MPAndroidChart

Αναφορά Εντοπισμού Οσμών Κώδικα και Παραβιάσεων Αρχών Σχεδίασης

Δαμάσκος Σεραφείμ, mai24011

Μπάης Ευάγγελος, mai24032

Τσαούσης Άγγελος, mai25055

Χρυσοχοΐδης Αναστάσιος, mai25067

20 Ιαν 2025

Πίνακας Περιεχομένων

1. Εισαγωγή

1.1. Περιγραφή Λογισμικού

Το MPAndroidChart είναι μια δημοφιλής βιβλιοθήκη ανοιχτού κώδικα για την ανάπτυξη γραφημάτων σε εφαρμογές Android. Είναι γραμμένη σε Java και παρέχει τη δυνατότητα δημιουργίας διαφόρων τύπων γραφημάτων, όπως γραμμικά, ραβδογράμματα, πίτες, ραδαρικά, και πολλά άλλα. Ξεχωρίζει για την ευκολία ενσωμάτωσης, την υψηλή παραμετροποίηση και τις πλούσιες λειτουργίες, όπως ζουμ, κύλιση, και διαδραστικότητα. Είναι ιδανική για προγραμματιστές που θέλουν να προσθέσουν οπτικά ελκυστικά και λειτουργικά γραφήματα στις εφαρμογές τους.

1.2. Επισκόπηση - Μεθοδολογία

Η παρούσα αναφορά παρουσιάζει τα ευρήματα μας στην διαδικασία του εντοπισμού οσμών κώδικα και παραβιάσεων των Αρχών Σχεδίασης. Αναφέρονται 5 Bad Smells , με επεξήγηση στο ποιές Αρχές Σχεδίασης παραβιάζουν και στο πως επηρεάζουν την συντηρησιμότητα του project. Στη συνέχεια , προχωρήσαμε σε αναδομήσεις των συγκεκριμένων οσμών παρουσιάζοντας τον αρχικό κώδικα και και τον τελικό. Τέλος, συγκρίνουμε τις αναδομήσεις μας με τον αρχικό κώδικα από άποψη μετρικών και αναλύουμε πως κάθε αναδόμηση βοήθησε και στην βελτίωση των μετρικών και στην γενική βελτίωση του κώδικα

Για την ανάλυση και διόρθωση των Bad Smells χρησιμοποιήθηκε η τελευταία έκδοση του MPAndoridChart η 3.1.0. Για τον εντοπισμό των οσμών, χρησιμοποιήθηκε το εργαλείο SonarQube το οποίο είναι plugin στο JetBrains Intellij IDE. Με την προσθήκη του plugin, προχωρήσαμε στην ανάλυση όλου του Project και μας επιστράφηκαν όλα τα πιθανά προβλήματα που χρήζουν διόρθωσης. Τα αποτελέσματα της ανάλυσης χωρίζονται σε 5 κατηγορίες βάσει της επίπτωσης που θα έχει στην μελλοντική συντήρηση του project. Οι κατηγορίες είναι : Info, Low, Medium, High και Blocker.

Σε επιλεγμένα code smells, τα οποία έχουν χαρακτηρισμό High, επιλέξαμε να προχωρήσουμε σε αναδόμηση. Τις αναδομήσεις τις κάναμε μεμονωμένα, δηλαδή επιλέξαμε την οσμή προς διόρθωση, προχωρήσαμε σε Αναδόμηση, κάναμε commit στο αποθετήριό μας στο github και δημιουργήσαμε το tag. Για την επόμενη Αναδόμηση, κάναμε pull την αμέσως προηγούμενη έκδοση και προχωρήσαμε όπως περιγράφηκε. Στο αποθετήριο υπάρχουν στο σύνολο 6 tag, 1 η τελευταία έκδοση του project και άλλα 5 , 1 για κάθε αναδόμηση που κάναμε.

Πολλά από τα issues που μας επέστρεψε το SonarQube, αποτελούσαν σχολιασμένο κώδικα που δεν χρησιμοποιείται καθόλου, προσθέτοντας επιπλέον “θόρυβο” στην αναγνωσιμότητα του, κενές μέθοδοι δίχως λειτουργικότητα αυξάνοντας άσκοπα την μετρική LoC και σε πολλές περιπτώσεις public constructors σε Abstract κλάσεις. Στη συνέχεια παρουσιάζονται οι 5 οσμές μαζί

με τις απαραίτητες Αναδομήσεις, με μια σύντομη περιγραφή της φύσης του κάθε προβλήματος και των επιπτώσεων που μπορεί να έχει στη συντηρησιμότητα του συστήματος.

1.3. Αναφορές

Όλες οι εκδόσεις του original Project βρίσκονται στο ακόλουθο gituhub repository : https://github.com/PhilJay/MPAndroidChart .

