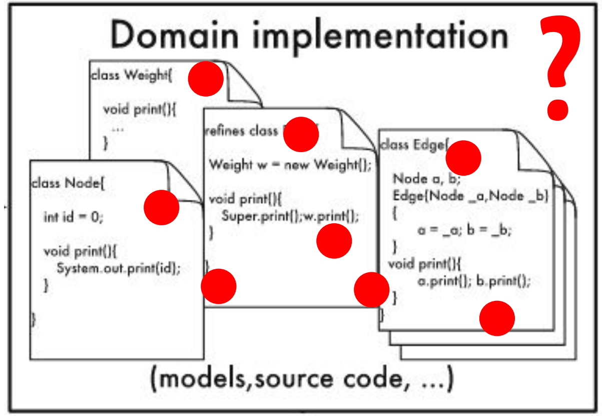
OO codebases use OO mechanisms to implement variability in a single codebase

- inheritance
- overloading of methods and constructors
- design patterns

Creation of **complex zones** in the system

⇒ **understanding them is crucial** to comprehend the codebase variability

Undocumented OO variability implementations



```
class Weight{

  void print(){
    ...
  }
```

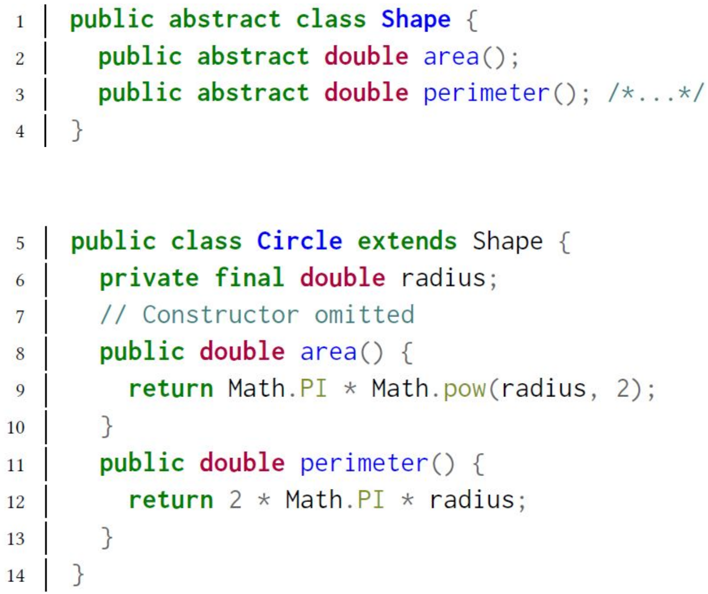
```
class Node{

  int id = 0;

  void print(){
    System.out.print(id);
  }

}
```

```
refines class

Weight w = new Weight();

void print(){
  Super.print();w.print();
}

}
```

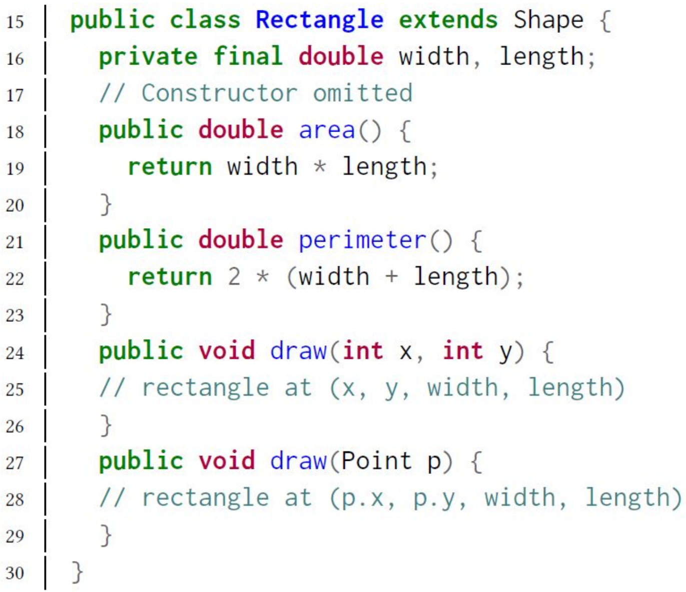
```
class Edge{

  Node a, b;
  Edge(Node _a,Node _b)
  {
      a = _a; b = _b;
  }
 void print(){
      a.print(); b.print();
 }
}
```

Problem: How to identify and comprehend object-oriented variability implementations?

Variation points and variants



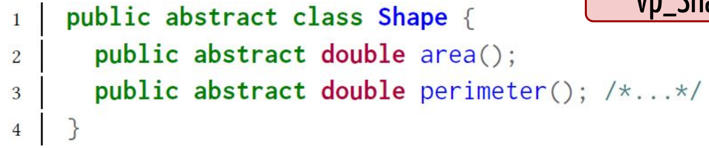
```java
public abstract class Shape {
  public abstract double area();
  public abstract double perimeter(); /*...*/
}


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;
  }
}
```



```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) {
  // rectangle at (p.x, p.y, width, length)
  }
}
```

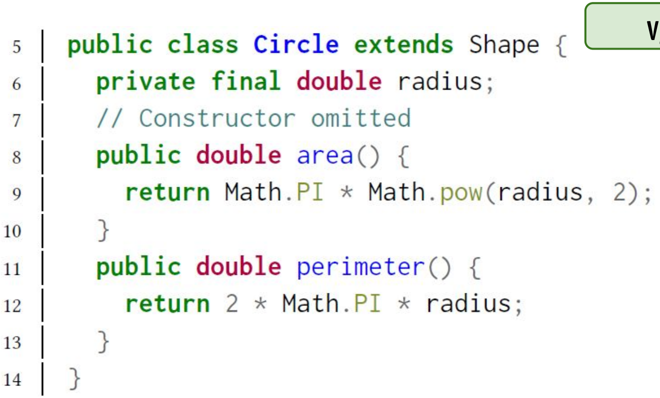
Variation points and variants

vp_Shape



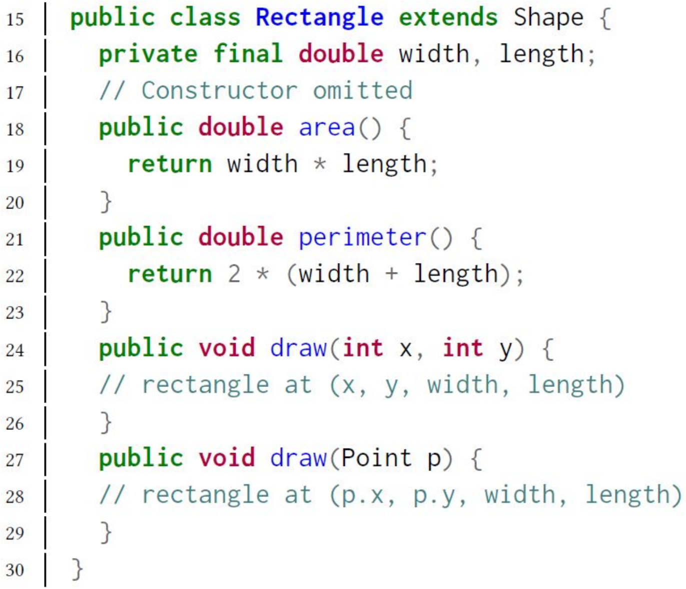
```java
public abstract class Shape {
  public abstract double area();
  public abstract double perimeter(); /*...*/
}
```

