Customizable Visualization of Quality Metrics for Object-Oriented Variability Implementations

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

Université Côte d'Azur, CNRS, I3S, France

SPLC '22 – Graz, Austria

September 15, 2022

Highly-variable Systems with a Single Code Base

2.000+ options generating variants for platforms, security levels… in 27M+ LoC

#ifdef & Object-orientation

NetBeans.IDE

Multiple configuration options for the editor, runners… in 5M+ LoC

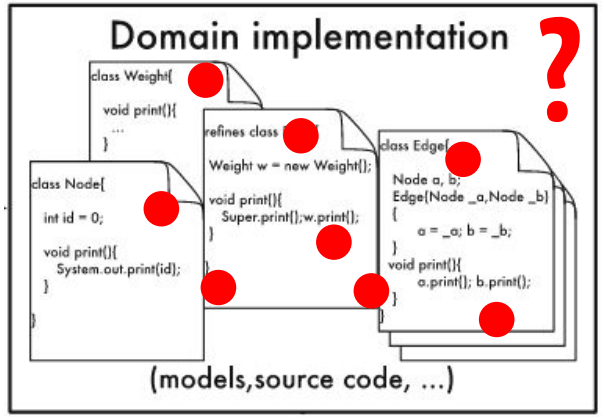
Object-orientation

and multiple implementation techniques…

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

- inheritance
- overloading of methods and constructors
- design patterns

Undocumented OO variability implementations



Creation of **complex zones** in the system

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

Variation points and variants

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

```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;
  }
}
```

```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

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

v_Circle

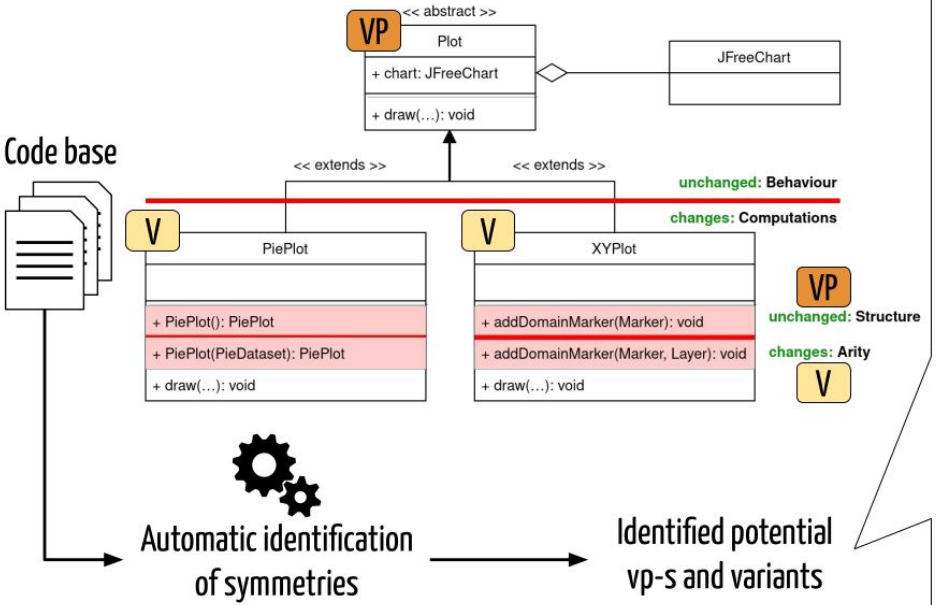
```
5  public class Circle extends Shape {
6    private final double radius;
7    // Constructor omitted
8    public double area() {
9      return Math.PI * Math.pow(radius, 2);
10   }
11   public double perimeter() {
12     return 2 * Math.PI * radius;
13   }
14 }
```

v_Rectangle