Τα tags που δημιουργήθηκαν μετά από κάθε αναδόμηση βρίσκονται στο ακόλουθο repository : https://github.com/AngelosTsaousis/SoftwareEngineering3d/tags

2. Οσμές Κώδικα,Παραβιάσεων Αρχών Σχεδίασης και Αναδομήσεις

2.1. 1η Αναδόμηση

2.1.1. Οσμή - Πρόβλημα

Η παρακάτω λογική εμφανιζεται 21 φορές μέσα στο Project.

```
if (chart.isRotationEnabled())
    chart.setRotationEnabled(false);
else
    chart.setRotationEnabled(true);
```

Η οσμή η οποία παρατηρείται σε αυτές τις γραμμές κώδικα ονομαζεται: Πλεονάζον Κώδικας και Πλεοναστική Λογική. Τα προβλήματα με την συγκεκριμένη οσμή ειναι:

1. Η μέθοδος καλείται και στα δύο σκέλη της συνθήκης, κάτι που είναι περιττό και δυσκολεύει την ανάγνωση και συντήρηση του κώδικα.
2. Η λογική αυτής της συνθήκης μπορεί να απλοποιηθεί σε μία μόνο γραμμή.

Σε αυτή την περίπτωση ο κώδικας μας είναι δυσνοητος και επαναλαμβανεται η ίδια μεθοδος πολλες φορές κατι που αυξάνει την πιθανότητα σφαλμάτων κατά την συντήρηση (π.χ., αν αλλάξει η μέθοδος) και κανει πιο δυσκολη την προσαρμογή και την επέκταση στο μελλον.

2.1.2. Αναδόμηση

Μέσα από την αναδόμηση μας εξαλείψαμε τον πλεονασμό σε συνθήκες και μειώσαμε τις γραμμές κώδικα. Επίσης, το προγραμμα μας πλεον ειναι πιο εύκολα κατανοητό, πιο συντηρήσιμο και εχουμε μειώσει την πιθανότητα λάθους, στα συγκεκριμένα τμήματα κώδικα, σε κάποια μελλοντική αλλαγη.

```
if (chart.isRotationEnabled())
    chart.setRotationEnabled(false);
else
    chart.setRotationEnabled(true);
```

```
chart.setRotationEnabled(!chart.isRotationEnabled());
```

2.2. 2η Αναδόμηση

2.2.1. Οσμή - Πρόβλημα

Η ύπαρξη δύο ίδιων ή σχεδόν ίδιων τμημάτων κώδικα μέσα σε μία μέθοδο αποτελεί κλασικό παράδειγμα του code smell Πλεοναστικός Κώδικας (Duplicate Code):

```
float left = x - barWidthHalf;
float right = x + barWidthHalf;
float bottom, top;

if (mInverted) {
    bottom = y >= 0 ? y : 0;
    top = y <= 0 ? y : 0;
} else {
    top = y >= 0 ? y : 0;
    bottom = y <= 0 ? y : 0;
}

// multiply the height of the rect with the phase
if (top > 0)
    top *= phaseY;
else
    bottom *= phaseY;

addBar(left, top, right, bottom);
```

BarBuffer.java 62-80

```
float left = x - barWidthHalf;
float right = x + barWidthHalf;
float bottom, top;

if (mInverted) {
    bottom = y >= yStart ? y : yStart;
    top = y <= yStart ? y : yStart;
} else {
    top = y >= yStart ? y : yStart;
    bottom = y <= yStart ? y : yStart;
}

// multiply the height of the rect with the phase
top *= phaseY;
bottom *= phaseY;

addBar(left, top, right, bottom);
```

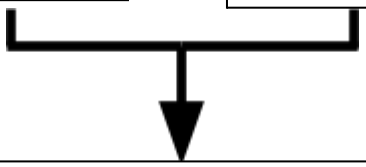
BarBuffer.java 107 - 123

Ο πλεοναστικός κώδικας μειώνει τη συντηρησιμότητα, καθώς απαιτεί αλλαγές σε πολλαπλά σημεία, αυξάνοντας τον κίνδυνο σφαλμάτων ή ασυνέπειας. Επιπλέον, αυξάνει την πολυπλοκότητα, κάνοντας τη μέθοδο μεγαλύτερη και πιο δύσκολη στην κατανόηση. Τέλος, παραβιάζει την αρχή DRY ("Don't Repeat Yourself"), που ενθαρρύνει την αποφυγή επαναλήψεων μέσω της επαναχρησιμοποίησης κώδικα.

2.2.2. Αναδόμηση

```
float left = x - barWidthHalf;
float right = x + barWidthHalf;
float bottom, top;

if (mInverted) {
    bottom = y >= 0 ? y : 0;
    top = y <= 0 ? y : 0;
} else {
    top = y >= 0 ? y : 0;
    bottom = y <= 0 ? y : 0;
}

// multiply the height of the rect with the phase
if (top > 0)
    top *= phaseY;
else
    bottom *= phaseY;

addBar(left, top, right, bottom);
```

```
float left = x - barWidthHalf;
float right = x + barWidthHalf;
float bottom, top;

if (mInverted) {
    bottom = y >= yStart ? y : yStart;
    top = y <= yStart ? y : yStart;
} else {
    top = y >= yStart ? y : yStart;
    bottom = y <= yStart ? y : yStart;
}

// multiply the height of the rect with the phase
top *= phaseY;
bottom *= phaseY;

addBar(left, top, right, bottom);
```



```
protected void calculateAndAddBar(float x,float barWidthHalf, float y, float value, boolean condition){
    float left = x - barWidthHalf;
    float right = x + barWidthHalf;
    float bottom;
    float top;

    if (mInverted) {
        bottom = Math.max(y, value);
        top = Math.min(y, value);
    } else {
        top = Math.max(y, value);
        bottom = Math.min(y, value);
    }

    if (condition){
        if (top > 0)
            top *= phaseY;
        else
            bottom *= phaseY;
    } else {
        top *= phaseY;
        bottom *= phaseY;
    }

    addBar(left, top, right, bottom);
}
```