v_Circle



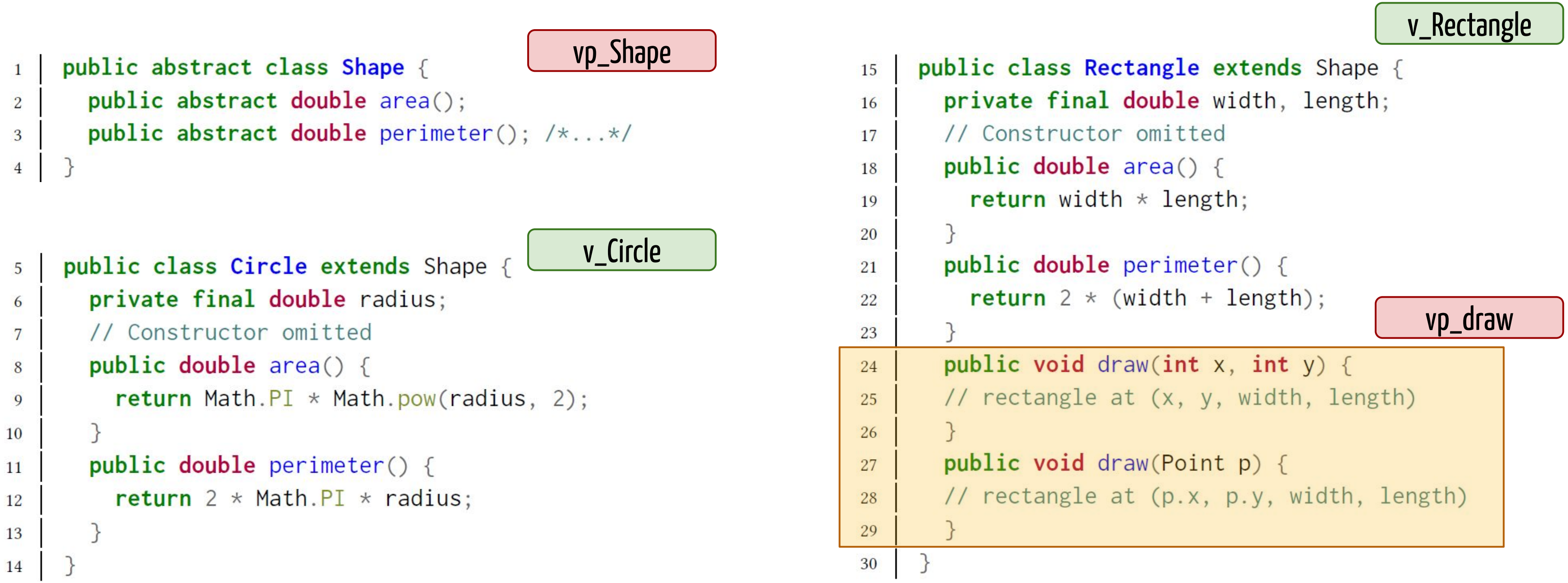
```java
public class Circle extends Shape {
  private final double radius;
  // Constructor omitted
  public double area() {
    return Math.PI * Math.pow(radius, 2);
  }
  public double perimeter() {
    return 2 * Math.PI * radius;
  }
}
```

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);
  }
  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)
  }
}
```

Variation points and variants



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;
  }
}
```

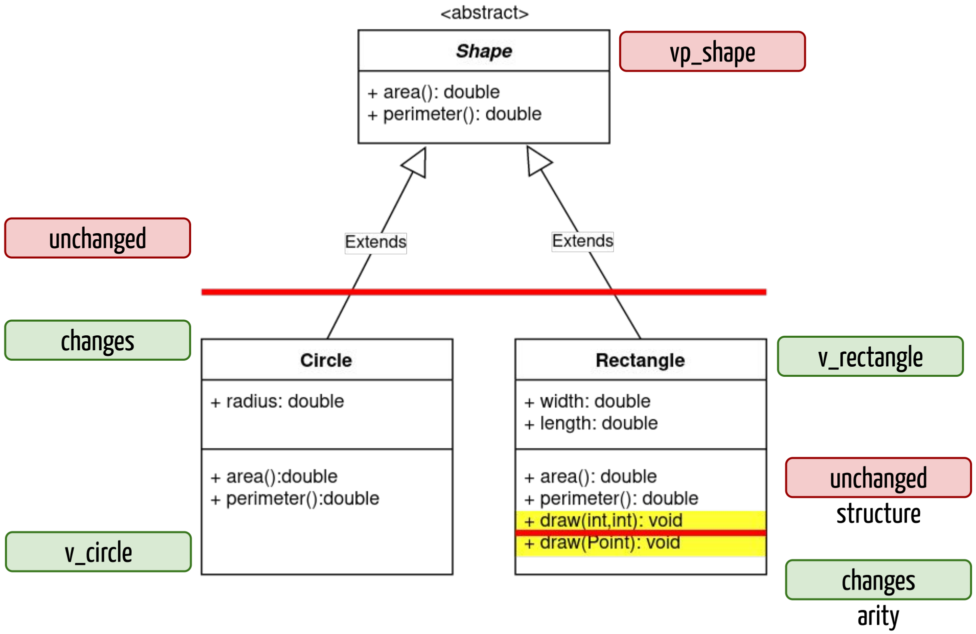
v_Rectangle

```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);
  }
  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)
  }
}
```