```
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 }
```

vp_draw

Automatic identification of variability implementations in an OO codebase



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

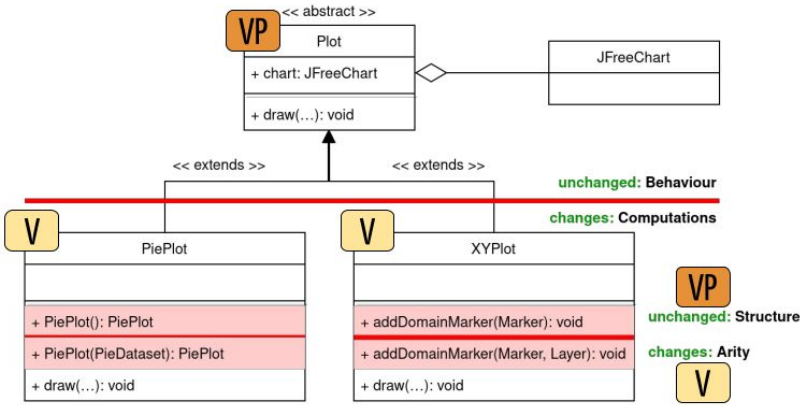
attributes: Plot, Title, …

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

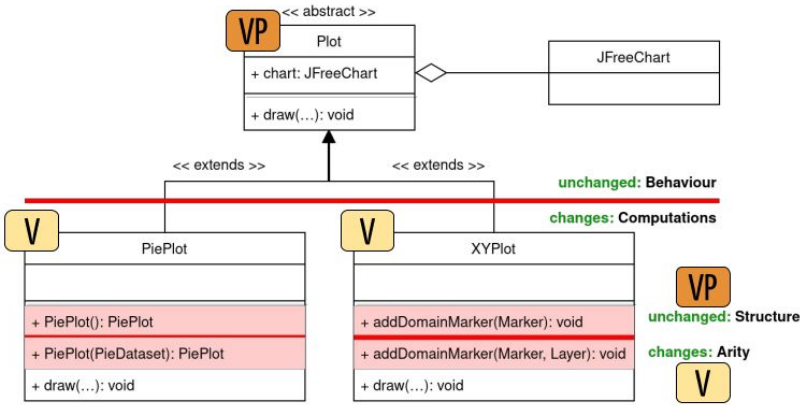
The reality of OO variability implementation

Variability implemented using mechanisms



The reality of OO variability implementation

Variability implemented using mechanisms

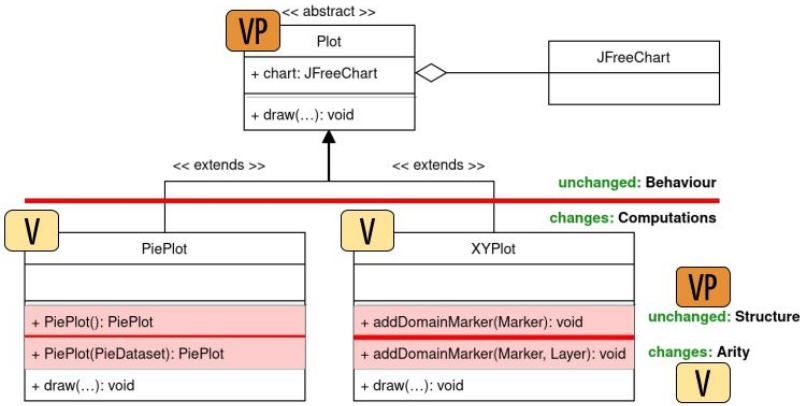


Variability implemented without using mechanisms

Plot
drawPiePlot(): void
drawXYPlot(): void

The reality of OO variability implementation

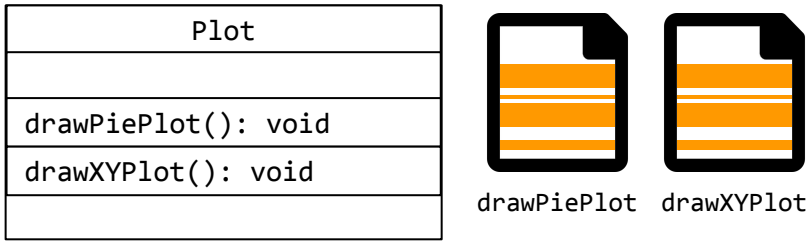
Variability implemented using mechanisms



Duplicated blocks: 3

Duplicated blocks: 2

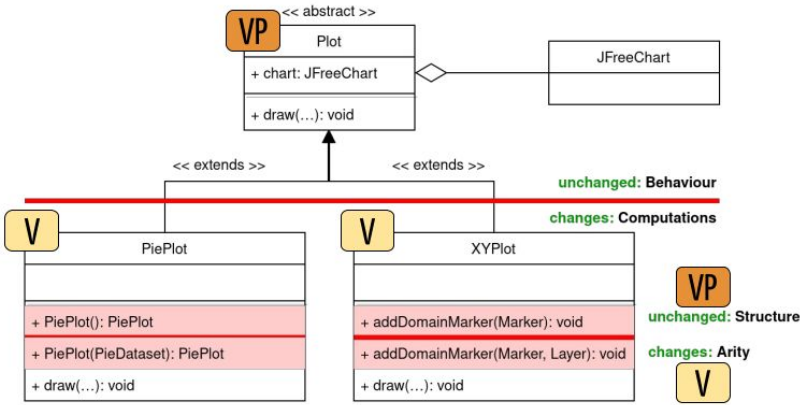
Variability implemented without using mechanisms



Duplicated blocks: 25

The reality of OO variability implementation

Variability implemented using mechanisms



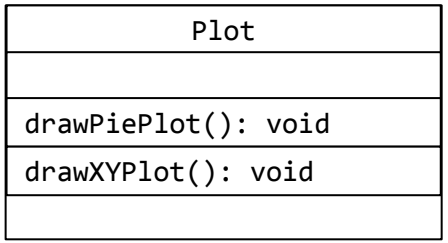
Duplicated blocks: 3

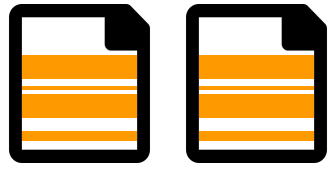
Code coverage: 80%

Duplicated blocks: 2

Code coverage: 75%

Variability implemented without using mechanisms



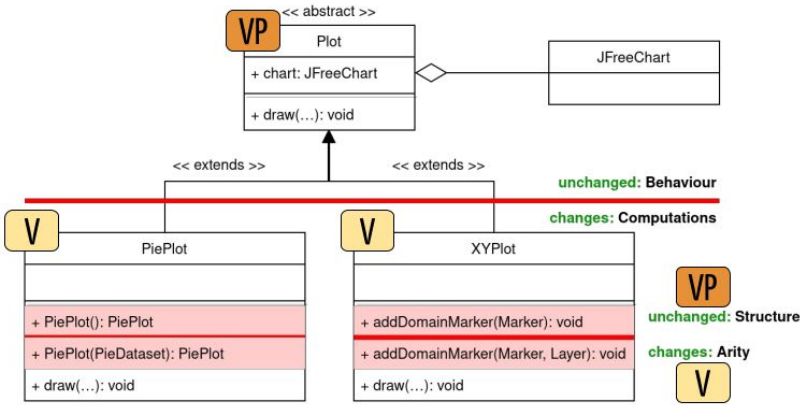


Duplicated blocks: 25

Code coverage: 55%

The reality of OO variability implementation

Variability implemented using mechanisms



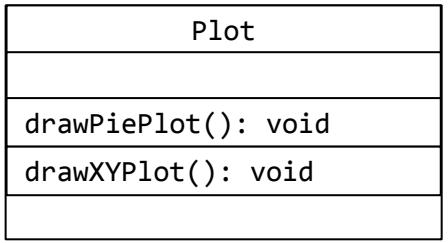
Duplicated blocks: 3

Code coverage: 80%