Τα δυο κομμάτια κώδικα ενώθηκαν και έγιναν μια μέθοδος,η calculateAndAddBar, η οποία καλείται από τα δύο σημεία που εχουν πανομοιότυπο κώδικα:

```
calculateAndAddBar(x,barWidthHalf,y,0, true);
```
BarBuffer.java 61

```
calculateAndAddBar(x,barWidthHalf,y,yStart, false);
```
BarBuffer.java 87

Η αναδόμηση του πλεοναστικού κώδικα ενοποίησε δύο παρόμοιες λογικές σε μία επαναχρησιμοποιήσιμη μέθοδο (calculateAndAddBar), μειώνοντας την επανάληψη και βελτιώνοντας τη συντηρησιμότητα. Με τη χρήση παραμέτρων, καλύφθηκαν και οι δύο περιπτώσεις χωρίς πλεονασμό. Το αποτέλεσμα είναι πιο ευανάγνωστος και ευέλικτος κώδικας, που διευκολύνει την προσθήκη νέας λογικής και την πραγματοποίηση αλλαγών, βελτιώνοντας τη συνολική ποιότητα του προγράμματος.

2.3. 3η Αναδόμηση

2.3.1. Οσμή - Πρόβλημα

Στην κλάση **Legent.java** εντοπίστηκε η μέθοδος calculateDimensions(Paint, ViewPortHandler) η οποία είχε υψηλό Cognitive Complexity βάσει του SonarQube. Αποτελεί μια μεγάλη, πολύπλοκη μέθοδο η οποία είναι δύσκολη στην ανάγνωση, κατανόηση, στα test και στην επεξεργασία καθώς έχει πολλά επίπεδα ενθυλάκωσης, έχοντας πάνω από 1 αρμοδιότητες. Παρατηρείται λοιπόν πως η κλάση παραβιάζει την Αρχή της Μοναδικής Αρμοδιότητας μιας και εμπεριέχει μέθοδο που πάσχει από την οσμή “Long Method”.

```
public void calculateDimensions(Paint labelpaint, ViewPortHandler viewPortHandler) {

    float defaultFormSize = Utils.convertDpToPixel(mFormSize);
    float stackSpace = Utils.convertDpToPixel(mStackSpace);
    float formToTextSpace = Utils.convertDpToPixel(mFormToTextSpace);
    float xEntrySpace = Utils.convertDpToPixel(mXEntrySpace);
    float yEntrySpace = Utils.convertDpToPixel(mYEntrySpace);
    boolean wordWrapEnabled = mWordWrapEnabled;
    LegendEntry[] entries = mEntries;
    int entryCount = entries.length;

    mTextWidthMax = getMaximumEntryWidth(labelpaint);
    mTextHeightMax = getMaximumEntryHeight(labelpaint);

    switch (mOrientation) {
        case VERTICAL: {

            float maxWidth = 0f, maxHeight = 0f, width = 0f;
            float labelLineHeight = Utils.getLineHeight(labelpaint);
            boolean wasStacked = false;

            for (int i = 0; i < entryCount; i++) {

                LegendEntry e = entries[i];
                boolean drawingForm = e.form != LegendForm.NONE;
                float formSize = Float.isNaN(e.formSize)
                        ? defaultFormSize
                        : Utils.convertDpToPixel(e.formSize);
                String label = e.label;

                if (!wasStacked)
                    width = 0.f;

                if (drawingForm) {
                    if (wasStacked)
                        width += stackSpace;
                    width += formSize;
                }
```

Legend : 652-689

```
                // grouped forms have null labels
                if (label != null) {

                    // make a step to the left
                    if (drawingForm && !wasStacked)
                        width += formToTextSpace;
                    else if (wasStacked) {
                        maxWidth = Math.max(maxWidth, width);
                        maxHeight += labelLineHeight + yEntrySpace;
                        width = 0.f;
                        wasStacked = false;
                    }

                    width += Utils.calcTextWidth(labelpaint, label);

                    if (i < entryCount - 1)
                        maxHeight += labelLineHeight + yEntrySpace;
                } else {
                    wasStacked = true;
                    width += formSize;
                    if (i < entryCount - 1)
                        width += stackSpace;
                }

                maxWidth = Math.max(maxWidth, width);
            }

            mNeededWidth = maxWidth;
            mNeededHeight = maxHeight;

            break;
        }
        case HORIZONTAL: {

            float labelLineHeight = Utils.getLineHeight(labelpaint);
            float labelLineSpacing = Utils.getLineSpacing(labelpaint) + yEntrySpace;
            float contentWidth = viewPortHandler.contentWidth() * mMaxSizePercent;
```

