

# On Variability Debt in Object-Oriented Implementations

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

Université Côte d'Azur, CNRS, I3S, France

TD4ViS @ SPLC '22 – Graz, Austria

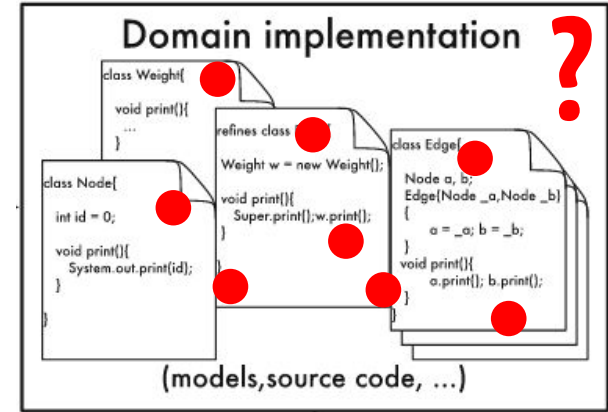
September 13, 2022

# OO variability implementations

Undocumented  
OO variability  
implementations

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

# Variation points and variants

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

```
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 | }
```

```
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 | }
```

# Variation points and variants

```
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

# Variation points and variants

```
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 |    }
```

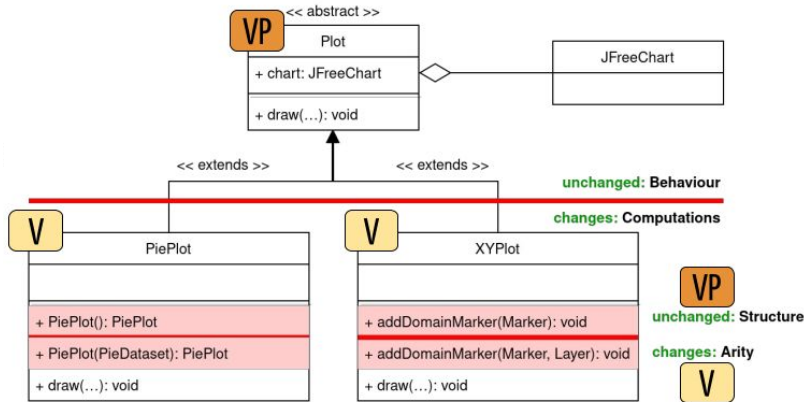
v\_Rectangle

```
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

# The reality of OO variability implementation

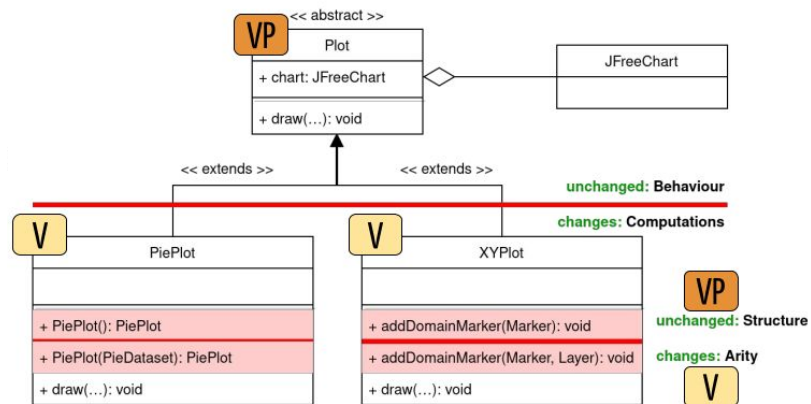
## Variability implemented using mechanisms





# The reality of OO variability implementation

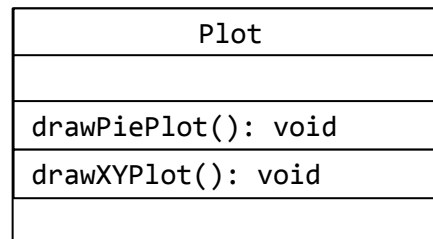
## Variability implemented using mechanisms