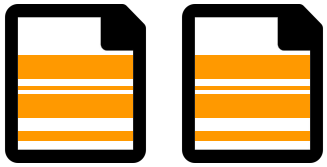
Duplicated blocks: 2

Code coverage: 75%

Variability implemented without using mechanisms



Duplications



drawPiePlot drawXYPlot

Duplicated blocks: 25

Code coverage: 55%

⇒ technical debt

Variability debt

"Technical debt **caused by defects and sub-optimal solutions in the implementation of variability** management in software systems. [...] Variability debt **leads to maintenance and evolution difficulties** to manage families of systems or highly configurable systems."

Potential identified causes:

- lack of knowledge of the implemented variability
- absence of traceability
- no known variability implementation mechanisms ⇒ artifact duplication + ↑ code complexity

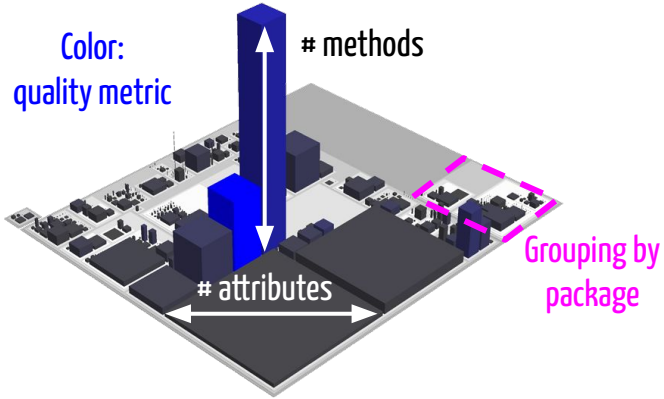
OO variability implementations are prone to variability debt

Need for identification and visualization

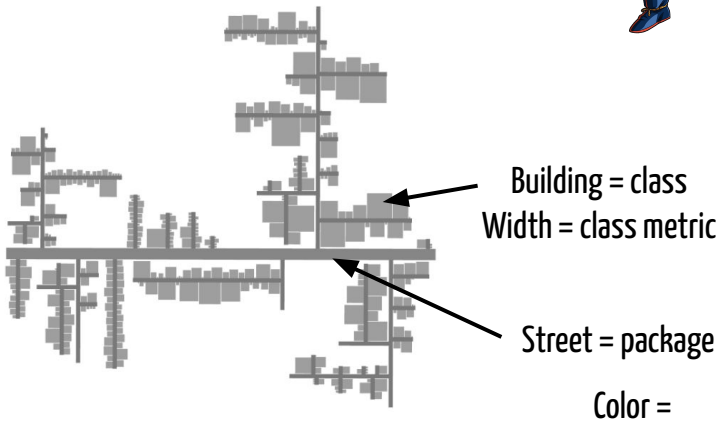
Daniele Wolfart, Wesley Klewerton Guez Assunção, and Jabier Martinez. 2021. Variability Debt: Characterization, Causes and Consequences. In XX Brazilian Symposium on Software Quality. 1–10.

Identifying and visualizing technical debt in OO variability implementations

OO code quality metrics visualized in the form of a city



CodeCity



Evo-Streets

Richard Wettel and Michele Lanza. 2007. Visualizing software systems as cities. 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis.
Frank Steinbrückner and Claus Lewerentz. 2013. Understanding software evolution with software cities. Information Visualization 12, 2 (April 2013), 200–216.