Legend : 691-727

```
// Start calculating layout
float maxLineWidth = 0.f;
float currentLineWidth = 0.f;
float requiredWidth = 0.f;
int stackedStartIndex = -1;

mCalculatedLabelBreakPoints.clear();
mCalculatedLabelSizes.clear();
mCalculatedLineSizes.clear();

for (int i = 0; i < entryCount; i++) {

    LegendEntry e = entries[i];
    boolean drawingForm = e.form != LegendForm.NONE;
    float formSize = Float.isNaN(e.formSize)
            ? defaultFormSize
            : Utils.convertDpToPixel(e.formSize);
    String label = e.label;

    mCalculatedLabelBreakPoints.add(false);

    if (stackedStartIndex == -1) {
        // we are not stacking, so required width is for this label
        // only
        requiredWidth = 0.f;
    } else {
        // add the spacing appropriate for stacked labels/forms
        requiredWidth += stackSpace;
    }
```

Legend : 729-757

```
// grouped forms have null labels
if (label != null) {

    mCalculatedLabelSizes.add(Utils.calcTextSize(labelpaint, label));
    requiredWidth += drawingForm ? formToTextSpace + formSize : 0.f;
    requiredWidth += mCalculatedLabelSizes.get(i).width;
} else {

    mCalculatedLabelSizes.add(FSize.getInstance(0.f, 0.f));
    requiredWidth += drawingForm ? formSize : 0.f;

    if (stackedStartIndex == -1) {
        // mark this index as we might want to break here later
        stackedStartIndex = i;
    }
}

if (label != null || i == entryCount - 1) {

    float requiredSpacing = currentLineWidth == 0.f ? 0.f : xEntrySpace;

    if (!wordWrapEnabled // No word wrapping, it must fit.
            // The line is empty, it must fit
            || currentLineWidth == 0.f
            // It simply fits
            || (contentWidth - currentLineWidth >=
            requiredSpacing + requiredWidth)) {
        // Expand current line
        currentLineWidth += requiredSpacing + requiredWidth;
    } else { // It doesn't fit, we need to wrap a line

        // Add current line size to array
        mCalculatedLineSizes.add(FSize.getInstance(currentLineWidth, labelLineHeight));
        maxLineWidth = Math.max(maxLineWidth, currentLineWidth);

        // Start a new line
        mCalculatedLabelBreakPoints.set(
                stackedStartIndex > -1 ? stackedStartIndex
                        : i, true);
        currentLineWidth = requiredWidth;
    }
```

Legend : 759-799

```
                if (i == entryCount - 1) {
                    // Add last line size to array
                    mCalculatedLineSizes.add(FSize.getInstance(currentLineWidth, labelLineHeight));
                    maxLineWidth = Math.max(maxLineWidth, currentLineWidth);
                }
            }

            stackedStartIndex = label != null ? -1 : stackedStartIndex;
        }

        mNeededWidth = maxLineWidth;
        mNeededHeight = labelLineHeight
                * (float) (mCalculatedLineSizes.size())
                + labelLineSpacing *
                (float) (mCalculatedLineSizes.size() == 0
                        ? 0
                        : (mCalculatedLineSizes.size() - 1));

        break;
    }
}

mNeededHeight += mYOffset;
mNeededWidth += mXOffset;
}
```

Legend : 801-821

2.3.2. Αναδόμηση

Συνεπώς, εφαρμόσαμε την αναδόμηση “Extract Method”, όπου την αρχική μέθοδο την σπάμε σε μικρότερες προκειμένου να διαχωριστούν οι αρμοδιότητες. Συγκεκριμένα, από την μέθοδο calculatedDimensions() προέκυψαν 2 επιμέρους μέθοδοι , οι CalculateVerticalDimension() και caluclateHorizontalDimension(), που οποίες αντικατέστησαν μια switch με 2 cases εντός μεθόδου , μειώνοντας την πολυπλοκότητα. Συγκεκριμένα ο κώδικας είχε ως εξής :

```
public void calculateDimensions(Paint labelPaint, ViewPortHandler viewPortHandler) {
    final float defaultFormSize = Utils.convertDpToPixel(mFormSize);
    final float stackSpace = Utils.convertDpToPixel(mStackSpace);
    final float formToTextSpace = Utils.convertDpToPixel(mFormToTextSpace);
    final float xEntrySpace = Utils.convertDpToPixel(mXEntrySpace);
    final float yEntrySpace = Utils.convertDpToPixel(mYEntrySpace);

    mTextWidthMax = getMaximumEntryWidth(labelPaint);
    mTextHeightMax = getMaximumEntryHeight(labelPaint);

    if (mOrientation == LegendOrientation.VERTICAL) {
        calculateVerticalDimensions(labelPaint, stackSpace, formToTextSpace, yEntrySpace, defaultFormSize);
    } else if (mOrientation == LegendOrientation.HORIZONTAL) {
        calculateHorizontalDimensions(labelPaint, viewPortHandler, stackSpace, formToTextSpace, xEntrySpace, yEntrySpace, defaultFormSize);
    }

    mNeededHeight += mYOffset;
    mNeededWidth += mXOffset;
}
```

