OpenRefine

Εντοπισμός Code Smells ή/και Παραβιάσεων Αρχών Σχεδίασης,Refactoring,Git

Κωνσταντίνος Θωμασιάδης, mai25016

Ευστάθιος Ιωσηφίδης, mai25017

Αικατερίνη Κρότκα, mai25031

Πίνακας Περιεχομένων

1. Περιγραφή Λογισμικού

Το **OpenRefine** είναι μια εφαρμογή επιφάνειας εργασίας ανοιχτού κώδικα που χρησιμοποιείται για την εκκαθάριση και τη μετατροπή δεδομένων σε άλλες μορφές, μια διαδικασία που συχνά ονομάζεται **"data wrangling"**. Το OpenRefine λειτουργεί με σύνολα δεδομένων που είναι δομημένα σε σειρές και στήλες, όπως αυτά που συναντώνται σε υπολογιστικά φύλλα (π.χ., **CSV**) ή βάσεις δεδομένων.

Ωστόσο, σε αντίθεση με τα παραδοσιακά υπολογιστικά φύλλα, το OpenRefine προσφέρει λειτουργικότητα που θυμίζει περισσότερο τις σχεσιακές βάσεις δεδομένων, επιτρέποντας **φιλτράρισμα, ομαδοποίηση και καθαρισμό δεδομένων** με εξειδικευμένα εργαλεία και κριτήρια. Μερικές από τις κύριες δυνατότητες του περιλαμβάνουν:

- **Εκκαθάριση Δεδομένων**: Αναγνώριση και διόρθωση ασυνεπειών στα δεδομένα.
- **Μετατροπή Δεδομένων**: Μετασχηματισμός δεδομένων μεταξύ διαφορετικών μορφών, π.χ., από CSV σε JSON ή XML.
- **Εργαλεία Σύνδεσης (Linking)**: Ενσωμάτωση δεδομένων από εξωτερικές πηγές - διασυνδέσεις με API.
- **Διαδραστικό Περιβάλλον**: Χρήση φιλικής διεπαφής που επιτρέπει τη γρήγορη διαχείριση μεγάλων συνόλων δεδομένων.

Η ανάλυση έγινε για το version 3.8.7

1.2. Code Smells και παραβιάσεις Αρχών Σχεδίασης

1. Στην πρώτη έκδοση εντοπίστηκε το code smell **duplicate code** στην κλάση RangeFacet**.** Και στις δύο υλοποιήσεις της μεθόδου computeChoices, ένα μεγάλο μέρος της λογικής είναι ακριβώς το ίδιο. Η βέλτιστη πρακτική είναι να αφαιρεθεί ο επαναλαμβανόμενος κώδικας σε μια κοινή βοηθητική μέθοδο, ώστε να γίνει ο κώδικας πιο συντηρήσιμος και καθαρός.
2. Στη δεύτερη έκδοση εντοπίστηκε η παραβίαση της Αρχής Μοναδικής Αρμοδιότητας (**Single Responsibility Principle - SRP**) στην κλάση ColumnReorderChange. Αυτή η Αρχή Σχεδίασης ορίζει ότι μία κλάση ή μέθοδος πρέπει να έχει μόνο έναν λόγο να αλλάξει. Όταν αυτό παραβιάζεται μειώνεται η επαναχρησιμότητα της καθως περιέχει

λογική που δεν είναι σχετική με όλα τα σενάρια ενώ παράλληλα ο συνδυασμός πολλαπλών λειτουργιών σε μία μέθοδο αυξάνει την πολυπλοκότητα, καθιστώντας τον κώδικα δύσκολο στην ανάγνωση και κατανόηση.

3. Στην τρίτη έκδοση εντοπίστηκε πάλι παραβίαση της Αρχής Μοναδικής Αρμοδιότητας (**Single Responsibility Principle - SRP**) και τα code smells **Duplicate Code** και **Long Method** στην κλάση DataExtensionChange. Όταν μια μέθοδος περιέχει υπερβολικά μεγάλο αριθμό γραμμών κώδικα ή εκτελεί πολλές λειτουργίες, την καθιστά δύσκολη στην κατανόηση και συντήρηση.
4. Στην τέταρτη έκδοση εντοπίστηκαν τα code smells **Long Method** και **Complex Class** στην κλάση CalendarParser. Το code smell Complex Class εμφανίζεται όταν μια κλάση περιέχει υπερβολικά μεγάλη ή περίπλοκη λογική, γεγονός που την καθιστά δύσκολη στην κατανόηση, συντήρηση και επαναχρησιμοποίηση.
5. Στην πέμπτη και τελική έκδοση εντοπίστηκε στην κλάση Engine το code smell **Long Method** και η παραβίαση της αρχής **Single Responsibility Principle (SRP)**, καθώς συγκεντρωνει διαφορετικές ευθύνες, καθιστώντας την υπερβολικά μεγάλη και δύσκολη στην κατανόηση και στην συντήρηση.

2. Refactoring

First Refactor

Για την απομάκρυνση των Code Smells και των παραβιάσεων Αρχών Σχεδίασης εφαρμόστηκαν οι παρακάτω αναδομήσεις:

- Στην κλάση RangeFacet αναδομήθηκε η μέθοδος computeChoices με την μέθοδο **Extract Method** καθώς εντοπίστηκε το code smell Duplicate Code. Συγκεκριμένα έγινε εξαγωγή των κοινών κομματιών κώδικα σε μια νέα μέθοδο.

Αρχική:
Θεση: modules/core/src/main/java/com/google/refine/browsing/facets/RangeFacet.java
Κλαση: RangeFacet

```
  @Override
  public void computeChoices(Project project, FilteredRows filteredRows) {
      if (_eval != null && _errorMessage == null) {
          RowEvaluable rowEvaluable = getRowEvaluable(project);
          Column column = project.columnModel.getColumnByCellIndex(_cellIndex);
          String key = "numeric-bin:row-based:" + _config._expression;
          NumericBinIndex index = (NumericBinIndex) column.getPrecompute(key);
          if (index == null) {
              index = new NumericBinRowIndex(project, rowEvaluable);
              column.setPrecompute(key, index);
          }
          retrieveDataFromBaseBinIndex(index);
          ExpressionNumericValueBinner binner = new ExpressionNumericValueBinner(rowEvaluable,
index);
          filteredRows.accept(project, binner);
          retrieveDataFromBinner(binner);
      }
  }
  @Override
  public void computeChoices(Project project, FilteredRecords filteredRecords) {
      if (_eval != null && _errorMessage == null) {
          RowEvaluable rowEvaluable = getRowEvaluable(project);
          Column column = project.columnModel.getColumnByCellIndex(_cellIndex);
          String key = "numeric-bin:record-based:" + _config._expression;
          NumericBinIndex index = (NumericBinIndex) column.getPrecompute(key);
          if (index == null) {
              index = new NumericBinRecordIndex(project, rowEvaluable);
              column.setPrecompute(key, index);
          }
          retrieveDataFromBaseBinIndex(index);
          ExpressionNumericValueBinner binner = new ExpressionNumericValueBinner(rowEvaluable,
index);
          filteredRecords.accept(project, binner);
          retrieveDataFromBinner(binner);
      }
  }
```