vp_draw

Identifying OO variability implementations with symmetries

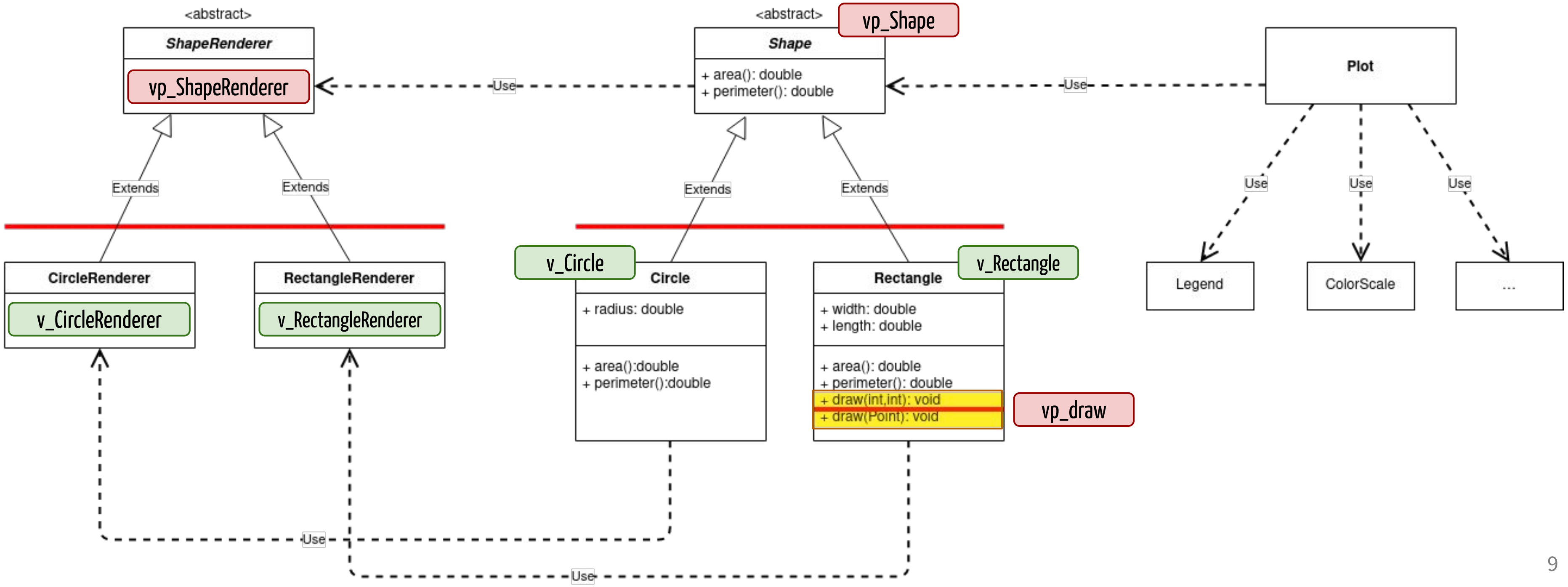
- **Symmetries** exist in each OO mechanism (Coplien and Zhao's work)
- Symmetries present in **mechanisms implementing variability**



High density of symmetries
⇒ high density of variability

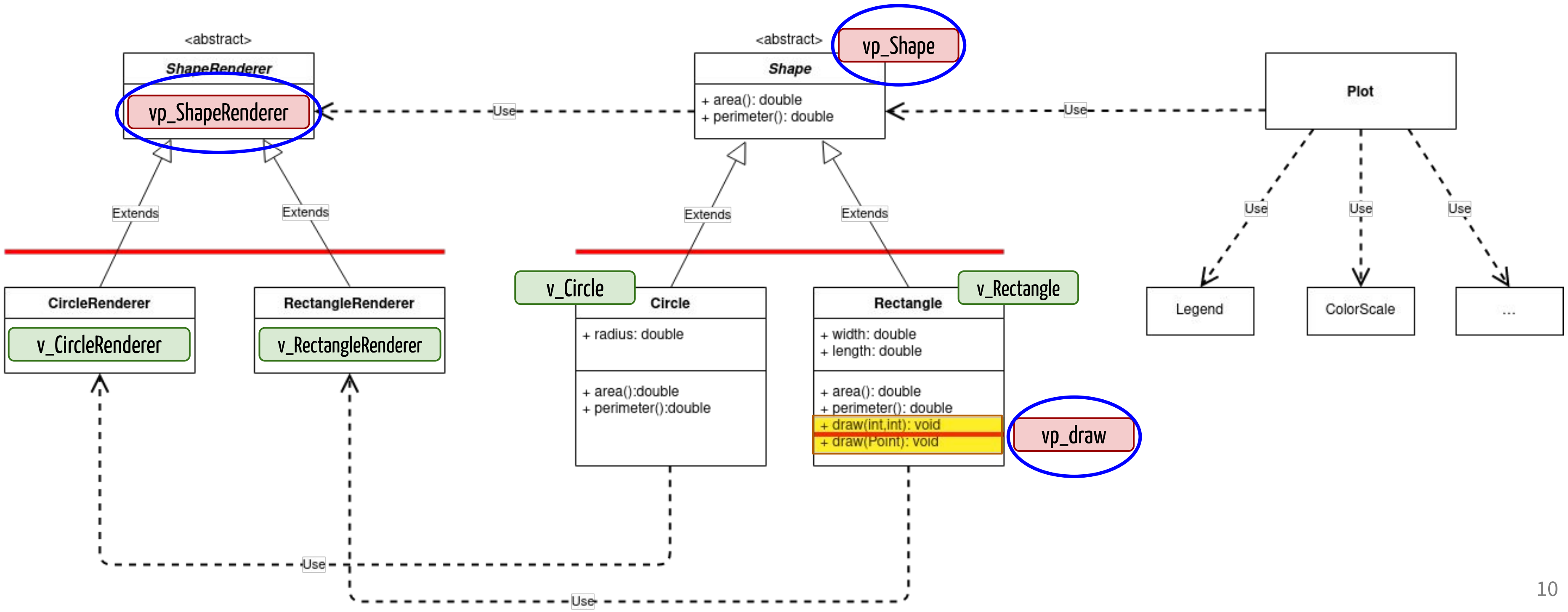
Xhevahire Tërnava, Johann Mortara, and Philippe Collet. "Identifying and visualizing variability in object-oriented variability-rich systems". In: the 23rd International Systems and Software Product Line Conference. Paris, France: ACM Press, Sept. 2019, pp. 231–243.