Legend 652-670

```
private void calculateVerticalDimensions(Paint labelPaint, float stackSpace, float formToTextSpace,
                                         float yEntrySpace, float defaultFormSize) {
    float maxWidth = 0f, maxHeight = 0f, lineWidth = 0f;
    final float labelLineHeight = Utils.getLineHeight(labelPaint);
    boolean wasStacked = false;

    for (LegendEntry entry : mEntries) {
        boolean hasForm = entry.form != LegendForm.NONE;
        float formSize = Float.isNaN(entry.formSize) ? defaultFormSize : Utils.convertDpToPixel(entry.formSize);
        String label = entry.label;

        if (!wasStacked) lineWidth = 0f;

        if (hasForm) {
            if (wasStacked) lineWidth += stackSpace;
            lineWidth += formSize;
        }

        if (label != null) {
            if (hasForm && !wasStacked) lineWidth += formToTextSpace;
            else if (wasStacked) {
                maxWidth = Math.max(maxWidth, lineWidth);
                maxHeight += labelLineHeight + yEntrySpace;
                lineWidth = 0f;
                wasStacked = false;
            }
            lineWidth += Utils.calcTextWidth(labelPaint, label);
            if (isNotLastEntry(entry)) maxHeight += labelLineHeight + yEntrySpace;
        } else {
            wasStacked = true;
            lineWidth += formSize + stackSpace;
        }

        maxWidth = Math.max(maxWidth, lineWidth);
    }

    mNeededWidth = maxWidth;
    mNeededHeight = maxHeight;
}
```

Legend 672-710

```
private void calculateHorizontalDimensions(Paint labelPaint, ViewPortHandler viewPortHandler, float stackSpace,
                                           float formToTextSpace, float xEntrySpace, float yEntrySpace, float defaultFormSize) {
    final float labelLineHeight = Utils.getLineHeight(labelPaint);
    final float labelLineSpacing = Utils.getLineSpacing(labelPaint) + yEntrySpace;
    final float contentWidth = viewPortHandler.contentWidth() * mMaxSizePercent;

    float maxLineWidth = 0f, currentLineWidth = 0f, requiredWidth = 0f;
    int stackedStartIndex = -1;

    mCalculatedLabelBreakPoints.clear();
    mCalculatedLabelSizes.clear();
    mCalculatedLineSizes.clear();

    for (int i = 0; i < mEntries.length; i++) {
        LegendEntry entry = mEntries[i];
        boolean hasForm = entry.form != LegendForm.NONE;
        float formSize = Float.isNaN(entry.formSize) ? defaultFormSize : Utils.convertDpToPixel(entry.formSize);
        String label = entry.label;

        mCalculatedLabelBreakPoints.add(false);
        requiredWidth = (stackedStartIndex == -1) ? 0f : requiredWidth + stackSpace;

        if (label != null) {
            mCalculatedLabelSizes.add(Utils.calcTextSize(labelPaint, label));
            requiredWidth += (hasForm ? formToTextSpace + formSize : 0f) + mCalculatedLabelSizes.get(i).width;
        } else {
            mCalculatedLabelSizes.add(FSize.getInstance(0f, 0f));
            requiredWidth += (hasForm ? formSize : 0f);
            if (stackedStartIndex == -1) stackedStartIndex = i;
        }

        if (label != null || i == mEntries.length - 1) {
            float requiredSpacing = (currentLineWidth == 0f) ? 0f : xEntrySpace;
            if (!mWordWrapEnabled || currentLineWidth == 0f || contentWidth - currentLineWidth >= requiredSpacing + requiredWidth) {
                currentLineWidth += requiredSpacing + requiredWidth;
            } else {
                mCalculatedLineSizes.add(FSize.getInstance(currentLineWidth, labelLineHeight));
                maxLineWidth = Math.max(maxLineWidth, currentLineWidth);
                mCalculatedLabelBreakPoints.set((stackedStartIndex > -1) ? stackedStartIndex : i, true);
                currentLineWidth = requiredWidth;
            }

            if (i == mEntries.length - 1) {
                mCalculatedLineSizes.add(FSize.getInstance(currentLineWidth, labelLineHeight));
                maxLineWidth = Math.max(maxLineWidth, currentLineWidth);
            }
        }
```

Legend 712-765

Με την παραπάνω αναδόμηση “Extract Method” απλοποιήθηκε η αναγνωσιμότητα καθιστώντας την κλάση πιο εύκολη στην συντήρηση .Ταυτόχρονα εξάγωντας 2 επιμέρους μεθόδους, τηρείται η αρχή της Μοναδικής Αρμοδιότητας και η λειτουργικότητα των μεθόδων μπορεί να αξιοποιηθεί για μελλοντική χρήση αποφεύγοντας την συγγραφή διπτότυπου κώδικα.