Duplicated blocks: 3

Duplicated blocks: 2

## Variability implemented without using mechanisms



Duplications



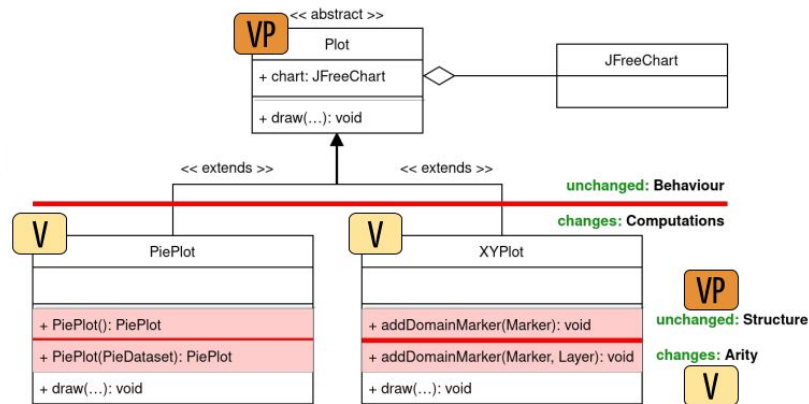
drawPiePlot drawXYPlot

Duplicated blocks: 25



# The reality of OO variability implementation

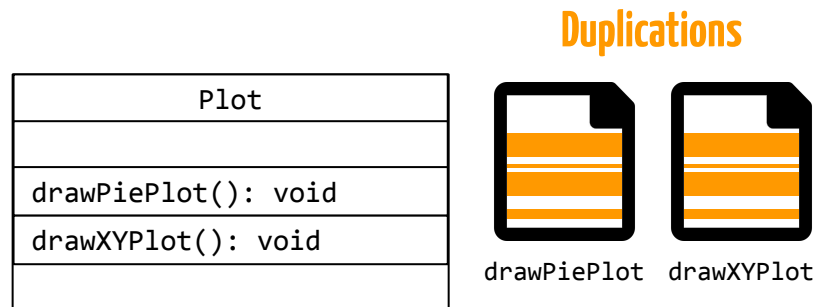
## Variability implemented using mechanisms



**Duplicated blocks: 3**  
**Code coverage: 80%**

**Duplicated blocks: 2**  
**Code coverage: 75%**

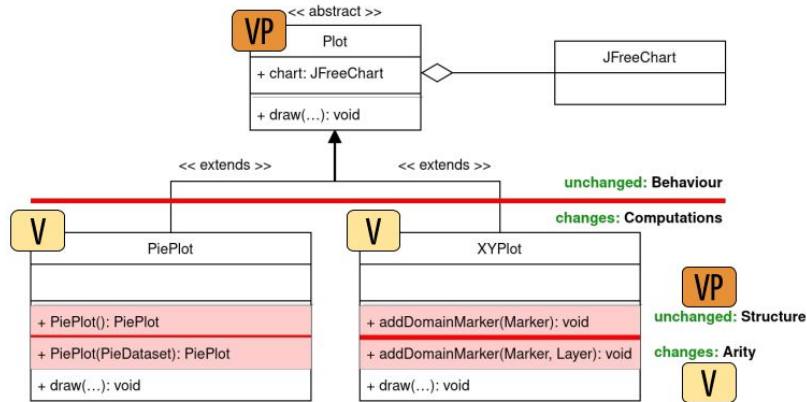
## Variability implemented without using mechanisms



**Duplicated blocks: 25**  
**Code coverage: 55%**

# The reality of OO variability implementation

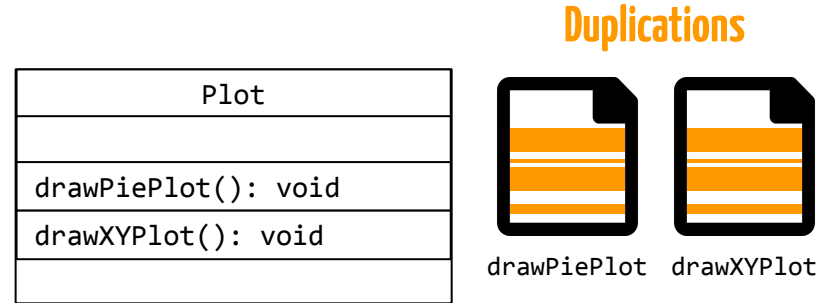
## Variability implemented using mechanisms



Duplicated blocks: 3  
Code coverage: 80%

Duplicated blocks: 2  
Code coverage: 75%

## Variability implemented without using mechanisms



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.”