Μετά την αναδόμηση **Extract Method**:

```
  @Override
   public void computeChoices(Project project, FilteredRows filteredRows) {
       computeChoicesCommon(project, filteredRows, true);
   }
   @Override
   public void computeChoices(Project project, FilteredRecords filteredRecords) {
       computeChoicesCommon(project, filteredRecords, false);
   }
   private void computeChoicesCommon(Project project, Object filter, boolean isRowBased) {
       if (_eval != null && _errorMessage == null) {
           RowEvaluable rowEvaluable = getRowEvaluable(project);
           Column column = project.columnModel.getColumnByCellIndex(_cellIndex);
           String key = "numeric-bin:row-based:" + _config._expression;
           NumericBinIndex index = (NumericBinIndex) column.getPrecompute(key);
           if (index == null) {
               if (isRowBased) {
                   index = new NumericBinRowIndex(project, rowEvaluable);
               } else {
                   index = new NumericBinRecordIndex(project, rowEvaluable);
               }
               column.setPrecompute(key, index);
           }
           retrieveDataFromBaseBinIndex(index);
           ExpressionNumericValueBinner binner = new ExpressionNumericValueBinner(rowEvaluable,
index);
           if (isRowBased) {
               ((FilteredRows) filter).accept(project, binner);
           } else {
               ((FilteredRecords) filter).accept(project, binner);
           }
           retrieveDataFromBinner(binner);
       }
   }
```

Εξάγαμε τα κοινά κομμάτια κώδικα σε μια νέα μέθοδο computeChoicesCommon. Χρησιμοποιούμε μια παράμετρο boolean την isRowBased για να κάνουμε τη μέθοδο πιο γενική ώστε να καλύψει και τις δύο περιπτώσεις. Ετσι εξασφαλίζουμε ότι η κοινή λογική γράφεται μόνο μία φορά

Second Refactor

Εφαρμόστηκε Refactoring σύμφωνα με τη μέθοδο **Extract Class** στην κλάση ColumnReorderChange, η οποία στοχεύει στην αναδιάρθρωση του κώδικα με την εξαγωγή σχετικών ευθυνών σε ξεχωριστές κλάσεις.

1. Δημιουργία Νέας Κλάσης ColumnReorderHandler: Η λογική για την αναδιάταξη των στηλών και τη διαχείριση των δεδομένων αποσπάστηκε από την κλάση ColumnReorderChange και μεταφέρθηκε στην κλάση ColumnReorderHandler.

2. Εφαρμογή Καθαρών Ευθυνών:
    a. Η ColumnReorderHandler είναι υπεύθυνη μόνο για την αναδιάταξη και τη διαχείριση των στηλών.
    b. Η ColumnReorderChange είναι υπεύθυνη για τον χειρισμό της αλλαγής στο ιστορικό του Project.
3. Αφαίρεση Διπλότυπου Κώδικα: Ο κώδικας για την εκκαθάριση των κελιών και την αντιγραφή δεδομένων οργανώθηκε σε διακριτές μεθόδους (backupOldCells, clearRemovedCells, updateProjectColumns).

Αρχική:

Θεση:
OpenRefine/modules/core/src/main/java/com/google/refine/model/changes/ColumnReorderChange,java
Κλάση: ColumnReorderChange

```
public class ColumnReorderChange extends ColumnChange {

   final protected List<String> _columnNames;
   protected List<Column> _oldColumns;
   protected List<Column> _newColumns;
   protected List<Column> _removedColumns;
   protected List<ColumnGroup> _oldColumnGroups;
   protected CellAtRowCellIndex[] _oldCells;

   public ColumnReorderChange(List<String> columnNames) {
      _columnNames = columnNames;
   }

  @Override
   public void apply(Project project) {
      synchronized (project) {
         if (_newColumns == null) {
                  …
         }
}
  @Override
   public void revert(Project project) {
      synchronized (project) {
         project.columnModel.columns.clear();
                  ….
         }
}

   @Override
   public void save(Writer writer, Properties options) throws IOException {
      writer.write("columnNameCount=");
          …
}
   static public Change load(LineNumberReader reader, Pool pool) throws Exception {
      List<String> columnNames = new ArrayList<String>();
          …
```

```
}
```

Μετά την αναδόμηση **Extract Class** :

```
package com.google.refine.model.changes;
        ….
   private final List<String> columnNames;
   private final ColumnReorderHandler handler;
   private final ColumnReorderSerializer serializer;

public class ColumnReorderChange extends ColumnChange {

   private final List<String> columnNames;
   private final ColumnReorderHandler handler;
   private final ColumnReorderSerializer serializer;

   public ColumnReorderChange(List<String> columnNames) {
      this.columnNames = columnNames;
      this.handler = new ColumnReorderHandler(columnNames);
      this.serializer = new ColumnReorderSerializer();
   }

   @Override
   public void apply(Project project) {
      handler.reorderColumns(project);
   }

   @Override
   public void revert(Project project) {
      handler.revertColumnOrder(project);
   }

   @Override
   public void save(Writer writer, Properties options) throws IOException {
      serializer.save(writer, handler);
   }

   public static Change load(LineNumberReader reader, Pool pool) throws Exception {
      return ColumnReorderSerializer.load(reader, pool);
   }
}
```