2.4. 4η Αναδόμηση

2.4.1. Οσμή - Πρόβλημα

Η μέθοδος feed() είναι αρκετά σύνθετη και περιέχει πολλές εντολές και διακλαδώσεις που αυξάνουν την γνωσιακή πολυπλοκότητα (cognitive complexity). Η οσμή αυτή σχετίζεται με το πόσο δύσκολο είναι για έναν προγραμματιστή να κατανοήσει, να διατηρήσει και να εξελίξει τον κώδικα της κλάσης. Όσο πιο περίπλοκη είναι η κλάση, τόσο μεγαλύτερες είναι οι πιθανότητες να προκύψουν προβλήματα.

```
public void feed(IBarDataSet data) {

    float size = data.getEntryCount() * phaseX;
    float barWidthHalf = mBarWidth / 2f;

    for (int i = 0; i < size; i++) {

        BarEntry e = data.getEntryForIndex(i);

        if(e == null)
            continue;

        float x = e.getX();
        float y = e.getY();
        float[] vals = e.getYVals();

        if (!mContainsStacks || vals == null) {

            float bottom = x - barWidthHalf;
            float top = x + barWidthHalf;
            float left, right;
            if (mInverted) {
                left = y >= 0 ? y : 0;
                right = y <= 0 ? y : 0;
            } else {
                right = y >= 0 ? y : 0;
                left = y <= 0 ? y : 0;
            }

            // multiply the height of the rect with the phase
            if (right > 0)
                right *= phaseY;
            else
                left *= phaseY;

            addBar(left, top, right, bottom);

        } else {

            float posY = 0f;
            float negY = -e.getNegativeSum();
```

```
                float yStart = 0f;

                // fill the stack
                for (int k = 0; k < vals.length; k++) {

                    float value = vals[k];

                    if (value >= 0f) {
                        y = posY;
                        yStart = posY + value;
                        posY = yStart;
                    } else {
                        y = negY;
                        yStart = negY + Math.abs(value);
                        negY += Math.abs(value);
                    }

                    float bottom = x - barWidthHalf;
                    float top = x + barWidthHalf;
                    float left, right;
                    if (mInverted) {
                        left = y >= yStart ? y : yStart;
                        right = y <= yStart ? y : yStart;
                    } else {
                        right = y >= yStart ? y : yStart;
                        left = y <= yStart ? y : yStart;
                    }

                    // multiply the height of the rect with the phase
                    right *= phaseY;
                    left *= phaseY;

                    addBar(left, top, right, bottom);
                }
            }
        }

        reset();
    }
```

2.4.2. Αναδόμηση

Μείωση Πολυπλοκότητας: Η feed() περιέχει πλέον μόνο μια λογική υψηλού επιπέδου και αναθέτει τις λεπτομέρειες στις βοηθητικές μεθόδους processSingleBar, processStackedBar και addStackedBar.

```
private void processSingleBar(BarEntry e, float barWidthHalf) {
    float x = e.getX();
    float y = e.getY();
    float bottom = x - barWidthHalf;
    float top = x + barWidthHalf;
    float left, right;

    if (mInverted) {
        left = y >= 0 ? y : 0;
        right = y <= 0 ? y : 0;
    } else {
        right = y >= 0 ? y : 0;
        left = y <= 0 ? y : 0;
    }

    if (right > 0) right *= phaseY;
    else left *= phaseY;

    addBar(left, top, right, bottom);
}
```

```
private void processStackedBar(BarEntry e, float barWidthHalf) {
    float[] vals = e.getYVals();
    float posY = 0f;
    float negY = -e.getNegativeSum();

    for (float value : vals) {
        float y, yStart;
        if (value >= 0f) {
            y = posY;
            yStart = posY + value;
            posY = yStart;
        } else {
            y = negY;
            yStart = negY + Math.abs(value);
            negY += Math.abs(value);
        }

        addStackedBar(e.getX(), y, yStart, barWidthHalf);
    }
}
```

```
private void addStackedBar(float x, float y, float yStart, float barWidthHalf) {
    float bottom = x - barWidthHalf;
    float top = x + barWidthHalf;
    float left, right;

    if (mInverted) {
        left = y >= yStart ? y : yStart;
        right = y <= yStart ? y : yStart;
    } else {
        right = y >= yStart ? y : yStart;
        left = y <= yStart ? y : yStart;
    }

    right *= phaseY;
    left *= phaseY;

    addBar(left, top, right, bottom);
}
```

Επαναχρησιμοποίηση: Η μέθοδος addStackedBar επαναχρησιμοποιείται τόσο για stackable όσο και για μη stackable δεδομένα. Δομημένος Κώδικας: Κάθε υπο-μέθοδος επικεντρώνεται σε μία μόνο λειτουργία. Με τις αλλαγές που πραγματοποιήθηκαν πάνω, η γνωσιακή πολυπλοκότητα της feed() μειώνεται σημαντικά και είναι πλέον πιο εύκολη στην

κατανόηση και συντήρηση.