Lack of knowledge of the implemented variability  
+ absence of traceability

**Variability model is rarely available**

**Documentations are rarely up to date and exhaustive**

**No mapping in the implementations**

no known variability

implementation mechanisms

⇒ artifact duplication + ↑ code complexity

**Identical implementation mechanisms  
for variability and business logic**

**OO variability implementations are prone to variability debt**

# How to measure OO variability debt?

## 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

# How to measure OO variability debt?

## Different types of variability debt **applicable to our experimented OO codebases**

- System-level structure quality issues  
in the implementation
- Code Duplication
- Lack of tests
- ~~Out of date or incomplete documentation~~  
no documentation
- ~~Architectural antipatterns~~  
no information on architecture
- ~~Expensive tests~~  
no information on architecture
- ~~Multi-version support~~  
no information on versions
- ~~Old technology in use~~  
no information on versions
- ~~Duplicate documentation~~  
no documentation
- ~~Poor test of feature interactions~~  
no list of features with mapping

# How to measure OO variability debt?

## Different types of variability debt

- System-level structure quality issues 
- Code Duplication 
- Lack of tests 

in the implementation

## Chosen OO metrics

Cognitive complexity

Duplicated code blocks

Unit tests coverage

# Example of code duplication

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

```
protected List refreshTicksHorizontal(Graphics2D g2,
    Rectangle2D dataArea, Rectangle2D edge) {
    List result = new java.util.ArrayList();
    Font tickLabelFont = getTickLabelFont();
    g2.setFont(tickLabelFont);
    if (isAutoTickUnitSelection()) {
        selectAutoTickUnit(g2, dataArea, edge);
    }
    Date tickUnit = getTickUnit();
    Date tickDate = calculateNextVisibleTickValue(unit);
    Date upperDate = getMaxUsableDate();
    boolean hasHole = false;
    while (tickDate.before(upperDate)) {
        // could add a flag to make the following correction optional...
        if (hasHole) {
            tickDate = correctTickDateForPosition(tickDate, unit,
                this.tickMarkPosition);
        }
        long lowestTickTime = tickDate.getTime();
        long distance = unit.addDate(tickDate, this.timeZone).getTime()
            - lowestTickTime;
        int minorTickSpaces = getMinorTickCount();
        if (minorTickSpaces == 0) {
            minorTickSpaces = unit.getMinorTickCount();
        }
        for (int minorTick = 1; minorTick <= minorTickSpaces; minorTick++) {
            long minorTickTime = lowestTickTime - distance
                + minorTick * minorTickSpaces;
            if (minorTickTime >= 0 && getRange().contains(minorTickTime)
                && (!isHoleDateValue(minorTickTime))) {
                result.add(new DateTick(TickType.MAJOR,
                    new Date(minorTickTime), "", TextAnchor.TOP_CENTER,
                    TextAnchor.CENTER, 0.0));
            }
        }
        if (!isHoleDateValue(tickDate.getTime())) {
            // work out the value, label and position
            String tickLabel;
            DateFormatter formatter = getDateFormatOverride();
            if (formatter != null) {
                tickLabel = formatter.format(tickDate);
            }
            else {
                tickLabel = this.tickUnit.dateToString(tickDate);
            }
            TextAnchor anchor, rotationAnchor;
            double angle = 0.0;
            if (isVerticalTickLabels()) {
                anchor = TextAnchor.CENTER_RIGHT;
                rotationAnchor = TextAnchor.CENTER_RIGHT;
                if (edge == RectangleEdge.TOP) {
                    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;
                }
            }
            Tick tick = new DateTick(tickDate, tickLabel, anchor,
                rotationAnchor, angle);
            result.add(tick);
            hasHole = false;
            long currentTickTime = tickDate.getTime();
            tickDate = unit.addDate(tickDate, this.timeZone);
            long nextTickTime = tickDate.getTime();
            for (int minorTick = 1; minorTick <= minorTickSpaces;
                minorTick++) {
                long minorTickTime = currentTickTime
                    + (nextTickTime - currentTickTime)
                    * (minorTick / minorTickSpaces);
                if (getRange().contains(minorTickTime)
                    && (!isHoleDateValue(minorTickTime))) {
                    result.add(new DateTick(TickType.MAJOR,
                        new Date(minorTickTime), "",
                            TextAnchor.TOP_CENTER, TextAnchor.CENTER,
                                0.0));
                }
            }
        }
        else {
            tickDate = unit.rollDate(tickDate, this.timeZone);
            hasHole = true;
            continue;
        }
    }
    return result;
}
```