Identifying and visualizing technical debt in OO variability implementations

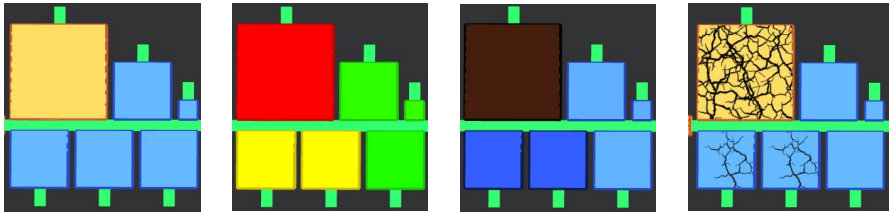
Johann Mortara, Philippe Collet, Anne-Marie Dery-Pinna. Visualization of Object-Oriented Variability Implementations as Cities. 9th IEEE Working Conference on Software Visualization (VISSOFT 2021), Sep 2021, Luxembourg (virtual), Luxembourg. ff10.1109/VISSOFT52517.2021.00017ff. ffhal03312487

FUUUUUUUUSION!

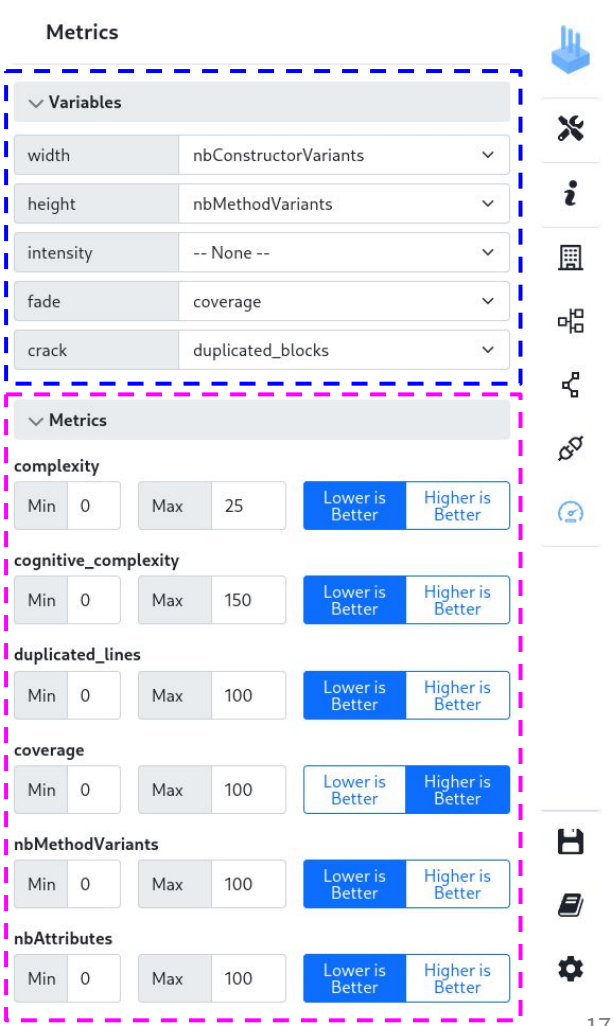
VariMetrics

Configurable visualization of OO quality metrics on the variability implementations

- Additional visual axes that can be combined to display multiple metrics simultaneously
- Ranges of values for metrics are also configurable

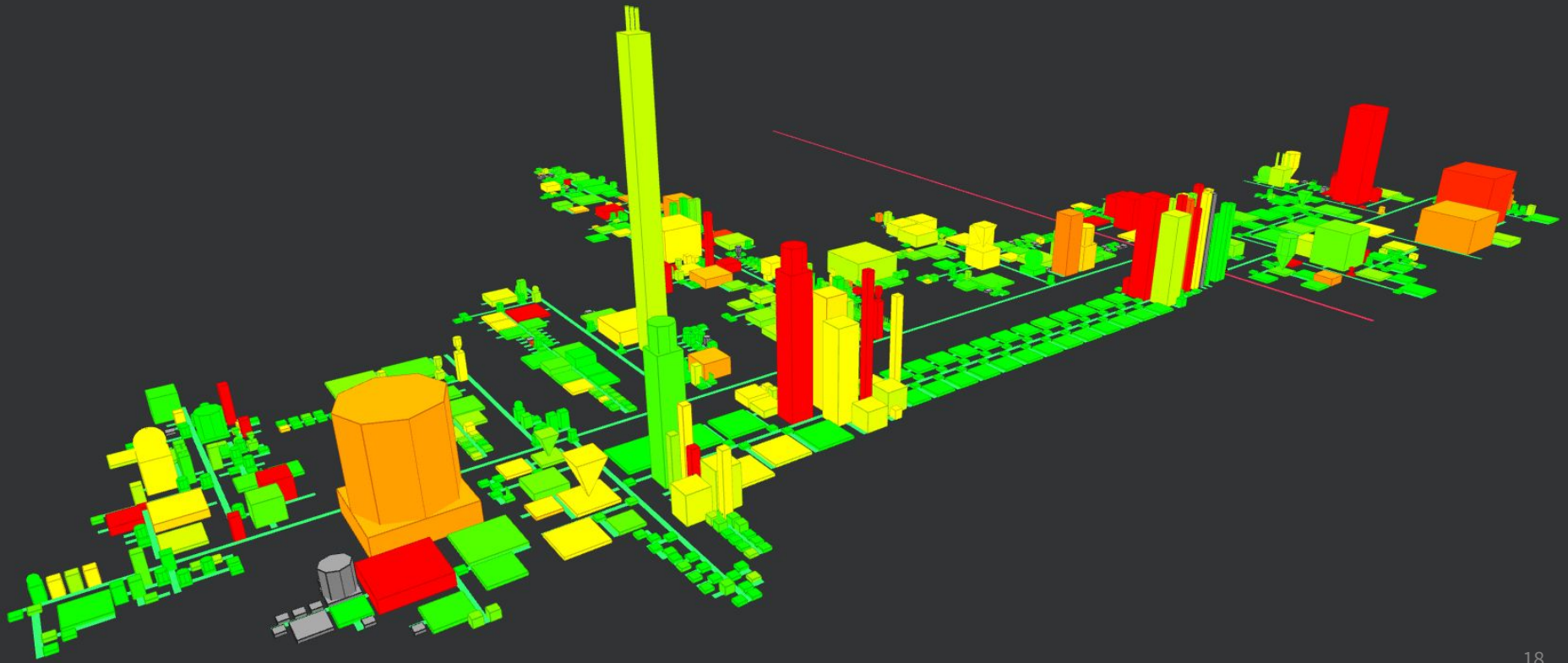


VariCity | Red-green | Saturation | Cracks



Example of VariMetrics view

Project: **GeoTools**
Red-to-green color scale: **cognitive complexity**



Quantitative evaluation

Does VariMetrics allow to visualize indebted zones of variability implementations?

Visual observation on **7 medium to large open source variable systems written in Java**

Determination of **relevant** VariMetrics visualizations:

- w.r.t. variability ⟶ some classes exhibit concentration of variability implementation mechanisms
- w.r.t. quality ⟶ some classes have quality issues

Quantitative evaluation

Does VariMetrics allow to visualize indebted zones of variability implementations?

Visual observation on **7 medium to large open source variable systems written in Java**

Determination of **relevant** VariMetrics visualizations:

- w.r.t. variability ⟶ some classes exhibit concentration of variability implementation mechanisms
- w.r.t. quality ⟶ some classes have quality issues **related to which metrics?**

Determining relevant quality metrics

Different types of variability debt

- System-level structure quality issues
- Code Duplication
- Lack of tests
- Out-of-date or incomplete documentation
- Architectural antipatterns
- Expensive tests
- Multi-version support
- Old technology in use
- Duplicate documentation
- Poor test of feature interactions

Daniele Wolfart, Wesley Klewerton Guez Assunção, and Jabier Martinez. 2021. Variability Debt: Characterization, Causes and Consequences. In XX Brazilian Symposium on Software Quality. 1–10.

Determining relevant quality metrics

Different types of variability debt applicable to OO codebases

- System-level structure quality issues
  in the implementation
- Code Duplication
- Lack of tests
- Out-of-date or incomplete documentation
- Architectural antipatterns
- Expensive tests
- Multi-version support
- Old technology in use
- Duplicate documentation
- Poor test of feature interactions

Daniele Wolfart, Wesley Klewerton Guez Assunção, and Jabier Martinez. 2021. Variability Debt: Characterization, Causes and Consequences. In XX Brazilian Symposium on Software Quality. 1–10.

Determining relevant quality metrics

Different types of variability debt applicable to OO codebases

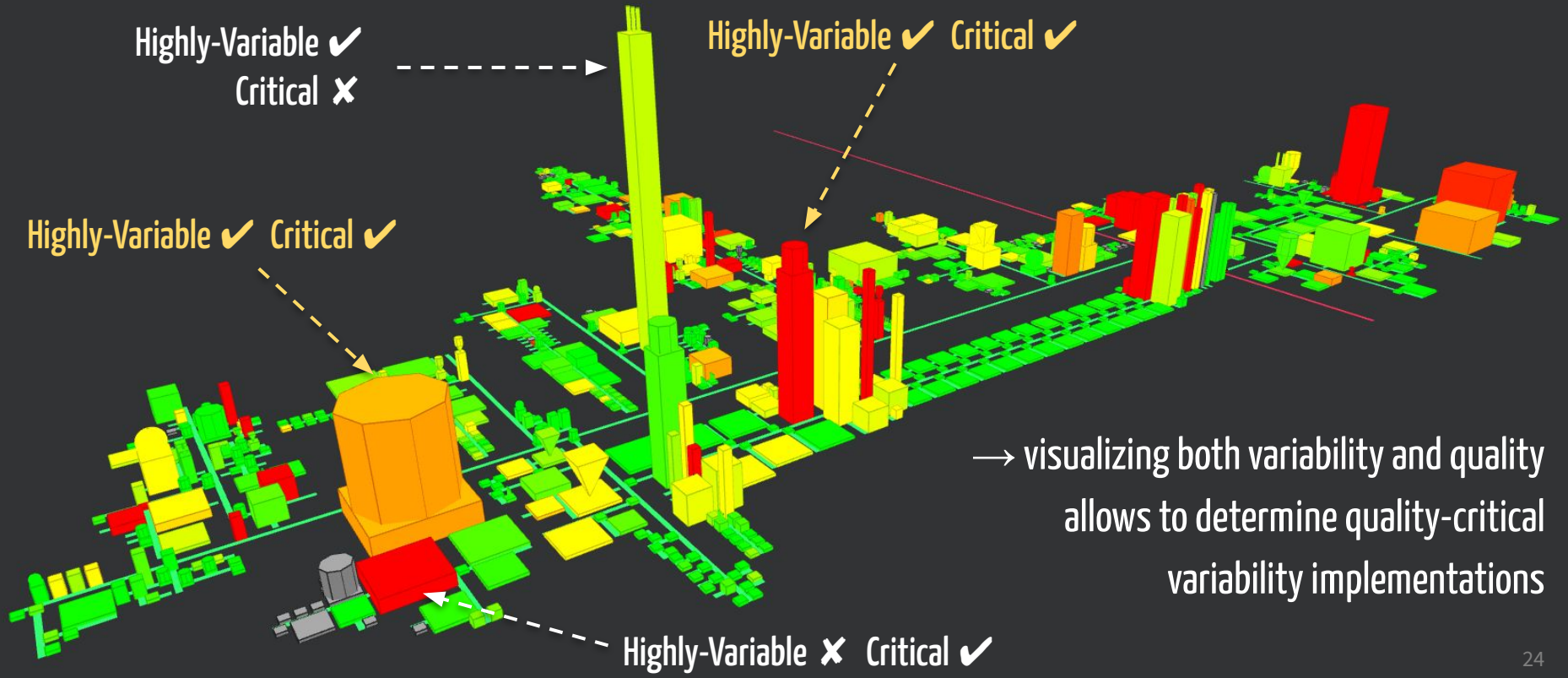
Chosen OO metrics

- System-level structure quality issues in the implementation → Cognitive complexity