2.5. 5η Αναδόμηση

2.5.1. Οσμή - Πρόβλημα

Η μέθοδος computeLegend έχει το πρόβλημα γνωστό ως “Brain Method”.

```
public void computeLegend(ChartData<?> data) {

    if (!mLegend.isLegendCustom()) {

        computedEntries.clear();

        // loop for building up the colors and labels used in the legend
        for (int i = 0; i < data.getDataSetCount(); i++) {

            IDataSet dataSet = data.getDataSetByIndex(i);

            List<Integer> clrs = dataSet.getColors();
            int entryCount = dataSet.getEntryCount();

            // if we have a barchart with stacked bars
            if (dataSet instanceof IBarDataSet && ((IBarDataSet) dataSet).isStacked()) {

                IBarDataSet bds = (IBarDataSet) dataSet;
                String[] sLabels = bds.getStackLabels();

                for (int j = 0; j < clrs.size() && j < bds.getStackSize(); j++) {

                    computedEntries.add(new LegendEntry(
                            sLabels[j % sLabels.length],
                            dataSet.getForm(),
                            dataSet.getFormSize(),
                            dataSet.getFormLineWidth(),
                            dataSet.getFormLineDashEffect(),
                            clrs.get(j)
                    ));
                }

                if (bds.getLabel() != null) {
                    // add the legend description label
                    computedEntries.add(new LegendEntry(
                            dataSet.getLabel(),
                            Legend.LegendForm.NONE,
                            Float.NaN,
                            Float.NaN,
                            null,
                            ColorTemplate.COLOR_NONE
                    ));
                }

            } else if (dataSet instanceof IPieDataSet) {
```

```
    IPieDataSet pds = (IPieDataSet) dataSet;

    for (int j = 0; j < clrs.size() && j < entryCount; j++) {

        computedEntries.add(new LegendEntry(
                pds.getEntryForIndex(j).getLabel(),
                dataSet.getForm(),
                dataSet.getFormSize(),
                dataSet.getFormLineWidth(),
                dataSet.getFormLineDashEffect(),
                clrs.get(j)
        ));
    }

    if (pds.getLabel() != null) {
        // add the legend description label
        computedEntries.add(new LegendEntry(
                dataSet.getLabel(),
                Legend.LegendForm.NONE,
                Float.NaN,
                Float.NaN,
                null,
                ColorTemplate.COLOR_NONE
        ));
    }

} else if (dataSet instanceof ICandleDataSet && ((ICandleDataSet) dataSet).getDecreasingColor() !=
        ColorTemplate.COLOR_NONE) {

    int decreasingColor = ((ICandleDataSet) dataSet).getDecreasingColor();
    int increasingColor = ((ICandleDataSet) dataSet).getIncreasingColor();

    computedEntries.add(new LegendEntry(
            null,
            dataSet.getForm(),
            dataSet.getFormSize(),
            dataSet.getFormLineWidth(),
            dataSet.getFormLineDashEffect(),
            decreasingColor
    ));

    computedEntries.add(new LegendEntry(
            dataSet.getLabel(),
            dataSet.getForm(),
            dataSet.getFormSize(),
```

```
                        dataSet.getFormLineWidth(),
                        dataSet.getFormLineDashEffect(),
                        increasingColor
                ));

            } else { // all others

                for (int j = 0; j < clrs.size() && j < entryCount; j++) {

                    String label;

                    // if multiple colors are set for a DataSet, group them
                    if (j < clrs.size() - 1 && j < entryCount - 1) {
                        label = null;
                    } else { // add label to the last entry
                        label = data.getDataSetByIndex(i).getLabel();
                    }

                    computedEntries.add(new LegendEntry(
                            label,
                            dataSet.getForm(),
                            dataSet.getFormSize(),
                            dataSet.getFormLineWidth(),
                            dataSet.getFormLineDashEffect(),
                            clrs.get(j)
                    ));
                }
            }
        }

        if (mLegend.getExtraEntries() != null) {
            Collections.addAll(computedEntries, mLegend.getExtraEntries());
        }

        mLegend.setEntries(computedEntries);
    }

    Typeface tf = mLegend.getTypeface();

    if (tf != null)
        mLegendLabelPaint.setTypeface(tf);

    mLegendLabelPaint.setTextSize(mLegend.getTextSize());
    mLegendLabelPaint.setColor(mLegend.getTextColor());
```

Η έννοια της **"Brain Method"** αναφέρεται σε μια μέθοδο που είναι υπερβολικά περίπλοκη επειδή εκτελεί πάρα πολλά πράγματα ταυτόχρονα. Αυτό συνήθως παραβιάζει την αρχή της **μοναδικής ευθύνης** (Single Responsibility Principle). Κάποια από τα πιο συνηθισμένα προβλήματα που προκαλεί μια τέτοια μέθοδος ειναι:

- **Δυσκολία Κατανόησης**
- **Δυσκολία Συντήρησης**
- **Περιορισμένη Επαναχρησιμοποίηση**
- **Υψηλή Γνωσιακή Πολυπλοκότητα**
- **Δυσκολία στον Έλεγχο**

2.5.2. Αναδόμηση

Η μέθοδος computeLegend έχει αναδιαμορφωθεί ώστε να είναι πιο καθαρή και κατανοητή. Κάθε τύπος IDataSet επεξεργάζεται σε ξεχωριστές μεθόδους (processBarDataSet, processPieDataSet, κ.λπ.), βελτιώνοντας την αναγνωσιμότητα και μειώνοντας τη γνωστική πολυπλοκότητα. Επίσης, δημιουργήθηκε μέθοδος applyLegendStyle για την εφαρμογή στυλ

```
private void processBarDataSet(IBarDataSet dataSet, List<Integer> colors) {
    String[] stackLabels = dataSet.getStackLabels();

    for (int j = 0; j < colors.size() && j < dataSet.getStackSize(); j++) {
        computedEntries.add(new LegendEntry(
                stackLabels[j % stackLabels.length],
                dataSet.getForm(),
                dataSet.getFormSize(),
                dataSet.getFormLineWidth(),
                dataSet.getFormLineDashEffect(),
                colors.get(j)
        ));
    }

    if (dataSet.getLabel() != null) {
        addDescriptionLabel(dataSet.getLabel());
    }
}
```

```
private void processPieDataSet(IPieDataSet dataSet, List<Integer> colors, int entryCount) {
    for (int j = 0; j < colors.size() && j < entryCount; j++) {
        computedEntries.add(new LegendEntry(
                dataSet.getEntryForIndex(j).getLabel(),
                dataSet.getForm(),
                dataSet.getFormSize(),
                dataSet.getFormLineWidth(),
                dataSet.getFormLineDashEffect(),
                colors.get(j)
        ));
    }

    if (dataSet.getLabel() != null) {
        addDescriptionLabel(dataSet.getLabel());
    }
}
```

```
private void applyLegendStyle() {
    Typeface typeface = mLegend.getTypeface();
    if (typeface != null) {
        mLegendLabelPaint.setTypeface(typeface);
    }

    mLegendLabelPaint.setTextSize(mLegend.getTextSize());
    mLegendLabelPaint.setColor(mLegend.getTextColor());
}
```