Variable part

```
protected List refreshTicksVertical(Graphics2D g2,
    Rectangle2D dataArea, Rectangle2D edge) {
    List result = new java.util.ArrayList();
    Font tickLabelFont = getTickLabelFont();
    g2.setFont(tickLabelFont);
    if (isAutoTickUnitSelection()) {
        selectAutoTickUnit(g2, dataArea, edge);
    }
    Date tickUnit = getTickUnit();
    Date tickDate = calculateNextVisibleTickValue(unit);
    Date upperDate = getMaxUsableDate();
    boolean hasHole = false;
    while (tickDate.before(upperDate)) {
        // could add a flag to make the following correction optional...
        if (hasHole) {
            tickDate = correctTickDateForPosition(tickDate, unit,
                this.tickMarkPosition);
        }
        long lowestTickTime = tickDate.getTime();
        long distance = unit.addDate(tickDate, this.timeZone).getTime()
            - lowestTickTime;
        int minorTickSpaces = getMinorTickCount();
        if (minorTickSpaces == 0) {
            minorTickSpaces = unit.getMinorTickCount();
        }
        for (int minorTick = 1; minorTick <= minorTickSpaces; minorTick++) {
            long minorTickTime = lowestTickTime - distance
                + minorTick * minorTickSpaces;
            if (minorTickTime >= 0 && getRange().contains(minorTickTime)
                && (!isHoleDateValue(minorTickTime))) {
                result.add(new DateTick(TickType.MAJOR,
                    new Date(minorTickTime), "", TextAnchor.TOP_CENTER,
                    TextAnchor.CENTER, 0.0));
            }
        }
        if (!isHoleDateValue(tickDate.getTime())) {
            // work out the value, label and position
            String tickLabel;
            DateFormatter formatter = getDateFormatOverride();
            if (formatter != null) {
                tickLabel = formatter.format(tickDate);
            }
            else {
                tickLabel = this.tickUnit.dateToString(tickDate);
            }
            TextAnchor anchor, rotationAnchor;
            double angle = 0.0;
            if (isVerticalTickLabels()) {
                anchor = TextAnchor.BOTTOM_CENTER;
                rotationAnchor = TextAnchor.BOTTOM_CENTER;
                if (edge == RectangleEdge.LEFT) {
                    angle = -Math.PI / 2.0;
                }
            }
            else {
                anchor = TextAnchor.CENTER_LEFT;
                rotationAnchor = TextAnchor.CENTER_LEFT;
            }
            Tick tick = new DateTick(tickDate, tickLabel, anchor,
                rotationAnchor, angle);
            result.add(tick);
            hasHole = false;
            long currentTickTime = tickDate.getTime();
            tickDate = unit.addDate(tickDate, this.timeZone);
            long nextTickTime = tickDate.getTime();
            for (int minorTick = 1; minorTick <= minorTickSpaces;
                minorTick++) {
                long minorTickTime = currentTickTime
                    + (nextTickTime - currentTickTime)
                    * (minorTick / minorTickSpaces);
                if (getRange().contains(minorTickTime)
                    && (!isHoleDateValue(minorTickTime))) {
                    result.add(new DateTick(TickType.MAJOR,
                        new Date(minorTickTime), "",
                            TextAnchor.TOP_CENTER, TextAnchor.CENTER,
                                0.0));
                }
            }
        }
        else {
            tickDate = unit.rollDate(tickDate, this.timeZone);
            hasHole = true;
            continue;
        }
    }
    return result;
}
```

