

Report of Project - GanttProject

Bárbara Correia 59837 – p.5

Gonçalo Rodrigues 60044 – p.2

Guilherme Santana 60182 - p.1

Joana Cruz 60264 – p.2

Gonçalo Gomes 60774 – p.2

Git: https://github.com/barbara-correia/ES_projeto_1

Índice

A. Introduction to our Git Repository	5
1. Relevant branches	5
2. Issues with Git and Gantt Project	6
3. Who coded what	7
4. What were the plans and what were the results	7
B. Code Smells	9
1. Bárbara Correia's Code Smell 1	9
2. Bárbara Correia's Code Smell 2	10
3. Bárbara Correia's Code Smell 3	11
4. Gonçalo Rodrigues's Code Smell 1	12
5. Gonçalo Rodrigues's Code Smell 2	14
6. Gonçalo Rodrigues's Code Smell 3	15
7. Guilherme Santana's Code Smell 1	16
8. Guilherme Santana's Code Smell 2	17
9. Guilherme Santana's Code Smell 3	18
10. Joana Cruz's Code Smells 1	20
11. Joana Cruz's Code Smell 2	24
12. Joana Cruz's Code Smell 3	26
13. Gonçalo Gomes's Code Smell 1	28
14. Gonçalo Gomes's Code Smell 2	29
15. Gonçalo Gomes's Code Smell 3	30
C. Design Patterns	33
1. Bárbara Correia Design Pattern 1	33
2. Bárbara Correia's Design Pattern 2	34
3. Bárbara Correia's Design Pattern 3	35
4. Gonçalo Rodrigues's Design Pattern 1	37
5. Gonçalo Rodrigues's Design Pattern 2	38
6. Gonçalo Rodrigues's Design Pattern 3	40
7. Guilherme Santana's Design Pattern	41
8. Guilherme Santana's Design Pattern 2	42
9. Guilherme Santana's Design Pattern 3	43

10. Joana Cruz Design Pattern 1	46
11. Joana Cruz's Design Pattern 2	47
12. Joana Cruz's Design Pattern 3	49
13. Gonalo Gomes's Design Pattern 1	51
14. Gonalo Gomes's Design Pattern 2	53
15. Gonalo Gomes's Design Pattern 3	54
D. Use Case Diagrams	56
1. Brbara Correia's Use Case	56
2. Gonalo Rodrigues's Use Cases	58
3. Guilherme Santana's Use Case	60
4. Joana Cruz's Use Case	63
5. Gonalo Gomes's Use Case	64
E. Metrics Analysis	66
1. Lines of code	66
a. Methods	66
b. Classes	68
c. Interfaces	69
2. Dependency	70
a. Classes	70
b. Interfaces	71
c. Packages	71
d. Conclusion	72
3. Complexity Metrics	73
a. Cognitive complexity – CogC	73
b. Essential cyclomatic complexity – ev(G)	73
c. Design Complexity – iv(G)	74
d. Cyclomatic Complexity – v(G)	75
e. Conclusion	75
4. Martin Packaging Metrics	76
a. Efferent coupling	76
b. Afferent coupling	77
c. Abstractness	77

d. Instability	78
e. Abstractness and Instability are closely related	78
f. Distance from the Main sequence	79
5. Chidamber and Kemerer metrics	80
F. Conclusion	83

A. Introduction to our Git Repository

1. Relevant branches

In terms of SCRUM the team started by creating a branch per week/per sprint, those are the *week1*, *week2* and *week3* branches. Eventually we felt that it would be easier to just add some folders per sprint on one of the branches and keep everything on the same place.

So, in **week3** branch there's a folder for each phase of the project and for each sprint, this is where all the files related to code analyses and SCRUM management can be found, from code smells and patterns to use cases and metrics, to the Burndown Charts and Sprint Backlogs.

As according to the *Issues with Git and Gantt Project* section right below this, a few branches were created to try and run the stable version of the project, this are: *stable*, *stable_version*, *Stable-2_8_9* and *test*. **Stable-2_8_9** worked at the start but then we had the issue mentioned next and only with a test after a quite unusual fix all team members were able to run the stable version of the project.