- Code Duplication → Duplicated code blocks
- Lack of tests → Unit tests coverage

Daniele Wolfart, Wesley Klewerton Guez Assunção, and Jabier Martinez. 2021. Variability Debt: Characterization, Causes and Consequences. In XX Brazilian Symposium on Software Quality. 1–10.

No variability / quality correlation

Project: **GeoTools**
Red-to-green color scale: **cognitive complexity**



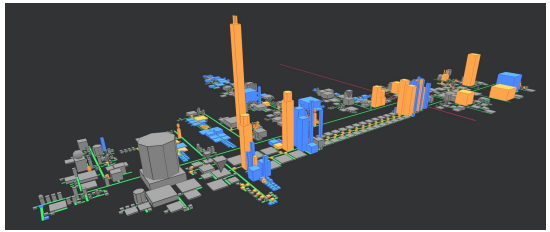
→ visualizing both variability and quality allows to determine quality-critical variability implementations

Quantifying the noticeable classes on VariMetrics

Protocol:

Quantifying the noticeable classes on VariMetrics

Protocol:

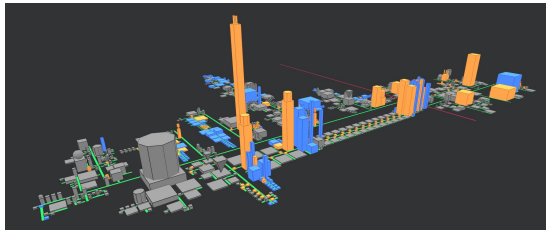


VariCity view

Visual identification of noticeable classes w.r.t. **variability**

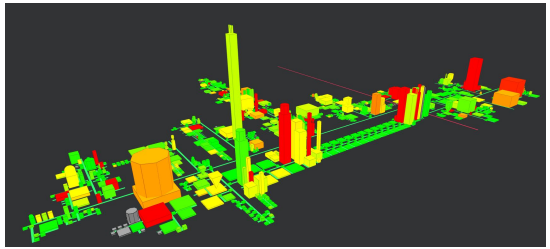
Quantifying the noticeable classes on VariMetrics

Protocol:



VariCity view

Visual identification of noticeable classes w.r.t. **variability**

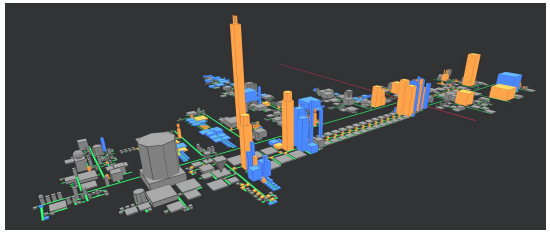


VariMetrics view

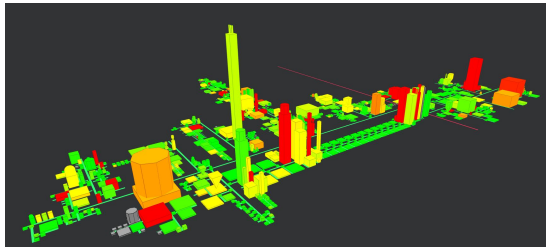
Visual identification of noticeable classes w.r.t. **criticality**

Quantifying the noticeable classes on VariMetrics

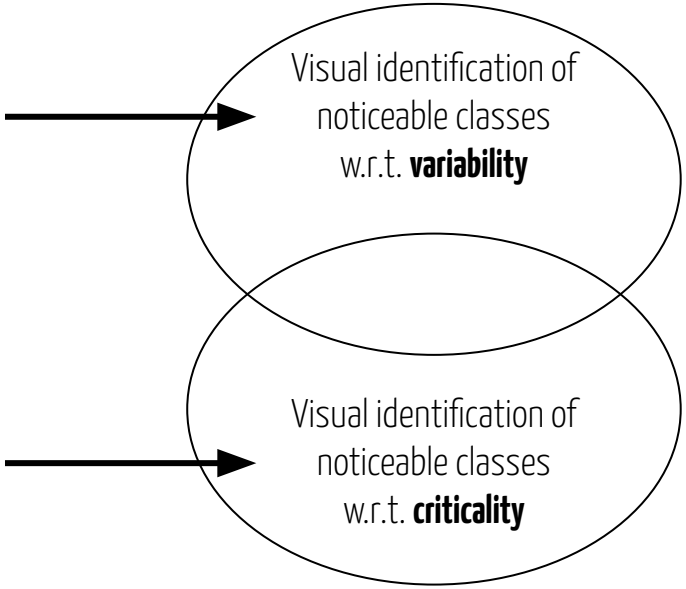
Protocol:



VariCity view

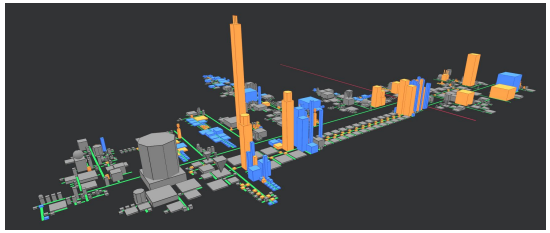


VariMetrics view

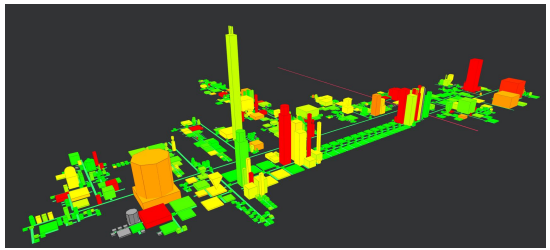


Quantifying the noticeable classes on VariMetrics

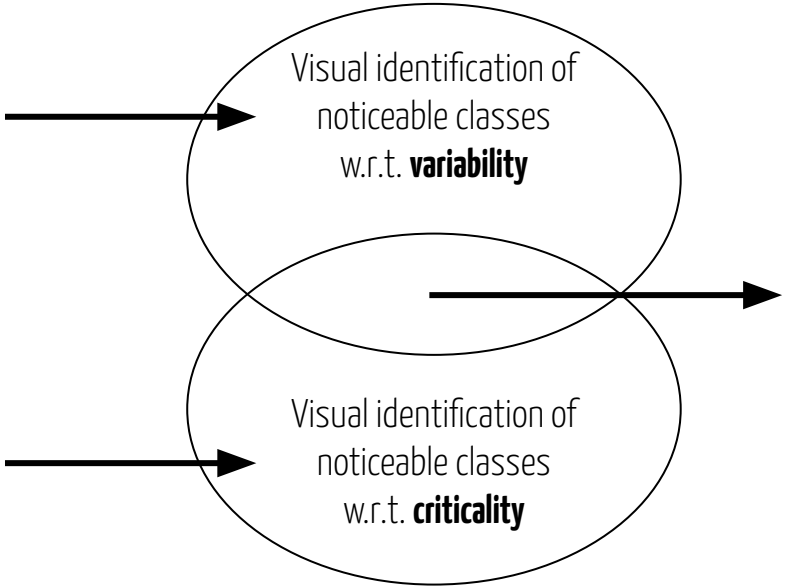
Protocol:



VariCity view



VariMetrics view