# (Obvious) Example of low coverage

```
3442 double ymin = yAxis.getLowerBound();
3443 double yymin = yAxis.valueToJava2D(ymin, area, getRangeAxisEdge());
3444
3445 double ymax = yAxis.getUpperBound();
3446 double yymax = yAxis.valueToJava2D(ymax, area, getRangeAxisEdge());
3447
3448 Rectangle2D[] r = new Rectangle2D[] {null, null, null, null};
3449 if (this.quadrantPaint[0] != null) {
3450     if (x > xmin && y < ymax) {
3451         if (this.orientation == PlotOrientation.HORIZONTAL) {
3452             r[0] = new Rectangle2D.Double(Math.min(yymax, yy),
3453                 Math.min(xxmin, xx), Math.abs(yy - yymax),
3454                 Math.abs(xx - xxmin));
3455         }
3456         else { // PlotOrientation.VERTICAL
3457             r[0] = new Rectangle2D.Double(Math.min(xxmin, xx),
3458                 Math.min(yymax, yy), Math.abs(xx - xxmin),
3459                 Math.abs(yy - yymax));
3460         }
3461         somethingToDraw = true;
3462     }
3463 }
3464 if (this.quadrantPaint[1] != null) {
3465     if (x < xmax && y < ymax) {
3466         if (this.orientation == PlotOrientation.HORIZONTAL) {
3467             r[1] = new Rectangle2D.Double(Math.min(yymax, yy),
3468                 Math.min(xxmax, xx), Math.abs(yy - yymax),
3469                 Math.abs(xx - xxmax));
3470         }
3471         else { // PlotOrientation.VERTICAL
3472             r[1] = new Rectangle2D.Double(Math.min(xx, xxmax),
```