The branches relevant to the functionalities we added to Gantt Project are as follows:

Tag_feature, **test**, **tag_windows** and **tag_mostworking** have commits related to the development of the creating and assigning tags to tasks in order to categorize them feature.

test_label and **favManager**, have commits related to the development of the favorite or recurrent tasks list feature.

tag_mostworking ended up becoming the branch in which we merged all our work, it has the project running with both functionalities implemented.

Note that we started programming the tags related feature first and the sub-team of the favorites list feature started second, so when merging or creating branches from others the tags feature code was sometimes based for the favorites list code.

2. Issues with Git and Gantt Project

Although the Project Description and instructions to run the Gantt Project were released around 11 of October, the version we were trying to use was not stable and so only after another week or two the instructions on how to run the stable version were made available.

Because a very specific version of Gradle and Java was required for the project to build a lot of students ran into some complications with incompatible versions and had to reconfigure their environmental variables among other things and some needed the teacher's assistance for this setup.

Also our group hadn't selected the option to get all branches when forking the professor's Gantt Project repository, so we had some trouble getting the branch that had the stable version code in the first place and created a few branched trying. When we finally had the branch `Stable-2_8_9` the same as the branch `BRANCH_2_8_9` from the teacher's repository and two out of five people being able to run it, we noticed that it just suddenly stopped working about a week after.

No commits were made, and no one even pulled the repository from remote, it just stopped working on our computers. The only action done was a branch `tag_feature` created by Joana to which she pushed the first interfaces and classes we developed for extending the project, *Tag*, *TagImpl* and *TestTag*.

Our group requested help from the teachers Miguel Goulão and Vasco Amaral on this matter but only a classmate James Furtado was able to help with a very unusual fix, that reveals bugs in the actual project's build code: The fix was to change the *getBuildNum()* method in *ganttproject-builder/build.gradle* (line 107) in order to just return 476. This version was on the branch named *test*

Even after this fix and all the elements of the group being finally able to run the stable version of the project, we still had issues with commits. Barbara committed a few classes, and it just wouldn't run, although we found out that it wasn't compiling because a few returns were missing, the error message we first got wasn't very helpful and the group started being afraid of pushing things to the repository.

So, a lot of commits weren't done by the person who actually wrote the code or some code was written by more than one student as we felt that a Pair Programming approach, along

with all its benefits, could help us manage the time better as the project's delivery date ran closer each time one of these problems occurred. This report's *Who coded what and respective commits* section details what was implemented by whom.

It's also important to mention that Gantt Project apart from all the code smells, messy code, complex and many times unnecessarily complex solutions that made extending its functionalities quite challenging, also had the twist that the application was available in several languages, which made searching the code for menu titles or button labels very hard.

3. Who coded what

Although the commits in our git repository might be a bit confusing, this information is explicit in the backlog of each sprint.

4. What were the plans and what were the results

Initially, we had some plans for the two requested additional features. We decided to implement a system of tags for tasks and other feature for favoriting tasks.

For the tag system, the plan was to implement a filter on the calendar that would allow the user to filter through the tasks in the calendar so that the only tasks seen would be the ones with the selected tag in the filter. This turned out to be a little different as we found it a bit difficult to implement it the way we planned to. We have created a button on the toolbar with "etiquetas" which, when clicked, shows four options: "Nova etiqueta", "Apagar etiqueta", "Editar etiqueta" e "Tarefas por etiqueta".

When clicking on "Nova etiqueta" a panel shows up where the user can define a name for the tag as well as a color. After doing this, one can enter the properties of an existing task and choose in a dropdown list the created tag. This will change the color of the task to the color of the tag after clicking OK to exit the task properties.

When clicking on "Apagar etiqueta" a panel shows up where the user can write the name of the tag he wants to remove. After inserting the name and clicking OK the tag is removed and any task that had that tag associated will have its color changed back to its original color.