Density of symmetries



Density of symmetries

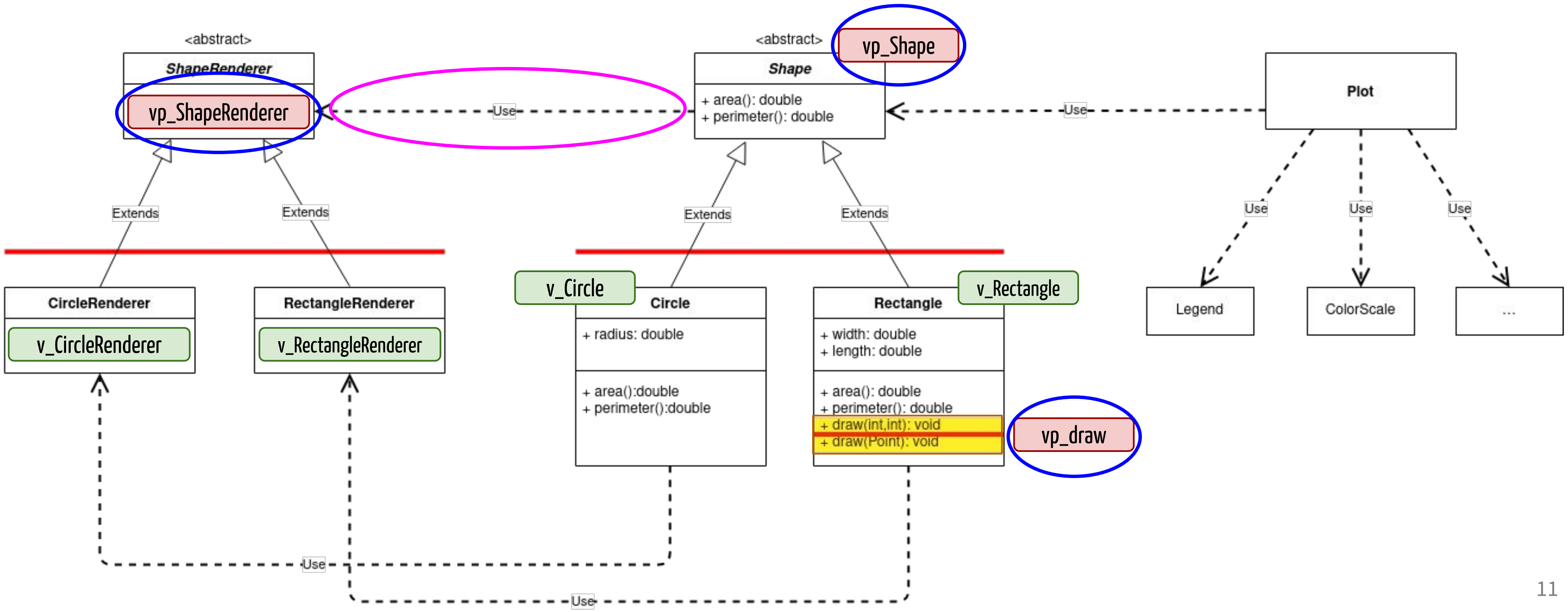
vp (class or method level) with important number of variants