org.jfree.chart.plot.XYPlot

refactor



pure  
technical debt

refactor



variability debt

```
protected Map drawAxes(Graphics2D g2, Rectangle2D plotArea,
    Rectangle2D dataArea, PlotRenderingInfo plotState) {
    AxisCollection axisCollection = new AxisCollection();

    // add domain axes to lists...
    for (CategoryAxis xAxis : this.domainAxes.values()) {
        if (xAxis != null) {
            int index = getDomainAxisIndex(xAxis);
            axisCollection.add(xAxis, getDomainAxisEdge(index));
        }
    }

    // add range axes to lists...
    for (ValueAxis yAxis : this.rangeAxes.values()) {
        if (yAxis != null) {
            int index = findRangeAxisIndex(yAxis);
            axisCollection.add(yAxis, getRangeAxisEdge(index));
        }
    }

    Map axisStateMap = new HashMap();

    // draw the top axes
    double cursor = dataArea.getMinY() - this.axisOffset.calculateTopOutset(
        dataArea.getHeight());
    Iterator iterator = axisCollection.getAxesAtTop().iterator();
    while (iterator.hasNext()) {
        Axis axis = (Axis) iterator.next();
        if (axis != null) {
            AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
                RectangleEdge.TOP, plotState);
            cursor = axisState.getCursor();
            axisStateMap.put(axis, axisState);
        }
    }

    // draw the bottom axes
    cursor = dataArea.getMaxY()
        + this.axisOffset.calculateBottomOutset(dataArea.getHeight());
    iterator = axisCollection.getAxesAtBottom().iterator();
    while (iterator.hasNext()) {
        Axis axis = (Axis) iterator.next();
        if (axis != null) {
            AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
                RectangleEdge.BOTTOM, plotState);
            cursor = axisState.getCursor();
            axisStateMap.put(axis, axisState);
        }
    }

    // draw the left axes
    cursor = dataArea.getMinX()
        - this.axisOffset.calculateLeftOutset(dataArea.getWidth());
    iterator = axisCollection.getAxesAtLeft().iterator();
    while (iterator.hasNext()) {
        Axis axis = (Axis) iterator.next();
        if (axis != null) {
            AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
                RectangleEdge.LEFT, plotState);
            cursor = axisState.getCursor();
            axisStateMap.put(axis, axisState);
        }
    }

    // draw the right axes
    cursor = dataArea.getMaxX()
        + this.axisOffset.calculateRightOutset(dataArea.getWidth());
    iterator = axisCollection.getAxesAtRight().iterator();
    while (iterator.hasNext()) {
        Axis axis = (Axis) iterator.next();
        if (axis != null) {
            AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
                RectangleEdge.RIGHT, plotState);
            cursor = axisState.getCursor();
            axisStateMap.put(axis, axisState);
        }
    }

    return axisStateMap;
}
```

```
protected void addRangesToList(AxisCollection axisCollection) {
    for (ValueAxis axis : this.rangeAxes.values()) {
        if (axis != null) {
            int index = findRangeAxisIndex(axis);
            axisCollection.add(axis, getRangeAxisEdge(index));
        }
    }
}
```

```
private void drawTheAxes(Iterator iterator, double cursor, Graphics2D g2, Rectangle2D plotArea, Rectangle2D
    dataArea, PlotRenderingInfo plotState, Map<Axis, AxisState> axisStateMap, RectangleEdge rectangleEdge) {
    while (iterator.hasNext()) {
        Axis axis = (Axis) iterator.next();
        if (axis != null) {
            AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
                rectangleEdge, plotState);
            cursor = axisState.getCursor();
            axisStateMap.put(axis, axisState);
        }
    }
}
```

Example of high cognitive complexity

```
protected Map drawAxes(Graphics2D g2, Rectangle2D plotArea,
    Rectangle2D dataArea, PlotRenderingInfo plotState) {
    AxisCollection axisCollection = new AxisCollection();

    // add domain axes to lists...
    for (CategoryAxis xAxis : this.domainAxes.values()) {
        if (xAxis != null) {
            int index = getDomainAxisIndex(xAxis);
            axisCollection.add(xAxis, getDomainAxisEdge(index));
        }
    }
}
```

```
// add range axes to lists...
for (ValueAxis yAxis : this.rangeAxes.values()) {
    if (yAxis != null) {
        int index = findRangeAxisIndex(yAxis);
        axisCollection.add(yAxis, getRangeAxisEdge(index));
    }
}
```

```
Map axisStateMap = new HashMap();

// draw the top axes
double cursor = dataArea.getMinY() - this.axisOffset.calculateTopOutset(
    dataArea.getHeight());
Iterator iterator = axisCollection.getAxesAtTop().iterator();
while (iterator.hasNext()) {
    Axis axis = (Axis) iterator.next();
    if (axis != null) {
        AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
            RectangleEdge.TOP, plotState);
        cursor = axisState.getCursor();
        axisStateMap.put(axis, axisState);
    }
}
```

```
// draw the bottom axes
cursor = dataArea.getMaxY()
    + this.axisOffset.calculateBottomOutset(dataArea.getHeight());
iterator = axisCollection.getAxesAtBottom().iterator();
while (iterator.hasNext()) {
    Axis axis = (Axis) iterator.next();
    if (axis != null) {
        AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
            RectangleEdge.BOTTOM, plotState);
        cursor = axisState.getCursor();
        axisStateMap.put(axis, axisState);
    }
}
```

```
// draw the left axes
cursor = dataArea.getMinX()
    - this.axisOffset.calculateLeftOutset(dataArea.getWidth());
iterator = axisCollection.getAxesAtLeft().iterator();
while (iterator.hasNext()) {
    Axis axis = (Axis) iterator.next();
    if (axis != null) {
        AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
            RectangleEdge.LEFT, plotState);
        cursor = axisState.getCursor();
        axisStateMap.put(axis, axisState);
    }
}
```

```
// draw the right axes
cursor = dataArea.getMaxX()
    + this.axisOffset.calculateRightOutset(dataArea.getWidth());
iterator = axisCollection.getAxesAtRight().iterator();
while (iterator.hasNext()) {
    Axis axis = (Axis) iterator.next();
    if (axis != null) {
        AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
            RectangleEdge.RIGHT, plotState);
        cursor = axisState.getCursor();
        axisStateMap.put(axis, axisState);
    }
}
```

```
return axisStateMap;
}
```