When clicking on “Editar etiqueta” a panel shows up where the user can insert the name of the tag he wants to edit, the new name for the tag, as well as the new pretended color. After doing so and clicking OK the tag has now a different name as well as a different color and any task that was associated with that tag has now a different color.

Finally, when clicking on “Tarefas por etiqueta” a panel shows up containing a tab for each task. Each tab contains the name of the tag with its respective color as well as the name of the tasks that are associated with that tag. When clicking OK the panel closes.

Now, for the favoriting system, the plan was to have a tab on the left side of the app, similar to the gantt tab, that would contain the favorite tasks. It would then be possible to duplicate them. This also turned out to be a bit difficult so, after trying to make it happen, we decided to have a button on the toolbar called “Favoritos”. Clicking on this button makes another button emerge called “Mostrar lista de favoritos”. When clicked, this button opens a panel with a list of the favorite tasks. The user can then copy a task to duplicate it by selecting the wanted task and clicking on “Copiar Tarefa”.

To favorite a task, the user needs to go on the properties of the task he wants to favorite and then click on the checkbox “Favorito”.

B. Code Smells

1. Bárbara Correia's Code Smell 1

Reviewer: Joana Cruz

```
static class Functions {  
    static Predicate<Object> NOT_EDITABLE = new Predicate<Object>() {  
        @Override  
        public boolean apply(Object input) {  
            return false;  
        }  
    };  
  
    static Predicate<Object> ALWAYS_EDITABLE = new Predicate<Object>() {  
        @Override  
        public boolean apply(Object input) {  
            return true;  
        }  
    };  
}
```

Location on the code base:

code\biz.ganttproject.core\src\main\java\biz.ganttproject\core\model\task\TaskDefaultColumn.java

Code Smell identification:

Duplicated code. This code snippet has two methods with the same exact structure, where the only changes are in the return value and in the name of the Predicate.

Refactoring proposal:

A possible solution for this code smell is to use an if statement to alter the value of a variable to later return in a function using the same structure, as well as to decide what to name the Predicate. The variable used should have a boolean value. In conclusion, in the end we should have two static objects instead of one.

2. Bárbara Correia's Code Smell 2

Reviewers: Gonalo Rodrigues & Guilherme Santana

```
public class Connector {  
    (...)  
    private final Vector myStart;  
    private final Vector myEnd;  
    private final String myStyle;  
  
    Connector(Vector start, Vector end, String style) {  
        myStart = start;  
        myEnd = end;  
        myStyle = style;  
    }  
  
    Vector getStart() {  
        return myStart;  
    }  
  
    Vector getEnd() {  
        return myEnd;  
    }  
  
    String getStyleName() {  
        return myStyle;  
    }  
}
```

Location on the code base:

code\biz.ganttproject.core\src\main\java\biz.ganttproject\core\chart\scene\gant\Connector.j
ava

Code Smell identification:

Data Class. This code snippet reveals a class that is too small. It only contains data and no real functionality, only getter methods.

Refactoring proposal:

In order to fix this code smell we would have to add some methods to this class that aren't getters nor setters. In case there aren't any methods to add, this class shouldn't exist.

3. Bárbara Correia's Code Smell 3

Reviewer: Joana Cruz

```
public void drawDependencies(Collection<Connector> connectors) {  
    if (myChartApi.getBarHeight() != myBarHeight) {  
        myFinishArrow = new Canvas.Arrow((int)(0.7f * myChartApi.getBarHeight()), (int)(0.3f*myChartApi.getBarHeight()));  
        myBarHeight = myChartApi.getBarHeight();  
    }  
    (...)  
    // 44 lines  
    (...)  
    lastLine.setStyle(lineStyle);  
    lastLine.setArrow(myFinishArrow);  
}  
}  
}
```

Location on the code base:

code\biz.ganttproject.core\src\main\java\biz.ganttproject\core\chart\scene\gantt\Dependenc
ySceneBuilder.java

Code Smell identification:

Long Method. This method is very complex and it shouldn't be. This complexity makes it difficult to read the method and know what it does.