Density of symmetries

vp (class or method level) with important number of variants

vp-s using each other

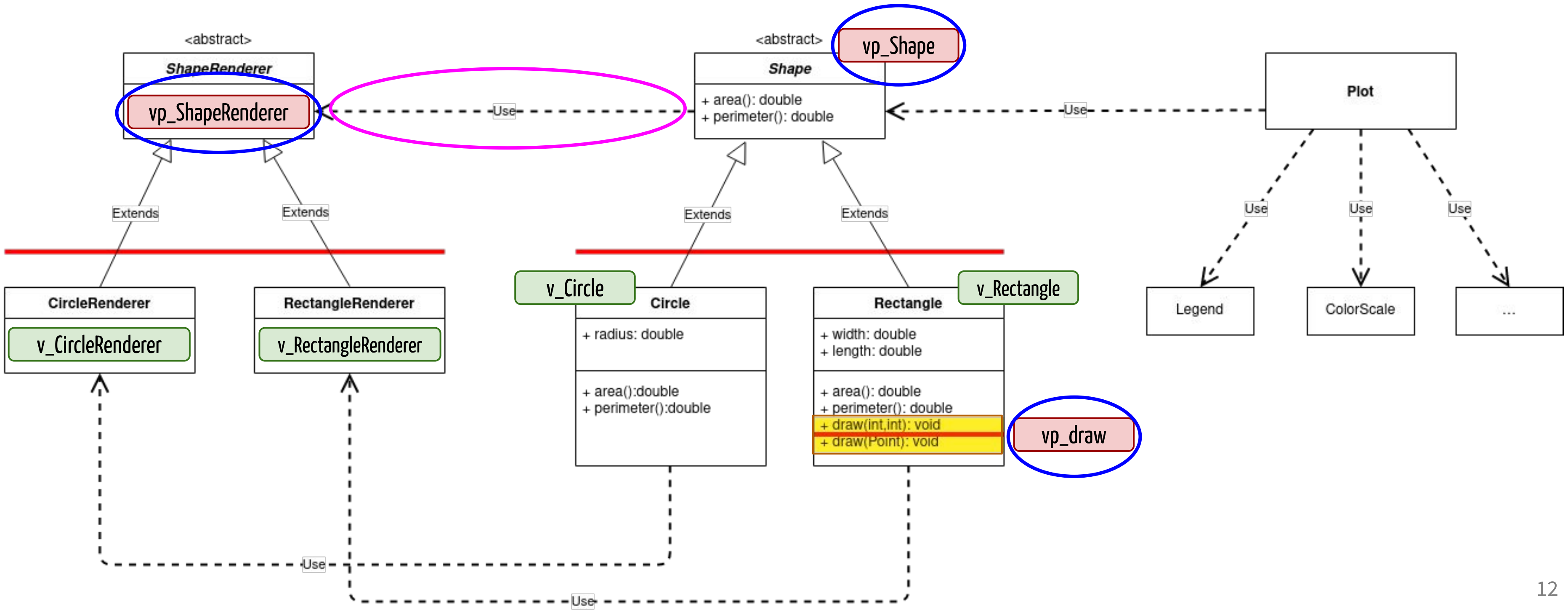


Density of symmetries

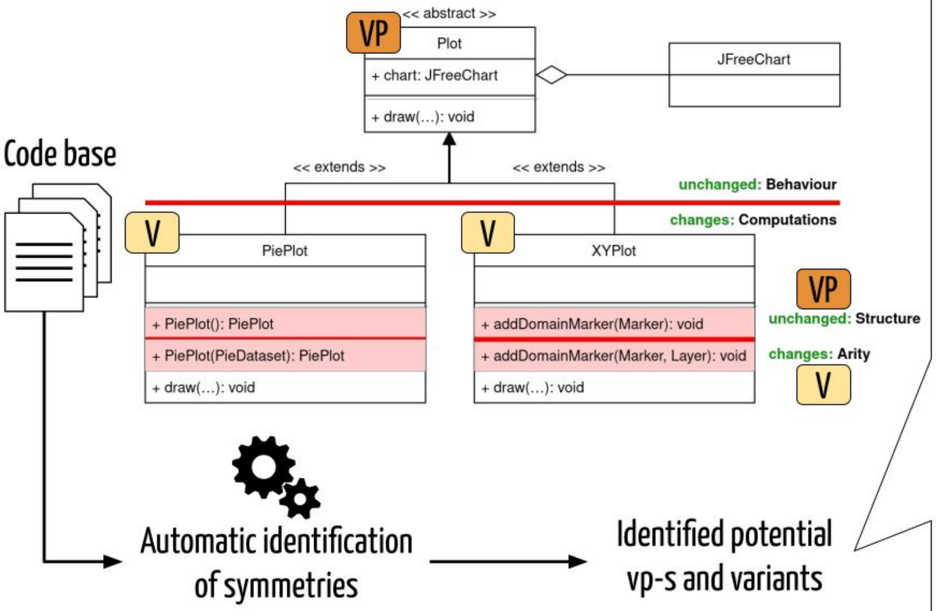
vp (class or method level) with important number of variants

vp-s using each other

HOTSPOTS



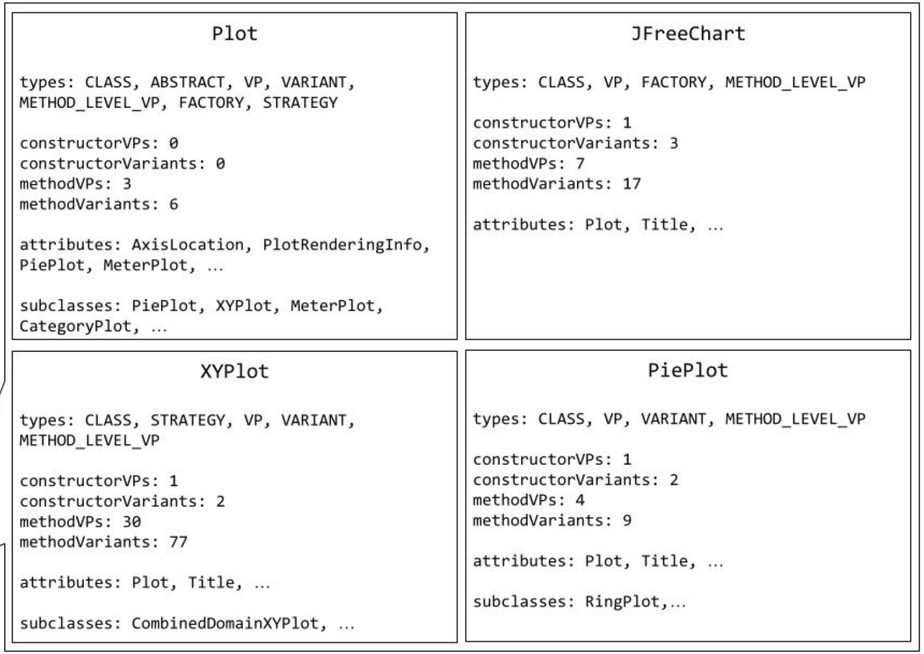
Automatic identification of variability implementations in an OO codebase



Automatic identification of symmetries

Identified potential vp-s and variants

metrics / properties



```
Plot

types: CLASS, ABSTRACT, VP, VARIANT,
METHOD_LEVEL_VP, FACTORY, STRATEGY

constructorVPs: 0
constructorVariants: 0
methodVPs: 3
methodVariants: 6

attributes: AxisLocation, PlotRenderingInfo,
PiePlot, MeterPlot, ...

subclasses: PiePlot, XYPlot, MeterPlot,
CategoryPlot, ...
```

```
JFreeChart

types: CLASS, VP, FACTORY, METHOD_LEVEL_VP

constructorVPs: 1
constructorVariants: 3
methodVPs: 7
methodVariants: 17

attributes: Plot, Title, ...
```

```
XYPlot

types: CLASS, STRATEGY, VP, VARIANT,
METHOD_LEVEL_VP

constructorVPs: 1
constructorVariants: 2
methodVPs: 30
methodVariants: 77

attributes: Plot, Title, ...

subclasses: CombinedDomainXYPlot, ...
```

```
PiePlot

types: CLASS, VP, VARIANT, METHOD_LEVEL_VP

constructorVPs: 1
constructorVariants: 2
methodVPs: 4
methodVariants: 9

attributes: Plot, Title, ...

subclasses: RingPlot,...
```

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

Finding an appropriate visualization

Goal: help the comprehension of variability intense zones in a large codebase

Finding an appropriate visualization

Goal: help the comprehension of variability intense zones in a large codebase

Requirement 1

the visualization must **display metrics on classes** and **relationships between them**, exhibiting the density of variability implementations

Finding an appropriate visualization

Goal: help the comprehension of variability intense zones in a large codebase

Requirement 1

the visualization must **display metrics on classes** and **relationships between them**, exhibiting the density of variability implementations

Requirement 2

the visualization must **scale on large systems**

Finding an appropriate visualization

Goal: help the comprehension of variability intense zones in a large codebase

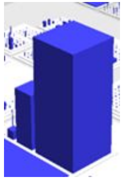
Requirement 1

the visualization must **display metrics on classes** and **relationships between them**, exhibiting the density of variability implementations

Requirement 2

the visualization must **scale on large systems**

The city metaphor



Finding an appropriate visualization

Goal: help the comprehension of variability intense zones in a large codebase

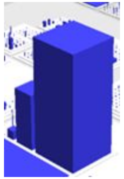
Requirement 1

the visualization must **display metrics on classes** and **relationships between them**, **exhibiting the density of variability implementations**