3. Αποτελέσματα - Εξέλιξη Ποιότητας

3.1. Πίνακας Μετρικών

Μετρικές	Αρχική Έκδοση	Έκδοση 1	Έκδοση 2	Έκδοση 3	Έκδοση 4	Έκδοση 5
LoC	24310	24188	24178	24116	24109	24086
CBO	1245	1245	1245	1246	1248	1249
LCOM	30940	30940	30945	31086	31092	31143
WMC*	361.114	357.628	357.394	357.322	352.822	348.405

3.2. Σχολιασμός ανα αναδόμηση

Σχετικά με την εξέλιξη της **1ης αναδόμησης**. Παρατηρείται μείωση στην μετρική **LoC (Lines of Code)**. Οι γραμμες κώδικα μειώθηκαν καθως αυτος ηταν ο στοχος της αναδόμησης.

Περνώντας στην **2η αναδόμηση**, πέρα από την μείωση της προαναφερθείσας μετρικής LoC, παρατηρείται μία μικρή αύξηση της τάξεως τον 5 μονάδων στην μετρική **LCOM (Lack of Cohesion of Methods)** και μια ελάχιστη μείωση στην μετρική ***WMC**** **(Weighted Methods per Class)**. Οι παραπάνω εξελίξεις θα αναλυθούν εκτενώς παρακάτω.

Στην **3η αναδόμηση** έχουμε επίσης μείωση στις γραμμές του κώδικα και στην μετρική WMC*, καθως και ελάχιστες αυξήσεις στις μετρικές **CBO (Coupling Between Objects)** και **LCOM.**

Στην **4η αναδόμηση** η μεγαλύτερη μεταβολή είναι αυτή της μετρικής WMC*, καθως οι συγκεκριμένη μετρική επηρεάζεται και από την τελευταία αναδόμηση, θα υπάρξει γενικός σχολιασμός στην συνέχεια.

Τέλος, περνώντας στην **5η αναδόμηση** παρατηρούμε οτι μειώθηκαν οι γραμμές κώδικα και η μετρική WMC*. Ειχαμε ελάχιστη αύξηση στην μετρική CBO καθως και μεγάλη αύξηση στην μετρική LCOM.

3.3. Αποτελέσματα

Συμψηφίζοντας τα αποτελεσματα απο ολες τις αναδομήσεις. Εχουμε εμφανή μείωση στις γραμμές του κώδικα απο 24310 σε 24086, κατι που επετεύχθη μεσα απο την αφαίρεση περιττού κώδικα και την απλοποίηση πολλών block κώδικα. Μέσα από αυτό έχουμε ενα κώδικα πιο κατανοητό και πιο συντηρήσιμο στο μελλον.

Η αύξηση της μετρικής CBO μπορεί να υποδηλώνει ελαφρώς αυξημένη σύζευξη μεταξύ των αντικειμένων αλλα λογω της μικρής αύξησης μπορει να θεωρηθει ανεκτή.

Η αυξηση των 200 περιπου μονάδων, της μετρικης LCOM σε πρώτη όψη θα μπορουσε να θεωρηθει ανησυχητική. Αν παρατηρήσουμε ομως, τις αναδομήσεις 2 ,3 ,4 και 5 έχουμε να κάνουμε με αναδομήσεις που ως τελικο αποτέλεσμα εχουν την εξαγωγή νεων μεθοδων που σκοπο εχουν να

απλοποιήσουν των κώδικα, να τον κάνουν πιο κατανοητό και πιο συντηρήσιμο σε ενα μελλοντικό μετασχηματισμό. Επίσης, σκοπός τον αναδομήσεων ηταν οι απλοποιήσεις συγκεκριμένων κλάσεων που ειχαν ως αποτελεσμα τον διαχωρισμό των αρμοδιοτήτων (βλ 5η Αναδόμηση “Brain Method”) .

Τελος, έχουμε μείωση για την μετρική WMC*. Η μείωση αυτή είναι θετική, καθώς λιγότερες σταθμισμένες μέθοδοι ανά κλάση συνήθως σημαίνουν λιγότερη πολυπλοκότητα ανά κλάση. Αυτό μπορεί να οδηγήσει σε πιο απλή δομή και καλύτερη συντηρησιμότητα.