Refactoring proposal:

To make this method more readable and shorter, one could create auxiliary methods in each if statement which would contain the code of the same region.

4. Gonalo Rodrigues's Code Smell 1

Reviewer: Joana Cruz & Gonalo Gomes

```
f.run0(new Function<CustomPropertyDefinition, S>() {  
    @Override  
    public S apply(@Nullable CustomPropertyDefinition def) {  
        String name;  
        switch (def.getPropertyClass()) {  
            case BOOLEAN:  
                name = "FLAG";  
                break;  
            case INTEGER:  
            case DOUBLE:  
                name = "NUMBER";  
                break;  
            case TEXT:  
                name = "TEXT";  
                break;  
            case DATE:  
                name = "DATE";  
                break;  
            default:  
                assert false : "Should not be here";  
                name = "TEXT";  
        }  
        for (int i = 1; i <= 30; i++) {  
            S tf1 = (S)Enum.valueOf(enumClass, name + String.valueOf(i));  
            if (mpxjFields.contains(tf1)) {  
                return tf1;  
            }  
        }  
        return null;  
    }  
});
```

Location on the code base:

code/biz.ganttproject.core/src/main/java/biz/ganttproject/core/option

Code Smell identification:

Switch Statements: This code has a "Switch statement" that checks the type of an object and assigns a value to a string based on the switch case. This code, although functional, takes up a lot of space and makes the code confusing and inextensible.

Refactoring proposal:

In this case, where we sought to differentiate a property from the class, we could create subclasses where the method was implemented in different ways based on this property. This polymorphism technique would replace the "switch statement" with a simple `"String name = def.getPropertyClass();"`

5. Gonalo Rodrigues's Code Smell 2

Reviewer: Joana Cruz

(...)

```
public Object getID() {  
    return myID;  
}
```

```
public Object getOldValue() {  
    return myOldValue;  
}
```

```
public Object getNewValue() {  
    return myNewValue;  
}
```

```
public Object getTriggerID() {  
    return myTriggerID;  
}
```

(...)

Location on the code base:

code/biz.ganttproject.core/src/main/java/biz/ganttproject/core/option/ChangeValueEvent.java

Code Smell identification:

Data Class: This class is only used to store values, query them through getters, and does not make any changes to them or implement any other features.

Refactoring proposal:

Implement more functionality in the class in order to give more meaning and functionality to it in order to justify its existence. If this is not possible, in the context of the project, the existence of this class does not make much sense and therefore I suggest that it be removed.

6. Gonçalo Rodrigues's Code Smell 3

Reviewer: Bárbara Correia & Joana Cruz

(...)

```
public void run() {  
    getUiFacade().getUndoManager().undoableEdit(ourLocalizer.formatText("importCalendar"), new Runnable() {
```

(...)

)

Location on the code base:

code/biz.ganttproject.impex.ical/src/main/java/biz/ganttproject/impex/ical/IcsFileImporter.java

Code Smell identification:

Inappropriate Intimacy: This call invokes a method of the class which then calls another method of the class that was returned by the first method and which in turn calls another method of the class that was returned by the previous method.

Refactoring proposal:

Build a method in the "getUiFacade" method class where a method of the "getUndoManager" method class is called, and therefore the "undoableEdit(...)" method would be called in another class. In short, instead of having a method that calls another and so on successively within the same class, these calls would be made in "ladder", with the creation or use of classes.

7. Guilherme Santana's Code Smell 1

Reviewer: Gonalo Gomes

(...)

```
case WORKING:
    if ((nextUnitMask & 1) == 1) {
        return nextUnitStart;
    }
    break;
case NON_WORKING:
case WEEKEND:
case HOLIDAY:
```

(...)

)

Location on the code base:

ganttproject\biz.ganttproject.core\bin\main\biz\ganttproject\core\calendar\GPCalendarBase.java

Code Smell identification:

Speculative Generality – the method doFindClosest(...) has unnecessary code at the time in the switch –“(...) case NOT_WORKING, case WEEKEND, case HOLIDAYS. (...)”