Requirement 2

the visualization must **scale on large systems**

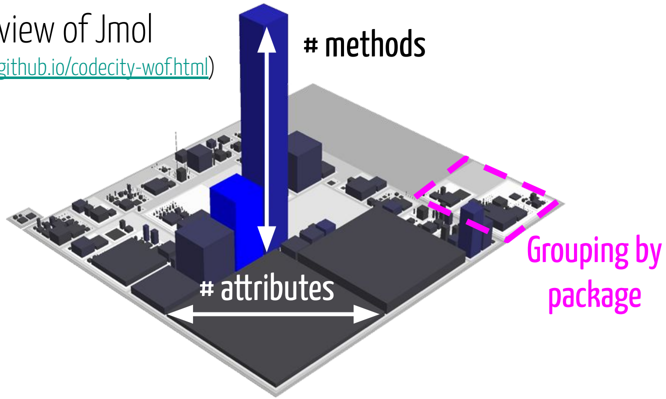
The city metaphor

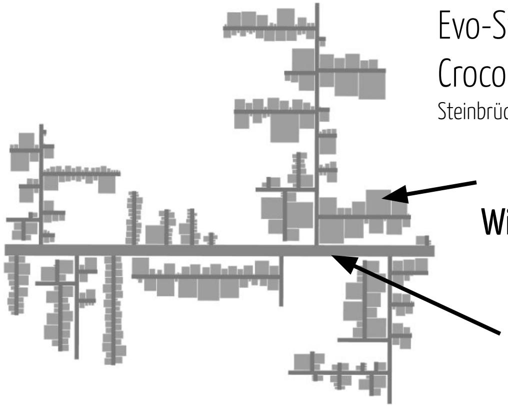


adapted for variability implementations

From CodeCity and Evo-Streets to VariCity

CodeCity view of Jmol
(https://wettel.github.io/codecity-wof.html)

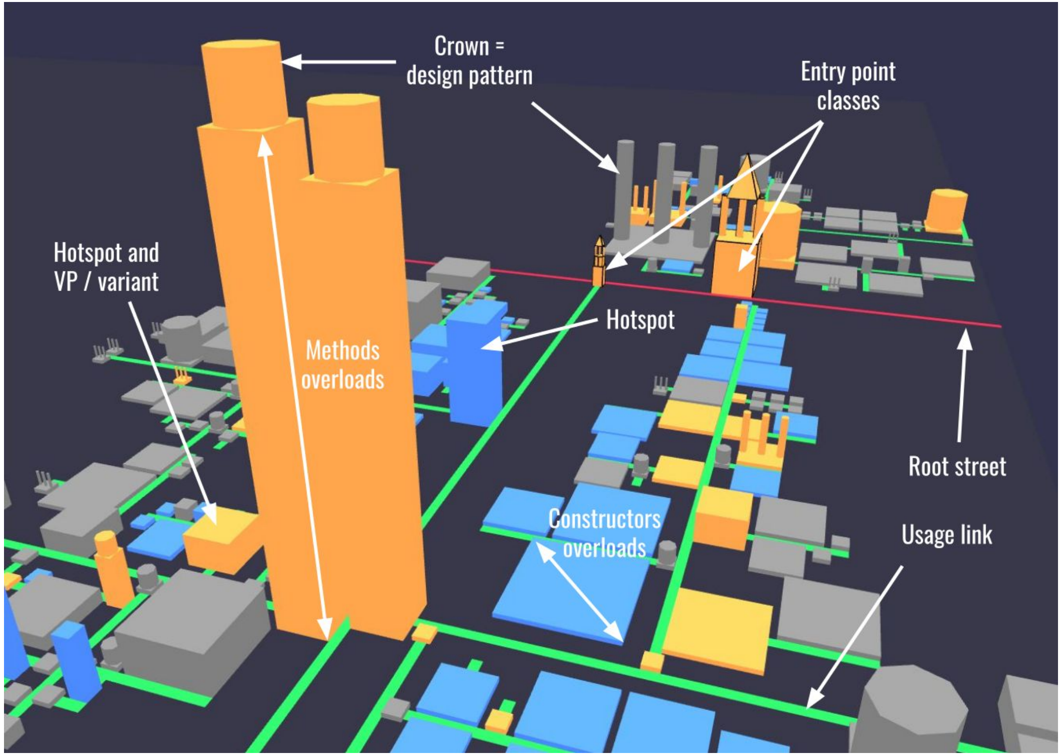




Evo-Streets view of CrocoCosmos
Steinbrückner and Lewerentz, 2010

Building = class
Width = class metric

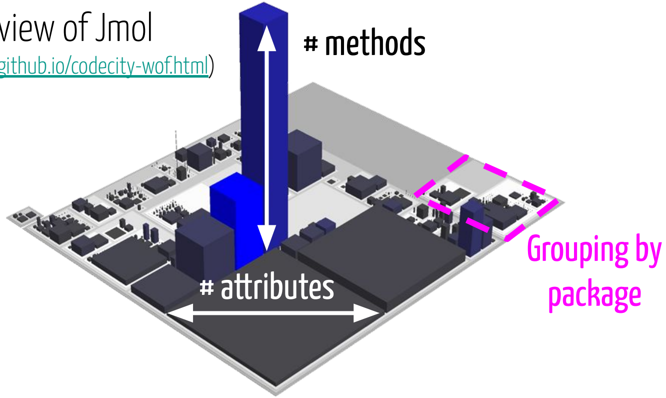
Street = package

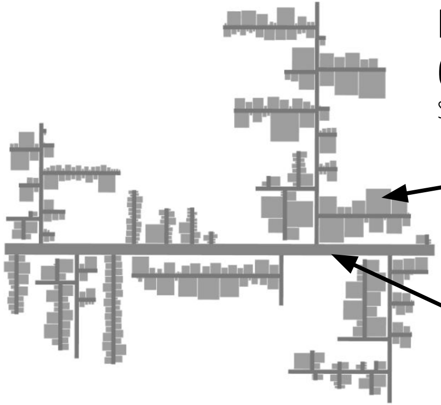


VariCity view of JFreeChart

From CodeCity and Evo-Streets to VariCity

CodeCity view of Jmol
(https://wettel.github.io/codecity-wof.html)