Δημιουργία νέας ξεχωριστης κλάσης ColumnReorderHandler

```
package com.google.refine.model.changes;
        ….

public class ColumnReorderHandler {

   private final List<String> columnNames;
   private List<Column> oldColumns;
   private List<Column> newColumns;
```

```
    private List<Column> removedColumns;
    private List<ColumnGroup> oldColumnGroups;
    private CellAtRowCellIndex[] oldCells;

    public ColumnReorderHandler(List<String> columnNames) {
        this.columnNames = columnNames;
    }

    public void reorderColumns(Project project) {
        synchronized (project) {
            if (newColumns == null) {
                initializeColumns(project);
            }

            if (removedColumns == null) {
                extractRemovedColumns(project);
            }

            if (oldCells == null) {
                backupOldCells(project);
            }

            clearRemovedCells(project);
            updateProjectColumns(project);
        }
    }

    public void revertColumnOrder(Project project) {
        synchronized (project) {
            project.columnModel.columns.clear();
            project.columnModel.columns.addAll(oldColumns);

            project.columnModel.columnGroups.clear();
            project.columnModel.columnGroups.addAll(oldColumnGroups);

            for (CellAtRowCellIndex cellInfo : oldCells) {
                Row row = project.rows.get(cellInfo.row);
                row.setCell(cellInfo.cellIndex, cellInfo.cell);
            }

            project.update();
        }
    }

    public void setOldColumns(List<Column> oldColumns) {
        this.oldColumns = oldColumns;
    }

    public void setNewColumns(List<Column> newColumns) {
        this.newColumns = newColumns;
    }
```

```
public void setRemovedColumns(List<Column> removedColumns) {
    this.removedColumns = removedColumns;
}

public void setOldColumnGroups(List<ColumnGroup> oldColumnGroups) {
    this.oldColumnGroups = oldColumnGroups;
}

public void setOldCells(CellAtRowCellIndex[] oldCells) {
    this.oldCells = oldCells;
}

public List<String> getColumnNames() {
    return columnNames;
}

public List<Column> getOldColumns() {
    return oldColumns;
}

public List<Column> getNewColumns() {
    return newColumns;
}

public List<Column> getRemovedColumns() {
    return removedColumns;
}

public CellAtRowCellIndex[] getOldCells() {
    return oldCells;
}

public List<ColumnGroup> getOldColumnGroups() {
    return oldColumnGroups;
}

private void initializeColumns(Project project) {
    newColumns = new ArrayList<>();
    oldColumns = new ArrayList<>(project.columnModel.columns);

    for (String name : columnNames) {
        Column column = project.columnModel.getColumnByName(name);
        if (column != null) {
            newColumns.add(column);
        }
    }

    oldColumnGroups = new ArrayList<>(project.columnModel.columnGroups);
}

private void extractRemovedColumns(Project project) {
    removedColumns = new ArrayList<>();
```

```
        for (String name : project.columnModel.getColumnNames()) {
            Column oldColumn = project.columnModel.getColumnByName(name);
            if (!newColumns.contains(oldColumn)) {
                removedColumns.add(oldColumn);
            }
        }
    }

    private void backupOldCells(Project project) {
        oldCells = new CellAtRowCellIndex[project.rows.size() * removedColumns.size()];

        int count = 0;
        for (int i = 0; i < project.rows.size(); i++) {
            for (int j = 0; j < removedColumns.size(); j++) {
                int cellIndex = removedColumns.get(j).getCellIndex();
                Row row = project.rows.get(i);

                Cell oldCell = null;
                if (cellIndex < row.cells.size()) {
                    oldCell = row.cells.get(cellIndex);
                }
                oldCells[count++] = new CellAtRowCellIndex(i, cellIndex, oldCell);
            }
        }
    }

    private void clearRemovedCells(Project project) {
        for (int i = 0; i < project.rows.size(); i++) {
            for (int j = 0; j < removedColumns.size(); j++) {
                int cellIndex = removedColumns.get(j).getCellIndex();
                Row row = project.rows.get(i);
                row.setCell(cellIndex, null);
            }
        }
    }

    private void updateProjectColumns(Project project) {
        project.columnModel.columns.clear();
        project.columnModel.columns.addAll(newColumns);
        project.columnModel.columnGroups.clear();
        project.update();
    }
}
```

Δημιουργία νέας ξεχωριστης κλάσης ColumnReorderSerializer

```
package com.google.refine.model.changes;
            …
public class ColumnReorderSerializer {

    public void save(Writer writer, ColumnReorderHandler handler) throws IOException {
        writer.write("columnNameCount=");
```

```
        writer.write(Integer.toString(handler.getColumnNames().size()));
        writer.write('\n');
        for (String columnName : handler.getColumnNames()) {
            writer.write(columnName);
            writer.write('\n');
        }

        writer.write("oldColumnCount=");
        writer.write(Integer.toString(handler.getOldColumns().size()));
        writer.write('\n');
        for (Column column : handler.getOldColumns()) {
            column.save(writer);
            writer.write('\n');
        }

        writer.write("newColumnCount=");
        writer.write(Integer.toString(handler.getNewColumns().size()));
        writer.write('\n');
        for (Column column : handler.getNewColumns()) {
            column.save(writer);
            writer.write('\n');
        }

        writer.write("removedColumnCount=");
        writer.write(Integer.toString(handler.getRemovedColumns().size()));
        writer.write('\n');
        for (Column column : handler.getRemovedColumns()) {
            column.save(writer);
            writer.write('\n');
        }
    }

    public static Change load(LineNumberReader reader, Pool pool) throws Exception {
        List<String> columnNames = new ArrayList<>();
        List<Column> oldColumns = new ArrayList<>();
        List<Column> newColumns = new ArrayList<>();
        List<Column> removedColumns = new ArrayList<>();
        CellAtRowCellIndex[] oldCells = new CellAtRowCellIndex[0];
        List<ColumnGroup> oldColumnGroups = new ArrayList<>();

        String line;
        while ((line = reader.readLine()) != null && !"/ec/".equals(line)) {
            int equal = line.indexOf('=');
            String field = line.substring(0, equal);

            if ("columnNameCount".equals(field)) {
                int count = Integer.parseInt(line.substring(equal + 1));
                for (int i = 0; i < count; i++) {
                    columnNames.add(reader.readLine());
                }
            } else if ("oldColumnCount".equals(field)) {
                int count = Integer.parseInt(line.substring(equal + 1));
```

```
                for (int i = 0; i < count; i++) {
                    oldColumns.add(Column.load(reader.readLine()));
                }
            } else if ("newColumnCount".equals(field)) {
                int count = Integer.parseInt(line.substring(equal + 1));
                for (int i = 0; i < count; i++) {
                    newColumns.add(Column.load(reader.readLine()));
                }
            } else if ("removedColumnCount".equals(field)) {
                int count = Integer.parseInt(line.substring(equal + 1));
                for (int i = 0; i < count; i++) {
                    removedColumns.add(Column.load(reader.readLine()));
                }
            }
        }

        ColumnReorderHandler handler = new ColumnReorderHandler(columnNames);
        handler.setOldColumns(oldColumns);
        handler.setNewColumns(newColumns);
        handler.setRemovedColumns(removedColumns);
        handler.setOldCells(oldCells);

        return new ColumnReorderChange(columnNames);
    }
}
```