Visible classes w.r.t. **variability and criticality**

Quantitative evaluation

Less relevant classes with VariMetrics than with VariCity

Results depend mainly on:

- **codebase size**
  ↑ codebase size ⇒ ↑identified variability intense zones
- **global quality**
  ↓ quality ⇒ ↑noticeable classes

⟶ explains mildly encouraging results on JKube

System	Visible classes w.r.t.			% reduction VariCity ⟶ VariMetrics
	variability	criticality	both	
Azureus	74	32	12	84 %
GeoTools	104	27	18	83 %
JDK	84	17	13	85 %
JFreeChart	35	31	10	71 %
JKube	28	115	14	50 %
OpenAPI Generator	77	51	21	72 %
Spring framework	57	13	6	91 %

Qualitative evaluation on JFreeChart

Are the shown indebted zones of variability implementations relevant?

Qualitative evaluation on JFreeChart

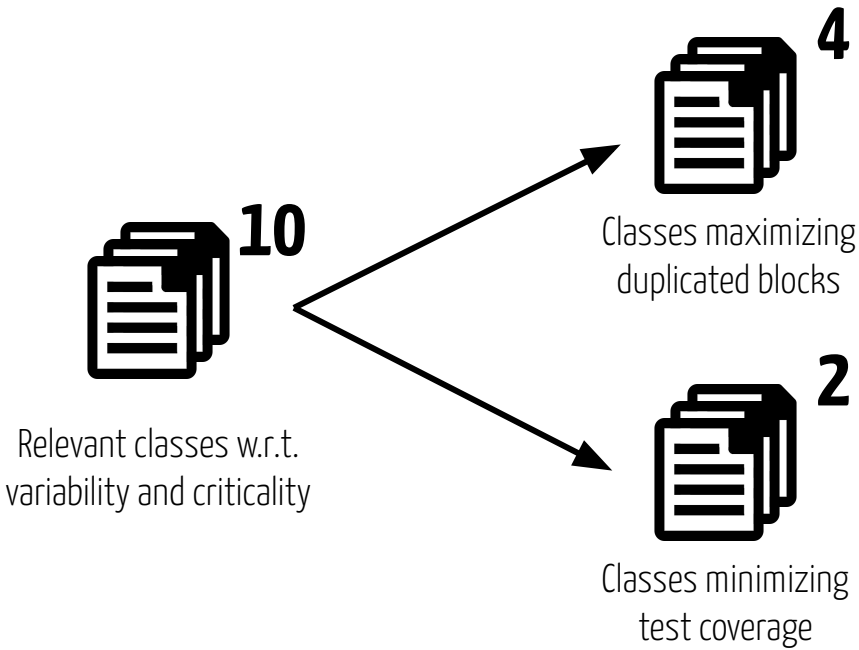
Are the shown indebted zones of variability implementations relevant?

10

Relevant classes w.r.t. variability and criticality

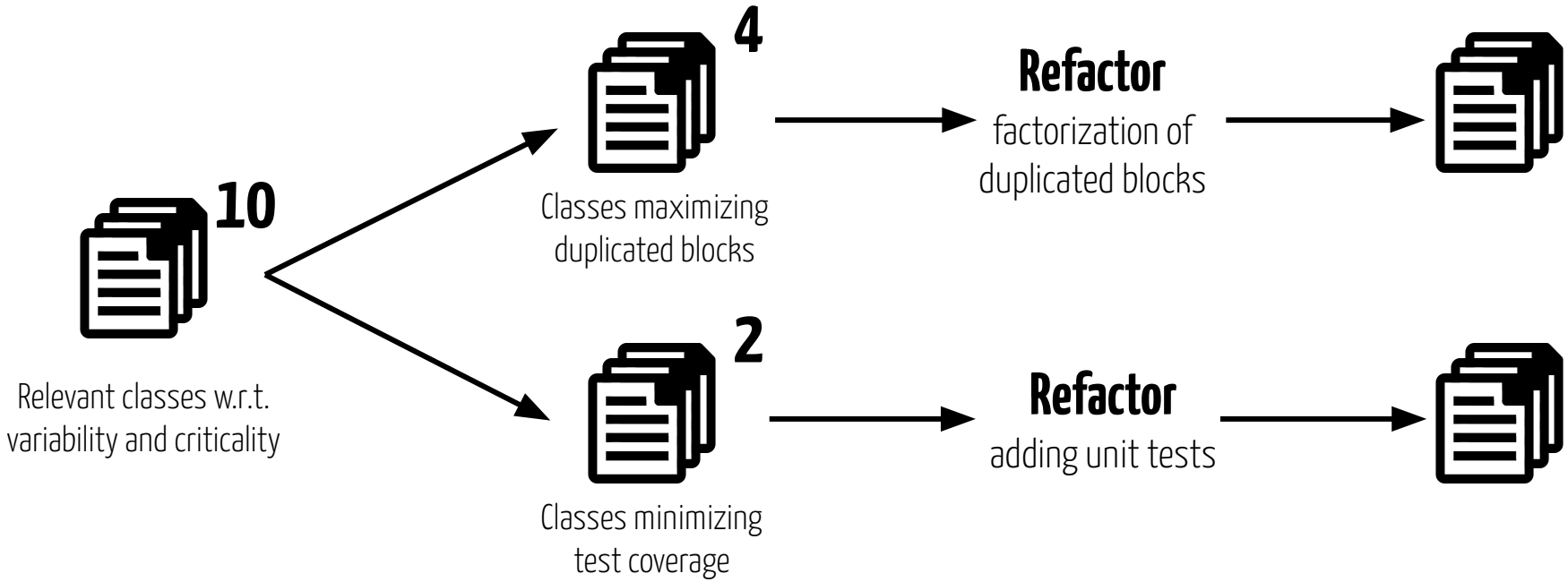
Qualitative evaluation on JFreeChart

Are the shown indebted zones of variability implementations relevant?

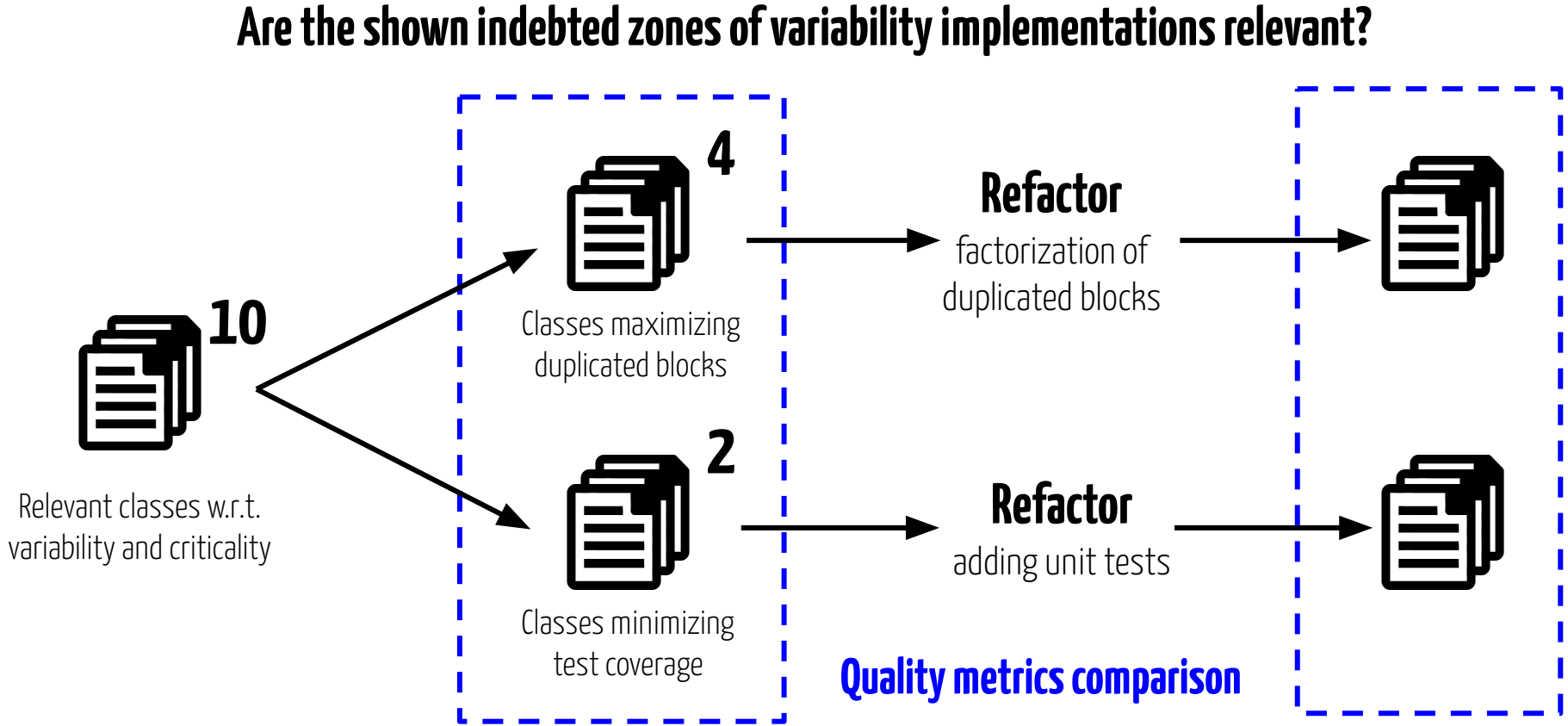


Qualitative evaluation on JFreeChart

Are the shown indebted zones of variability implementations relevant?

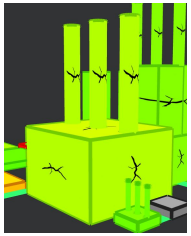


Qualitative evaluation on JFreeChart



Findings

org.jfree.chart.axis.DateAxis
as visualized on VariMetrics



Duplications can be pure technical debt in classes concentrating variability implementations, but can also be **improperly managed variability implementations**