refactor

core part

```
protected Map<Axis, AxisState> drawAxes(Graphics2D g2, Rectangle2D plotArea,
    Rectangle2D dataArea, PlotRenderingInfo plotState) {
```

```
    AxisCollection axisCollection = new AxisCollection();
```

```
    // add domain axes to lists...
```

```
    for (ValueAxis axis : this.domainAxes.values()) {
        if (axis != null) {
            int axisIndex = findDomainAxisIndex(axis);
            axisCollection.add(axis, getDomainAxisEdge(axisIndex));
        }
    }
}
```

```
    return extractedDraw(g2, plotArea, dataArea, plotState, axisCollection);
}
```

variability resolution part

```
protected Map<Axis, AxisState> drawAxes(Graphics2D g2, Rectangle2D plotArea, Rectangle2D dataArea,
    PlotRenderingInfo plotState, AxisCollection axisCollection) {
    // add range axes to lists...
    addRangeAxes(axisCollection);

    Map<Axis, AxisState> axisStateMap = new HashMap<Axis, AxisState>();

    // draw the top axes
    Iterator iterator = axisCollection.getAxesAtTop().iterator();
    double cursor = dataArea.getMinY() - this.axisOffset.calculateTopOutset(dataArea.getHeight());
    drawRange(iterator, cursor, g2, plotArea, dataArea, plotState, axisStateMap, RectangleEdge.TOP);

    // draw the bottom axes
    iterator = axisCollection.getAxesAtBottom().iterator();
    cursor = dataArea.getMaxY() + this.axisOffset.calculateBottomOutset(dataArea.getHeight());
    drawRange(iterator, cursor, g2, plotArea, dataArea, plotState, axisStateMap, RectangleEdge.BOTTOM);

    // draw the left axes
    iterator = axisCollection.getAxesAtLeft().iterator();
    cursor = dataArea.getMinX() - this.axisOffset.calculateLeftOutset(dataArea.getWidth());
    drawRange(iterator, cursor, g2, plotArea, dataArea, plotState, axisStateMap, RectangleEdge.LEFT);

    // draw the right axes
    iterator = axisCollection.getAxesAtRight().iterator();
    cursor = dataArea.getMaxX() + this.axisOffset.calculateRightOutset(dataArea.getWidth());
    drawRange(iterator, cursor, g2, plotArea, dataArea, plotState, axisStateMap, RectangleEdge.RIGHT);

    return axisStateMap;
}
```

call

```
private void drawRange(Iterator iterator, double cursor, Graphics2D g2, Rectangle2D plotArea, Rectangle2D
    dataArea, PlotRenderingInfo plotState, Map<Axis, AxisState> axisStateMap, RectangleEdge rectangleEdge) {
    while (iterator.hasNext()) {
        Axis axis = (Axis) iterator.next();
        if (axis != null) {
            AxisState axisState = axis.draw(g2, cursor, plotArea, dataArea,
                rectangleEdge, plotState);
            cursor = axisState.getCursor();
            axisStateMap.put(axis, axisState);
        }
    }
}
```

variable part

# Limitations

- Variable parts in the code are visible but not the variability resolution
- Pure technical and variability debt have the same shape



**Thanks for your  
attention!**

Time for questions!