Third Refactor

Έγινε αναδόμηση της κλάσης apply η οποία ανήκει στην DataExtensionChange.Συγκεκριμένα με την εφαρμογή της **Extract Method** έγινε ο Διαχωρισμός των Ευθυνών (SRP). Η μέθοδος apply τώρα εκτελεί μόνο τον συντονισμό της διαδικασίας. Η λογική έχει χωριστεί σε μικρότερες μεθόδους με συγκεκριμένη ευθύνη:

1. allocateNewCellIndices: Κατανέμει νέα indices για τις στήλες.
2. initializeNewRows: Δημιουργεί τις δομές δεδομένων για τις γραμμές.
3. processRows: Διαχειρίζεται την επεξεργασία των γραμμών.
4. processRowWithDataExtension: Επεξεργάζεται μεμονωμένες γραμμές με data extensions.
5. replaceOldRows: Αντικαθιστά τις παλιές γραμμές με τις νέες.
6. createNewColumns: Δημιουργεί και εισάγει τις νέες στήλες.

Αρχική θέση: modules/core/src/main/java/com/google/refine/model/changes/DataExtensionChange.java
Κλαση: DataExtensionChange

```
@Override
public void apply(Project project) {
  synchronized (project) {
    if (_firstNewCellIndex < 0) {
      _firstNewCellIndex = project.columnModel.allocateNewCellIndex();
```

```
        for (int i = 1; i < _columnNames.size(); i++) {
            project.columnModel.allocateNewCellIndex();
        }
        _oldRows = new ArrayList<Row>(project.rows);
        _newRows = new ArrayList<Row>(project.rows.size());
        int cellIndex = project.columnModel.getColumnByName(_baseColumnName).getCellIndex();
        int keyCellIndex =
project.columnModel.columns.get(project.columnModel.getKeyColumnIndex()).getCellIndex();
        int index = 0;
        int rowIndex = index < _rowIndices.size() ? _rowIndices.get(index) : _oldRows.size();
        DataExtension dataExtension = index < _rowIndices.size() ? _dataExtensions.get(index) : null;
        index++;
        Map<String, Recon> reconMap = new HashMap<String, Recon>();
        for (int r = 0; r < _oldRows.size(); r++) {
            Row oldRow = _oldRows.get(r);
            if (r < rowIndex) {
                _newRows.add(oldRow.dup());
                continue;
            }
            if (dataExtension == null || dataExtension.data.length == 0) {
                _newRows.add(oldRow);
            } else {
                Row firstNewRow = oldRow.dup();
                extendRow(firstNewRow, dataExtension, 0, reconMap);
                _newRows.add(firstNewRow);
                int r2 = r + 1;
                for (int subR = 1; subR < dataExtension.data.length; subR++) {
                    if (r2 < project.rows.size()) {
                        Row oldRow2 = project.rows.get(r2);
                        if (oldRow2.isCellBlank(cellIndex) &&
                                oldRow2.isCellBlank(keyCellIndex)) {
                            Row newRow = oldRow2.dup();
                            extendRow(newRow, dataExtension, subR, reconMap);
                            _newRows.add(newRow);
                            r2++;
                            continue;
                        }
                    }
                    Row newRow = new Row(cellIndex + _columnNames.size());
                    extendRow(newRow, dataExtension, subR, reconMap);
                    _newRows.add(newRow);
                }
                r = r2 - 1; // r will be incremented by the for loop anyway
            }
            rowIndex = index < _rowIndices.size() ? _rowIndices.get(index) : _oldRows.size();
            dataExtension = index < _rowIndices.size() ? _dataExtensions.get(index) : null;
            index++;
        }
    }
    project.rows.clear();
    project.rows.addAll(_newRows);
    for (int i = 0; i < _columnNames.size(); i++) {
```

```
            String name = _columnNames.get(i);
            int cellIndex = _firstNewCellIndex + i;
            Column column = new Column(cellIndex, name);
            ReconType columnType = _columnTypes.get(i);
            column.setReconConfig(new DataExtensionReconConfig(
                    _service,
                    _identifierSpace,
                    _schemaSpace,
                    columnType));
            if (project.columnModel.getColumnByName(_baseColumnName) != null) {

column.setSourceReconConfig(project.columnModel.getColumnByName(_baseColumnName).getRe
conConfig());
            }
            ReconStats reconStats = ReconStats.create(project, cellIndex);
            if (reconStats.matchedTopics > 0) {
                column.setReconStats(reconStats);
            }
            try {
                project.columnModel.addColumn(_columnInsertIndex + i, column, true);
                // the column might have been renamed to avoid collision
                _columnNames.set(i, column.getName());
            } catch (ModelException e) {
                // won't get here since we set the avoid collision flag
            }
        }
        project.update();
    }
}
```