```
if (isVerticalTickLabels()) {
    anchor = TextAnchor.CENTER_RIGHT;
    rotationAnchor = TextAnchor.CENTER_RIGHT;
    if (edge == RectangleEdge.TOP) {
        angle = Math.PI / 2.0;
    }
    else {
        angle = -Math.PI / 2.0;
    }
}
else {
    if (edge == RectangleEdge.TOP) {
        anchor = TextAnchor.BOTTOM_CENTER;
        rotationAnchor = TextAnchor.BOTTOM_CENTER;
    }
    else {
        anchor = TextAnchor.TOP_CENTER;
        rotationAnchor = TextAnchor.TOP_CENTER;
    }
}
```

refreshTicksHorizontal

```
if (isVerticalTickLabels()) {
    anchor = TextAnchor.BOTTOM_CENTER;
    rotationAnchor = TextAnchor.BOTTOM_CENTER;
    if (edge == RectangleEdge.LEFT) {
        angle = -Math.PI / 2.0;
    }
    else {
        angle = Math.PI / 2.0;
    }
}
else {
    if (edge == RectangleEdge.LEFT) {
        anchor = TextAnchor.CENTER_RIGHT;
        rotationAnchor = TextAnchor.CENTER_RIGHT;
    }
    else {
        anchor = TextAnchor.CENTER_LEFT;
        rotationAnchor = TextAnchor.CENTER_LEFT;
    }
}
```

refreshTicksVertical

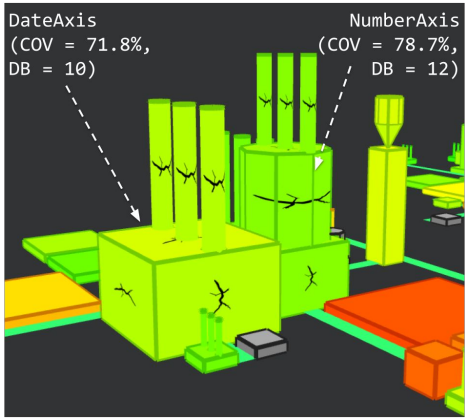
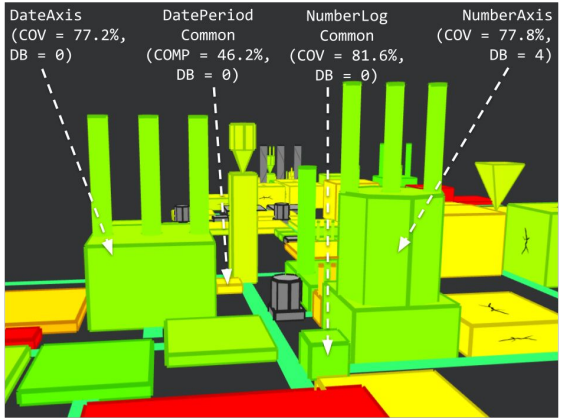
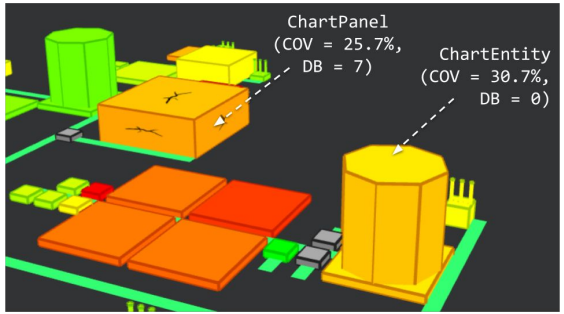
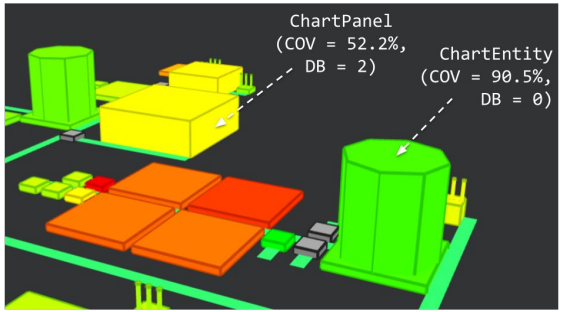
Common part

Variable part

Common part

Impact of fixing on the visualization

Project: **JFreeChart**
Red-to-green color scale: **test coverage**
Cracks: **# duplicated blocks**

	Before	After
Classes with duplicated blocks		
Classes lacking tests		

Impact of fixing on the classes metrics

Improvements of the quality metrics

- increased coverage
- decreased duplicated blocks

Class (in the org.jfree.chart package)		Coverage	Complexity
entity. ChartEntity	before	30.7 %	26
	AFTER	90.5 %	26
ChartPanel	before	25.7 %	322
	AFTER	52.2 %	295

Class (in the org.jfree.chart.axis package)		# duplicated blocks	Complexity
DateAxis	before	10	201
	AFTER	0	139
PeriodAxis	before	2	112
	AFTER	1	104
DatePeriodCommon	AFTER	0	8

Impact of fixing on the classes metrics

Improvements of the quality metrics

- increased coverage
- decreased duplicated blocks

Modifications also led to a **cognitive complexity improvement**

Class (in the org.jfree.chart package)		Coverage	Complexity
entity. ChartEntity	before	30.7 %	26
	AFTER	90.5 %	26
ChartPanel	before	25.7 %	322
	AFTER	52.2 %	295

Class (in the org.jfree.chart.axis package)		# duplicated blocks	Complexity
DateAxis	before	10	201
	AFTER	0	139
PeriodAxis	before	2	112
	AFTER	1	104
DatePeriodCommon	AFTER	0	8

Impact of fixing on the classes metrics

Improvements of the quality metrics

- increased coverage
- decreased duplicated blocks

Modifications also led to a **cognitive complexity improvement**

Added classes are not critical

Class (in the org.jfree.chart package)		Coverage	Complexity
entity. ChartEntity	before	30.7 %	26
	AFTER	90.5 %	26
ChartPanel	before	25.7 %	322
	AFTER	52.2 %	295

Class (in the org.jfree.chart.axis package)		# duplicated blocks	Complexity
DateAxis	before	10	201
	AFTER	0	139
PeriodAxis	before	2	112
	AFTER	1	104
DatePeriodCommon	AFTER	0	8

Future work

- Conducting an **empirical evaluation with real users** would help us in **validating our evaluation** and **identifying understandability limitations** of our visualization

- Explore deeper the relation between code smells and variability implementations (e.g. code duplication)

⇒ better understand how VariMetrics could be extended to match the industry's needs and expectations

Customizable Visualization of Quality Metrics for Object-Oriented Variability Implementations

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

OO variability implementations induce **technical debt** that **needs to be identified** as it **hampers the system's quality**

VariMetrics displays **OO quality metrics on VariCity**, a city visualization for OO variability implementations

The visualization exhibits **indebted zones of variability implementations**

Reproduction package:

https://doi.org/10.5281/zenodo.6644633

Obtained reproducibility badges

Artifacts Available

Functional & Reusable

Get the paper:

https://hal.archives-ouvertes.fr/hal-03717858/

VariMetrics website:

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

See you at the tool demo at 5:20 PM!