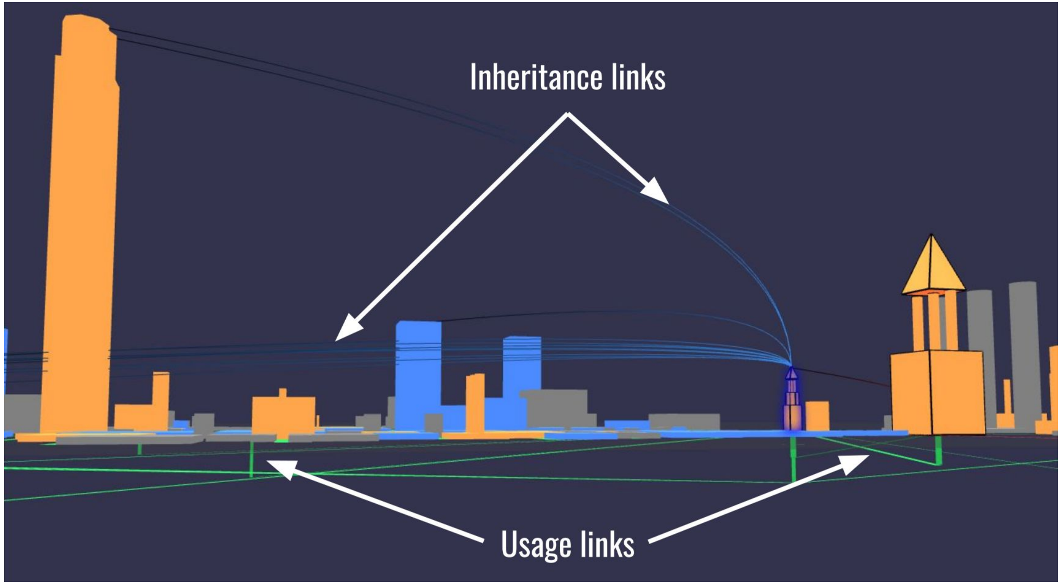




Evo-Streets view of CrocoCosmos
Steinbrückner and Lewerentz, 2010

Building = class
Width = class metric

Street = package



Additional links for inheritance

Interaction capabilities

Zooming, spanning

Hovering buildings to display additional links

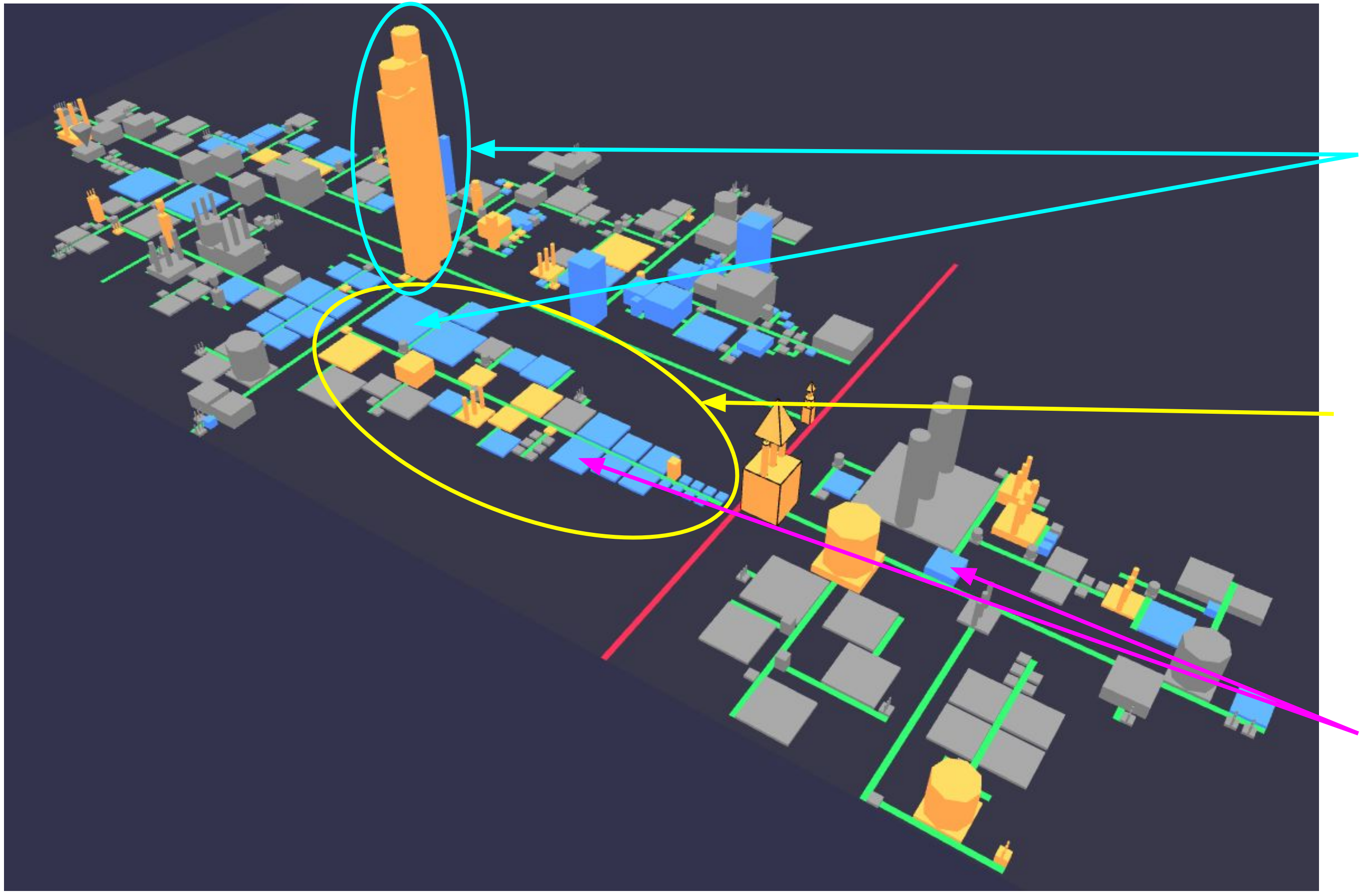
Increasing / decreasing the usage level

Orientation of the usage relationships

Adapting the entrypoints

Visualization of JFreeChart

Exhibiting density of variability implementations



Tall and / or large buildings = density at method level

Large neighbourhoods = density by usage relationships

Hotspots maximize both density types

Evaluation

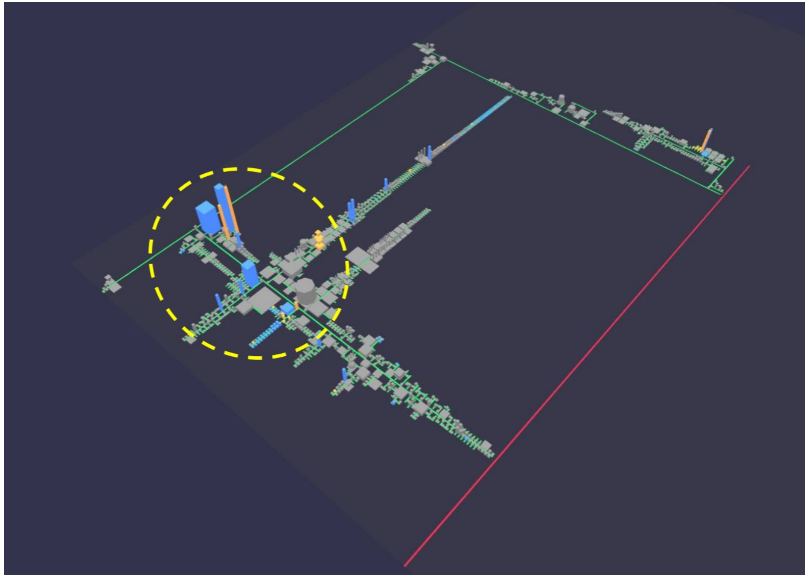
Variability implementations are **complex zones** in the code that **newcomers onboarding on a project seek to understand** [1]