Μετά την αναδόμηση **Extract Method**:

```
 @Override
    public void apply(Project project) {
        synchronized (project) {
            if (_firstNewCellIndex < 0) {
                allocateNewCellIndices(project);
                initializeNewRows(project);
                processRows(project);
                replaceOldRows(project);
                createNewColumns(project);
            }
            project.update();
        }
    }
    private void allocateNewCellIndices(Project project) {
        _firstNewCellIndex = project.columnModel.allocateNewCellIndex();
        for (int i = 1; i < _columnNames.size(); i++) {
            project.columnModel.allocateNewCellIndex();
        }
    }
```

```
  private void initializeNewRows(Project project) {
    _oldRows = new ArrayList<>(project.rows);
    _newRows = new ArrayList<>(_oldRows.size());
  }
  private void processRows(Project project) {
    Map<String, Recon> reconMap = new HashMap<>();
    int cellIndex = project.columnModel.getColumnByName(_baseColumnName).getCellIndex();
    int keyCellIndex =
project.columnModel.columns.get(project.columnModel.getKeyColumnIndex()).getCellIndex();
    int index = 0;
    int rowIndex = index < _rowIndices.size() ? _rowIndices.get(index) : _oldRows.size();
    DataExtension dataExtension = index < _rowIndices.size() ? _dataExtensions.get(index) : null;
    index++;
    for (int r = 0; r < _oldRows.size(); r++) {
      Row oldRow = _oldRows.get(r);
      if (r < rowIndex) {
        _newRows.add(oldRow.dup());
      } else {
        processRowWithDataExtension(oldRow, dataExtension, reconMap, cellIndex, keyCellIndex);
        rowIndex = index < _rowIndices.size() ? _rowIndices.get(index) : _oldRows.size();
        dataExtension = index < _rowIndices.size() ? _dataExtensions.get(index) : null;
        index++;
      }
    }
  }
  private void processRowWithDataExtension(Row oldRow, DataExtension dataExtension,
Map<String, Recon> reconMap, int cellIndex, int keyCellIndex) {
    if (dataExtension == null || dataExtension.data.length == 0) {
      _newRows.add(oldRow);
    } else {
      Row firstNewRow = oldRow.dup();
      extendRow(firstNewRow, dataExtension, 0, reconMap);
      _newRows.add(firstNewRow);
      int r2 = _newRows.size();
      for (int subR = 1; subR < dataExtension.data.length; subR++) {
        Row newRow = createOrExtendRow(dataExtension, reconMap, subR, r2, cellIndex,
keyCellIndex);
        _newRows.add(newRow);
        r2++;
      }
    }
  }
  private Row createOrExtendRow(DataExtension dataExtension, Map<String, Recon> reconMap, int
subR, int r2, int cellIndex, int keyCellIndex) {
    if (r2 < _oldRows.size()) {
      Row oldRow2 = _oldRows.get(r2);
      if (oldRow2.isCellBlank(cellIndex) && oldRow2.isCellBlank(keyCellIndex)) {
        Row newRow = oldRow2.dup();
        extendRow(newRow, dataExtension, subR, reconMap);
        return newRow;
      }
    }
```

```
        Row newRow = new Row(cellIndex + _columnNames.size());
        extendRow(newRow, dataExtension, subR, reconMap);
        return newRow;
    }
    private void replaceOldRows(Project project) {
        project.rows.clear();
        project.rows.addAll(_newRows);
    }
    private void createNewColumns(Project project) {
        for (int i = 0; i < _columnNames.size(); i++) {
            String name = _columnNames.get(i);
            int cellIndex = _firstNewCellIndex + i;
            Column column = new Column(cellIndex, name);
            ReconType columnType = _columnTypes.get(i);
            column.setReconConfig(new DataExtensionReconConfig(_service, _identifierSpace,
_schemaSpace, columnType));
            if (project.columnModel.getColumnByName(_baseColumnName) != null) {

column.setSourceReconConfig(project.columnModel.getColumnByName(_baseColumnName).getRe
conConfig());
            }
            ReconStats reconStats = ReconStats.create(project, cellIndex);
            if (reconStats.matchedTopics > 0) {
                column.setReconStats(reconStats);
            }
            try {
                project.columnModel.addColumn(_columnInsertIndex + i, column, true);
                _columnNames.set(i, column.getName());
            } catch (ModelException e) {
                // won't get here since we set the avoid collision flag
            }
        }
    }
```

Fourth Refactor

Η αρχική μέθοδος parseTime της κλάσης CalendarParser παραβιαζε την Single Responsibility Principle - SRP. Η παραβίαση διορθώθηκε με την εφαρμογή της **Extract Method** κατανέμοντας τη λογική σε μικρότερες, ορισμένες μεθόδους, που καθεμία έχει μία μόνο ευθύνη. Συγκεκριμένα ο διαχωρισμός των ευθυνών σε μικρότερες μεθόδους:

- **extractTimeAndSetMeridian**: Εστιάζει μόνο στον εντοπισμό AM/PM και την εξαγωγή του υπολοίπου της ώρας.
- **parseTimeComponents**: Επεξεργάζεται τα επιμέρους στοιχεία της ώρας (ώρες, λεπτά, δευτερόλεπτα, χιλιοστά).
- **parseTokenAsInteger**: Αποκλειστικά υπεύθυνη για τη μετατροπή μιας συμβολοσειράς σε αριθμητική τιμή, διατηρώντας τον έλεγχο σφαλμάτων σε ένα σημείο.
- **updateStateWithParsedValue**: Ενημερώνει την κατάσταση (ParserState) με βάση την τρέχουσα τιμή και θέση.
- **getNextPlace**: Επιστρέφει την επόμενη θέση επεξεργασίας (π.χ., από ώρα -> λεπτό).

Αρχική:

Θέση: modules/core/src/main/java/com/google/refine/expr/util/CalendarParser.java
Κλαση: CalendarParser

```
private static final void parseTime(String dateStr, ParserState state,
        String timeStr) throws CalendarParserException {
    int place = PLACE_HOUR;
    String tmpTime;
    final char lastChar = timeStr.charAt(timeStr.length() - 1);
    if (lastChar != 'm' && lastChar != 'M') {
        if (DEBUG) {
            System.err.println("No AM/PM in \"" + timeStr + "\" (time)");
        }
        tmpTime = timeStr;
    } else {
        final char preLast = timeStr.charAt(timeStr.length() - 2);
        if (preLast == 'a' || preLast == 'A') {
            state.setTimePostMeridian(false);
        } else if (preLast == 'p' || preLast == 'P') {
            state.setTimePostMeridian(true);
        } else {
            throw new CalendarParserException("Bad time \"" + timeStr
                    + "\" in date \"" + dateStr + "\"");
        }
        tmpTime = timeStr.substring(0, timeStr.length() - 2);
        if (DEBUG) {
            System.err.println("Found "
                    + (state.isTimePostMeridian() ? "PM" : "AM")
                    + ". now \"" + tmpTime + "\" (time)");
        }
    }
    String[] tList = tmpTime.split("[:\\.]");
    for (String token : tList) {
        if (DEBUG) {
            System.err.println("HOUR "
                    + (state.isHourSet() ? Integer
                            .toString(state.getHour()) : "UNSET")
                    + ", MINUTE "
                    + (state.isMinuteSet() ? Integer.toString(state
                            .getMinute()) : "UNSET")
                    + ", SECOND "
                    + (state.isSecondSet() ? Integer.toString(state
                            .getSecond()) : "UNSET")
                    + ", MILLISECOND "
                    + (state.isMillisecondSet() ? Integer.toString(state
                            .getMillisecond()) : "UNSET")
                    + ", TOKEN=\""
                    + token + "\"");
        }
        final int val;
        try {
            val = Integer.parseInt(token);
        } catch (NumberFormatException nfe) {
```

```
    throw new CalendarParserException("Bad "
            + getTimePlaceString(place) + " string \"" + token
            + "\" in \"" + dateStr + "\"");
}
switch (place) {
    case PLACE_HOUR:
        try {
            state.setHour(val);
        } catch (CalendarParserException dfe) {
            throw new CalendarParserException(dfe.getMessage()
                    + " in \"" + dateStr + "\"");
        }
        if (DEBUG) {
            System.err.println("Set hour to " + val);
        }
        place = PLACE_MINUTE;
        break;
    case PLACE_MINUTE:
        try {
            state.setMinute(val);
        } catch (CalendarParserException dfe) {
            throw new CalendarParserException(dfe.getMessage()
                    + " in \"" + dateStr + "\"");
        }
        if (DEBUG) {
            System.err.println("Set minute to " + val);
        }
        place = PLACE_SECOND;
        break;
    case PLACE_SECOND:
        try {
            state.setSecond(val);
        } catch (CalendarParserException dfe) {
            throw new CalendarParserException(dfe.getMessage()
                    + " in \"" + dateStr + "\"");
        }
        if (DEBUG) {
            System.err.println("Set second to " + val);
        }
        place = PLACE_MILLI;
        break;
    case PLACE_MILLI:
        try {
            state.setMillisecond(val);
        } catch (CalendarParserException dfe) {
            throw new CalendarParserException(dfe.getMessage()
                    + " in \"" + dateStr + "\"");
        }
        if (DEBUG) {
            System.err.println("Set millisecond to " + val);
        }
        place = PLACE_UNKNOWN;
```

```
                break;
            default:
                throw new CalendarParserException("Unexpected place value "
                        + place);
        }
    }
}
```

Μετά την αναδόμηση **Extract Method**

```
private static final void parseTime(String dateStr, ParserState state, String timeStr) throws
CalendarParserException {
    String tmpTime = extractTimeAndSetMeridian(dateStr, state, timeStr);
    parseTimeComponents(dateStr, state, tmpTime);
}

private static String extractTimeAndSetMeridian(String dateStr, ParserState state, String timeStr)
throws CalendarParserException {
  final char lastChar = timeStr.charAt(timeStr.length() - 1);
  if (lastChar != 'm' && lastChar != 'M') {
      if (DEBUG) {
          System.err.println("No AM/PM in \"" + timeStr + "\" (time)");
      }
      return timeStr;
  }
  final char preLast = timeStr.charAt(timeStr.length() - 2);
  if (preLast == 'a' || preLast == 'A') {
      state.setTimePostMeridian(false);
  } else if (preLast == 'p' || preLast == 'P') {
      state.setTimePostMeridian(true);
  } else {
      throw new CalendarParserException("Bad time \"" + timeStr + "\" in date \"" + dateStr + "\"");
  }
  if (DEBUG) {
      System.err.println("Found " + (state.isTimePostMeridian() ? "PM" : "AM") + ". now \"" +
timeStr.substring(0, timeStr.length() - 2) + "\" (time)");
  }
  return timeStr.substring(0, timeStr.length() - 2);
}
private static void parseTimeComponents(String dateStr, ParserState state, String timeStr) throws
CalendarParserException {
  String[] tList = timeStr.split("[:\\.]");
  int place = PLACE_HOUR;
  for (String token : tList) {
      if (DEBUG) {
          System.err.println("HOUR " + (state.isHourSet() ? state.getHour() : "UNSET") +
                  ", MINUTE " + (state.isMinuteSet() ? state.getMinute() : "UNSET") +
                  ", SECOND " + (state.isSecondSet() ? state.getSecond() : "UNSET") +
                  ", MILLISECOND " + (state.isMillisecondSet() ? state.getMillisecond() : "UNSET") +
                  ", TOKEN=\"" + token + "\"");
      }
      int val = parseTokenAsInteger(dateStr, token, place);
```

```
      updateStateWithParsedValue(dateStr, state, val, place);
      place = getNextPlace(place);
    }
  }
  private static int parseTokenAsInteger(String dateStr, String token, int place) throws
  CalendarParserException {
    try {
      return Integer.parseInt(token);
    } catch (NumberFormatException nfe) {
      throw new CalendarParserException("Bad " + getTimePlaceString(place) + " string \"" + token +
  "\" in \"" + dateStr + "\"");
    }
  }
  private static void updateStateWithParsedValue(String dateStr, ParserState state, int val, int place)
  throws CalendarParserException {
    try {
      switch (place) {
        case PLACE_HOUR:
          state.setHour(val);
          if (DEBUG) System.err.println("Set hour to " + val);
          break;
        case PLACE_MINUTE:
          state.setMinute(val);
          if (DEBUG) System.err.println("Set minute to " + val);
          break;
        case PLACE_SECOND:
          state.setSecond(val);
          if (DEBUG) System.err.println("Set second to " + val);
          break;
        case PLACE_MILLI:
          state.setMillisecond(val);
          if (DEBUG) System.err.println("Set millisecond to " + val);
          break;
        default:
          throw new CalendarParserException("Unexpected place value " + place);
      }
    } catch (CalendarParserException dfe) {
      throw new CalendarParserException(dfe.getMessage() + " in \"" + dateStr + "\"");
    }
  }
  private static int getNextPlace(int currentPlace) {
    switch (currentPlace) {
      case PLACE_HOUR: return PLACE_MINUTE;
      case PLACE_MINUTE: return PLACE_SECOND;
      case PLACE_SECOND: return PLACE_MILLI;
      default: return PLACE_UNKNOWN;
    }
```

Fifth Refactor

Η αρχική μέθοδος computeFacets της κλάσης Engine παραβίαζε την **Single Responsibility Principle (SRP)**, καθώς συνδύαζε την επεξεργασία "Row-Based" και "Record-Based" λειτουργιών σε μία μέθοδο. Η παραβίαση διορθώθηκε με την εφαρμογή της **Extract Method**, κατανέμοντας τη λογική σε μικρότερες, εξειδικευμένες μεθόδους, καθεμία με μία μόνο ευθύνη. Η λογική χωρίστηκε σε μικρότερες, εξειδικευμένες μεθόδους, καθεμία με μία μόνο ευθύνη:

- **computeRowBasedFacets**: Εστιάζει αποκλειστικά στον υπολογισμό των facets στη λειτουργία "Row-Based".
- **computeRecordBasedFacets**: Διαχειρίζεται τον υπολογισμό των facets στη λειτουργία "Record-Based".

Επιπλέον, οι μέθοδοι **getFilteredRows** και **getFilteredRecords** αναδομήθηκαν για να διατηρήσουν την αρχή SRP, διαχωρίζοντας τη λογική δημιουργίας φίλτρων σε νέες μεθόδους:

- **createRowFilters**: Διαχειρίζεται τη δημιουργία φίλτρων σε λειτουργία "Row-Based".
- **createRecordFilters**: Διαχειρίζεται τη δημιουργία φίλτρων σε λειτουργία "Record-Based".