Two types of users are part of onboarding scenarios:

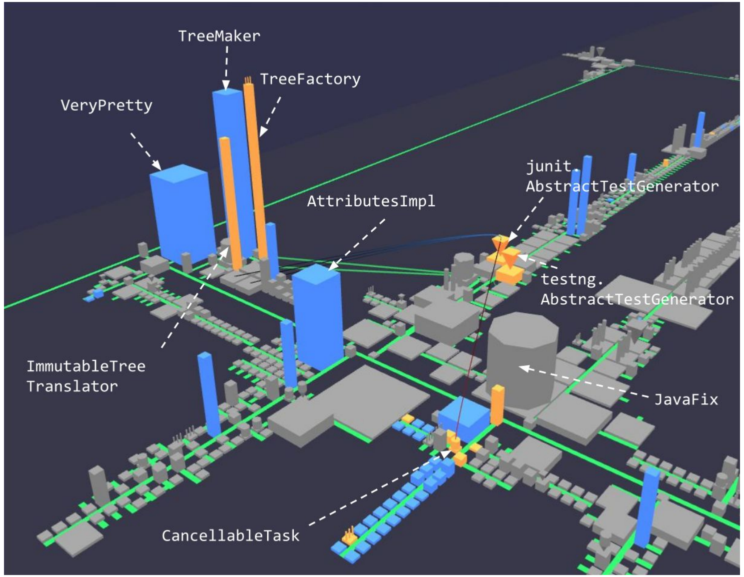
- a **newcomer** is **onboarded on a project** and has to grasp its important parts
- an **expert** has a **deep knowledge of the codebase**, and helps the newcomer to discover it

[1] R. Yates, N. Power, and J. Buckley, "Characterizing the transfer of program comprehension in onboarding: an information-push perspective," Empirical Software Engineering, vol. 25, no. 1, pp. 940–995, 2020.

Scenario 1: An expert wants to **facilitate the exploration** of the codebase by giving a pre-configured visualization to the newcomer.

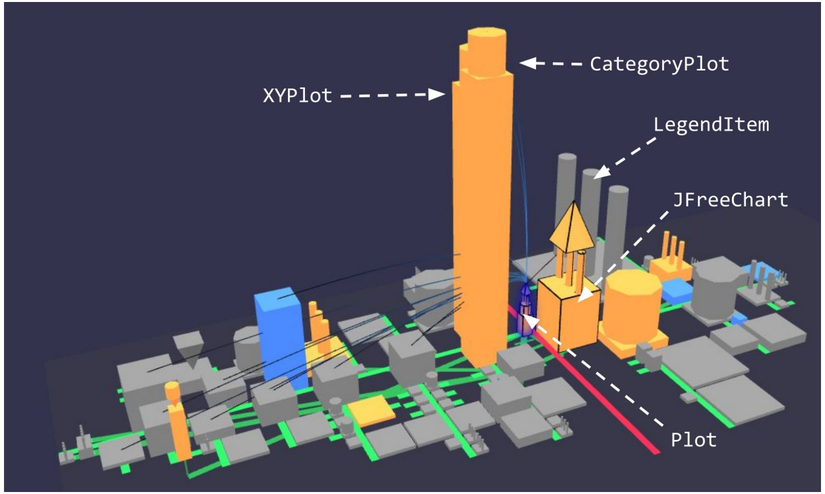


Preconfigured view of NetBeans, neighbourhood of tall and blue buildings detaches

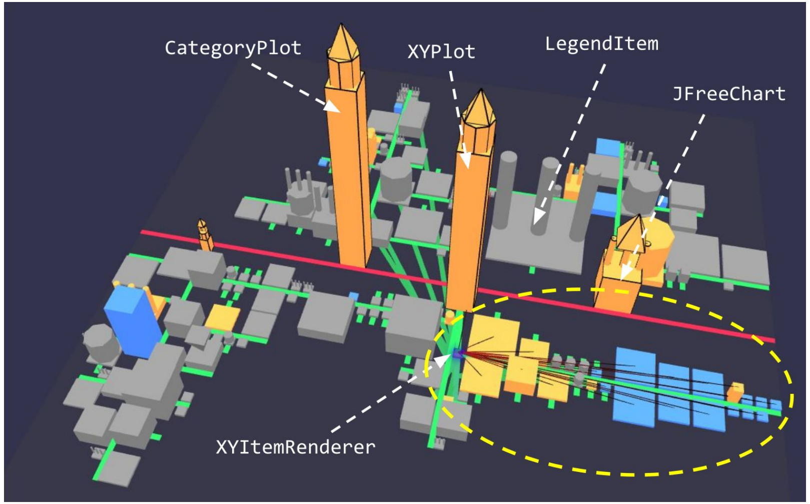


Zooming and spanning allow to explore at finer-grain the city

Scenario 2: The expert wants the newcomer to **comprehend a subpart** of the codebase for the newcomer to be able to reuse it.



Preconfigured view of JFreeChart with Plot as entrypoint. Displaying links of Plot reveals that XYPlot and CategoryPlot are subclasses.



Adding XYPlot and CategoryPlot as entrypoints allows to display other buildings forming a variability intense neighbourhood.

Future work

Real experts evaluation

Integration in an IDE

Add other metrics of code quality

⇒ gain new insights on how to better facilitate the identification of variability implementations

Visualization of Object-Oriented Variability Implementations as Cities

Johann Mortara — Philippe Collet — Anne-Marie Dery-Pinna

OO variability implementations are **complex to identify and comprehend**

VariCity provides a **visualization** relying on the **city metaphor** of OO variability implementations

Visualization **exhibits zones of high density of variability**, in classes and between classes

Get the paper:

https://hal.archives-ouvertes.fr/hal-03312487

VariCity website:

https://deathstar3.github.io/varicity-demo/

Best artifact award of the VISSOFT / ICSME 2021 conferences!

Reproduction package:

https://doi.org/10.5281/zenodo.5034199

Obtained reproducibility badges

Open Research Objects

Research Objects Reviewed