Αρχική:
Θέση: modules/core/src/main/java/com/google/refine/browsing/Engine.java
Κλάση: Engine

```
package com.google.refine.browsing;
        ….
public class Engine {
  static public enum Mode {
     @JsonProperty("row-based")
     RowBased, @JsonProperty("record-based")
     RecordBased
  }
  public final static String INCLUDE_DEPENDENT = "includeDependent";
  public final static String MODE = "mode";
  public final static String MODE_ROW_BASED = "row-based";
  public final static String MODE_RECORD_BASED = "record-based";
  @JsonIgnore
  protected Project _project;
  @JsonProperty("facets")
  protected List<Facet> _facets = new LinkedList<Facet>();
  @JsonIgnore
  protected EngineConfig _config = new EngineConfig(Collections.emptyList(), Mode.RowBased);
  static public String modeToString(Mode mode) {
     return mode == Mode.RowBased ? MODE_ROW_BASED : MODE_RECORD_BASED;
  }
  static public Mode stringToMode(String s) {
     return MODE_ROW_BASED.equals(s) ? Mode.RowBased : Mode.RecordBased;
```

```
}
public Engine(Project project) {
    _project = project;
}
@JsonProperty("engine-mode")
public Mode getMode() {
    return _config.getMode();
}
public void setMode(Mode mode) {
    _config = new EngineConfig(_config.getFacetConfigs(), mode);
}
@JsonIgnore
public FilteredRows getAllRows() {
    return new FilteredRows() {
        @Override
        public void accept(Project project, RowVisitor visitor) {
            try {
                visitor.start(project);
                int c = project.rows.size();
                for (int rowIndex = 0; rowIndex < c; rowIndex++) {
                    Row row = project.rows.get(rowIndex);
                    if (visitor.visit(project, rowIndex, rowIndex, row)) {
                        break;
                    }
                }
            } finally {
                visitor.end(project);
            }
        }
    };
}
@JsonIgnore
public FilteredRows getAllFilteredRows() {
    return getFilteredRows(null);
}
public FilteredRows getFilteredRows(Facet except) {
    if (_config.getMode().equals(Mode.RecordBased)) {
        return new FilteredRecordsAsFilteredRows(getFilteredRecords(except));
    } else if (_config.getMode().equals(Mode.RowBased)) {
        ConjunctiveFilteredRows cfr = new ConjunctiveFilteredRows();
        for (Facet facet : _facets) {
            if (facet != except) {
                RowFilter rowFilter = facet.getRowFilter(_project);
                if (rowFilter != null) {
                    cfr.add(rowFilter);
                }
            }
        }
        return cfr;
    }
    throw new InternalError("Unknown mode.");
}
```

```
  @JsonIgnore
  public FilteredRecords getAllRecords() {
    return new FilteredRecords() {
      @Override
      public void accept(Project project, RecordVisitor visitor) {
        try {
          visitor.start(project);
          int c = project.recordModel.getRecordCount();
          for (int r = 0; r < c; r++) {
            Record record = project.recordModel.getRecord(r);
            visitor.visit(project, record.fromRowIndex, record);
          }
        } finally {
          visitor.end(project);
        }
      }
    };
  }
  @JsonIgnore
  public FilteredRecords getFilteredRecords() {
    return getFilteredRecords(null);
  }
  public FilteredRecords getFilteredRecords(Facet except) {
    if (_config.getMode().equals(Mode.RecordBased)) {
      ConjunctiveFilteredRecords cfr = new ConjunctiveFilteredRecords();
      for (Facet facet : _facets) {
        if (facet != except) {
          RecordFilter recordFilter = facet.getRecordFilter(_project);
          if (recordFilter != null) {
            cfr.add(recordFilter);
          }
        }
      }
      return cfr;
    }
    throw new InternalError("This method should not be called when the engine is not in record
mode.");
  }
  public void initializeFromConfig(EngineConfig config) {
    _config = config;
    _facets = config.getFacetConfigs().stream()
        .map(c -> c.apply(_project))
        .collect(Collectors.toList());
  }
  public void computeFacets() {
    if (_config.getMode().equals(Mode.RowBased)) {
      for (Facet facet : _facets) {
        FilteredRows filteredRows = getFilteredRows(facet);
        facet.computeChoices(_project, filteredRows);
      }
    } else if (_config.getMode().equals(Mode.RecordBased)) {
      for (Facet facet : _facets) {
```

```
            FilteredRecords filteredRecords = getFilteredRecords(facet);
            facet.computeChoices(_project, filteredRecords);
        }
      } else {
          throw new InternalError("Unknown mode.");
      }
  }
}
```

Μετά την αναδόμηση Extract Method

```
package com.google.refine.browsing;
            …
public class Engine {
            …

   public void computeFacets() {
      if (_config.getMode().equals(Mode.RowBased)) {
          computeRowBasedFacets();
      } else if (_config.getMode().equals(Mode.RecordBased)) {
          computeRecordBasedFacets();
      } else {
          throw new InternalError("Unknown mode.");
      }
   }
   private void computeRowBasedFacets() {
      for (Facet facet : _facets) {
          FilteredRows filteredRows = getFilteredRows(facet);
          facet.computeChoices(_project, filteredRows);
      }
   }
   private void computeRecordBasedFacets() {
      for (Facet facet : _facets) {
          FilteredRecords filteredRecords = getFilteredRecords(facet);
          facet.computeChoices(_project, filteredRecords);
      }
   }

…
  public FilteredRows getFilteredRows(Facet except) {
      if (_config.getMode().equals(Mode.RecordBased)) {
          return new FilteredRecordsAsFilteredRows(getFilteredRecords(except));
      } else if (_config.getMode().equals(Mode.RowBased)) {
          return createRowFilters(except);
      }
      throw new InternalError("Unknown mode.");
   }
   private FilteredRows createRowFilters(Facet except) {
      ConjunctiveFilteredRows cfr = new ConjunctiveFilteredRows();
      for (Facet facet : _facets) {
          if (facet != except) {
              RowFilter rowFilter = facet.getRowFilter(_project);
```

```
            if (rowFilter != null) {
                cfr.add(rowFilter);
            }
        }
    }
    return cfr;
  }
…
  public FilteredRecords getFilteredRecords(Facet except) {
    if (_config.getMode().equals(Mode.RecordBased)) {
      return createRecordFilters(except);
    }
    throw new InternalError("This method should not be called when the engine is not in record
mode.");
  }
  private FilteredRecords createRecordFilters(Facet except) {
    ConjunctiveFilteredRecords cfr = new ConjunctiveFilteredRecords();
    for (Facet facet : _facets) {
      if (facet != except) {
        RecordFilter recordFilter = facet.getRecordFilter(_project);
        if (recordFilter != null) {
          cfr.add(recordFilter);
        }
      }
    }
    return cfr;
  }
```

3. Link github με τις 5 αναδομήσεις, 5 revisions

Στον παρακάτω σύνδεσμο του GitHub περιλαμβάνονται οι 5 αναδομήσεις (restructurings) και οι 5 αναθεωρήσεις (revisions) του έργου:

https://github.com/ihopewecanchangeitlater/System-Software-Technology

4. Επιλογή Μετρικών και Ανάλυση Αποτελεσμάτων

Για τον υπολογισμό των μετρικών χρησιμοποιήθηκε το εργαλείο “Metric Calculator με GUI”.

- **Μετρικές που επιλέχθηκαν**:
    a. SIZE1 : Lines of Code (LoC). Μετρά τον αριθμό των γραμμών κώδικα. Σχόλια και κενές γραμμές δεν υπολογίζονται
    b. WMC* : Weighted methods per class. Υπολογίζει την πολυπλοκότητα μιας κλάσης ως το άθροισμα της πολυπλοκότητας των μεθόδων της. Η πολυπλοκότητα κάθε μεθόδου μετράται με τον **κυκλικό βαθμό πολυπλοκότητας (Cyclomatic Complexity - CC)**.

- **Ανάλυση Μεταβολών**:
    a. Το refactoring στην κλάση **RangeFacet** μείωσε τη μετρική **SIZE1 (Lines of Code)** από 110089 σε 110077, καθώς τα κοινά κομμάτια κώδικα στις μεθόδους συγκεντρώθηκαν στη νέα βοηθητική μέθοδο **computeChoicesCommon**. Αυτή η μέθοδος χρησιμοποιεί μια παράμετρο boolean (**isRowBased**) για να καλύψει διαφορετικά σενάρια, εξασφαλίζοντας ότι η κοινή λογική γράφεται μόνο μία φορά. Ως αποτέλεσμα, αφαιρέθηκαν επαναλαμβανόμενες γραμμές κώδικα, μειώνοντας το συνολικό μέγεθος του κώδικα, ενώ η δομή έγινε πιο καθαρή και συντηρήσιμη.
    b. Το refactoring στην κλάση **ColumnReorderChange** μέσω της μεθόδου **Extract Class** μείωσε τη μετρική **WMC*** (Weighted Methods per Class), καθώς απέσπασε τη λογική αναδιάταξης στηλών σε μια νέα κλάση, την **ColumnReorderHandler**. Αυτή η αναδιοργάνωση περιόρισε την πολυπλοκότητα της αρχικής κλάσης, κάνοντας την πιο εστιασμένη και ευανάγνωστη, σύμφωνα με την Αρχή Μοναδικής Αρμοδιότητας (SRP). Ως αποτέλεσμα, οι μέθοδοι είναι πλέον πιο απλές και ευθύγραμμες, μειώνοντας τον συνολικό κυκλικό βαθμό πολυπλοκότητας.
    c. Το refactoring της μεθόδου **apply** στην κλάση **DataExtensionChange** μείωσε τη μετρική **SIZE1 (Lines of Code)** από 110084 σε 110069, καθώς η μέθοδος διαχωρίστηκε σε μικρότερες, εξειδικευμένες μεθόδους σύμφωνα με την Αρχή Μοναδικής Αρμοδιότητας (SRP). Οι νέες μέθοδοι (**allocateNewCellIndices**, **initializeNewRows**, **processRows**, **processRowWithDataExtension**, **replaceOldRows**, **createNewColumns**) ανέλαβαν συγκεκριμένες ευθύνες, μειώνοντας τον όγκο κώδικα στην κύρια μέθοδο. Αυτό οδήγησε σε καθαρότερη δομή και ευκολότερη συντήρηση, ενώ παράλληλα βελτιώθηκε η αναγνωσιμότητα και η επαναχρησιμοποίηση του κώδικα.

d. Η μέθοδος **parseTime** της κλάσης **CalendarParser** αναδομήθηκε μέσω της τεχνικής **Extract Method**, μειώνοντας τη μετρική **SIZE1 (Lines of Code)** από 110069 σε 110037. Η λογική διαχωρίστηκε σε μικρότερες, εξειδικευμένες μεθόδους, όπως **extractTimeAndSetMeridian**, **parseTimeComponents**, **parseTokenAsInteger**, **updateStateWithParsedValue**, και **getNextPlace**, καθεμία με μοναδική ευθύνη. Αυτός ο διαχωρισμός εναρμονίστηκε με την Αρχή Μοναδικής Αρμοδιότητας (SRP), μειώνοντας τον αριθμό των γραμμών κώδικα στην κύρια μέθοδο, ενώ βελτίωσε τη σαφήνεια, τη συντηρησιμότητα και την επαναχρησιμοποίηση του κώδικα.
e. Η μέθοδος **computeFacets** της κλάσης **Engine** αναδομήθηκε με τη χρήση της τεχνικής **Extract Method**, μειώνοντας τη μετρική **SIZE1 (Lines of Code)** από 110037 σε 110034. Η λογική διαχωρίστηκε σε εξειδικευμένες μεθόδους, όπως **computeRowBasedFacets** και **computeRecordBasedFacets**, καθεμία υπεύθυνη για τον υπολογισμό facets σε "Row-Based" και "Record-Based" λειτουργίες αντίστοιχα. Επιπλέον, οι μέθοδοι **getFilteredRows** και **getFilteredRecords** αναδομήθηκαν ώστε να διαχωρίσουν τη δημιουργία φίλτρων σε νέες μεθόδους (**createRowFilters**, **createRecordFilters**). Το αποτέλεσμα είναι καθαρότερος κώδικας, καλύτερη τήρηση της **Αρχής Μοναδικής Αρμοδιότητας (SRP)**, και μικρότερος αριθμός γραμμών στην κύρια μέθοδο, βελτιώνοντας τη συντηρησιμότητα και την αναγνωσιμότητα.

Πίνακας Μετρικών

Metric	Stat	Start	1	2	3	4	5
SIZE1	SUM	110089	110077	110084	110069	110037	110033
	AVG	80,1813547	80,17261471	80,06109091	80,05018182	80,02690909	80,024
	MAX	5251	5251	5251	5251	5251	5251
WMC*	SUM	1156,292083	1156,292083	1154,042083	1154,042083	1154,042083	1154,042083
	AVG	1,484328733	1,484328733	1,479541132	1,479541132	1,479541132	1,479541132
	MAX	21,5	21,5	21,5	21,5	21,5	21,5