Refactoring proposal:

Eliminate the speculative code and add a related comment for the future when we implement the code

8. Guilherme Santana's Code Smell 2

Reviewer: Bárbara Correia

(...)

```
public void setPublicHolidays(Collection<CalendarEvent> holidays) {  
    }  
    public String getBaseCalendarID() {  
        return null;  
    }  
    public void setBaseCalendarID(String id) {  
    }  
    public void importCalendar(GPCalendar calendar, ImportCalendarOption importOption) {  
    }  
}
```

Location on the code base:

ganttproject\biz.ganttproject.core\bin\main\biz\ganttproject\core\calendar\AlwaysWorkingTimeCalendarImpl.java

Code Smell identification:

Dead Code – the methods in the end of the class are not finished and/or not used.

Refactoring proposal:

Eliminate the dead code if it wouldn't be used.

9. Guilherme Santana's Code Smell 3

Reviewer: Gonalo Rodrigues

(...)

```
public PrjInfos(String sProjectName, String sDescription, String sOrganization, String weblink) {  
    this._sProjectName = sProjectName;  
    this._sDescription = sDescription;  
    this._sOrganization = sOrganization;  
    this._sWebLink = sWebLink;  
}  
  
public String getName() {  
    return this._sProjectName;  
}  
  
public void setName(String projectName) {  
    this._sProjectName = projectName;  
}  
  
public String getDescription() {  
    return this._sDescription;  
}  
  
public void setDescription(String description) {  
    this._sDescription = description;  
}  
  
public String getOrganization() {  
    return this._sOrganization;  
}  
  
public void setOrganization(String organization) {  
    this._sOrganization = organization;  
}  
  
public String getWebLink() {  
    return this._sWebLink;  
}  
  
public void setWebLink(String webLink) {  
    this._sWebLink = webLink;  
}  
}
```

Location on the code base:

ganttproject\bin\main\net\sourceforge\ganttproject\PrjInfos.class

Code Smell identification:

Data Class – PrjInfos is a class only with getters and setters and with no real functionality.

Refactoring proposal:

Review the code that uses the class. May find functionality that would be better located in the data class itself. If it can't be done another solution is to add other functionalities/methods in this class.

10. Joana Cruz's Code Smells 1

Reviewer: Guilherme Santana

```
public class Canvas {
    private ArrayList<Rectangle> myRectangles = new ArrayList<Rectangle>();

    private ArrayList<Line> myLines = new ArrayList<Line>();

    private ArrayList<Text> myTexts = new ArrayList<Text>();

    private Map<Object, Shape> myModelObject2primitive = new WeakHashMap<Object, Shape>();

    private List<Canvas> myLayers = new ArrayList<Canvas>();

    private int myDeltaX;

    private int myDeltaY;

    private List<TextGroup> myTextGroups = new ArrayList<TextGroup>();

    private final DummySpatialIndex<Text> myTextIndex = new DummySpatialIndex<Text>();

    private final DummySpatialIndex<Rhombus> myRhombusIndex = new DummySpatialIndex<>();

    /** Horizontal alignments for texts */
    public enum HAlignment {
        CENTER, LEFT, RIGHT
    };

    /** Vertical alignments for texts */
    public enum VAlignment {
        CENTER, TOP, BOTTOM
    };

    public static class Shape {
        private Color myBackgroundColor;

        private Color myForegroundColor;

        private String myStyleName;

        (...)
    }

    public static class Polygon extends Shape {
        private final int myLeftX;
```

```

private final int myRightX;

private final int myTopY;
private final int myBottomY;

private final int[] myPointsX;
private final int[] myPointsY;

private Polygon(int... points) {
    myPointsX = new int[points.length / 2];
    myPointsY = new int[points.length / 2];

    int leftX = Integer.MAX_VALUE;
    int topY = Integer.MAX_VALUE;
    int rightX = Integer.MIN_VALUE;
    int bottomY = Integer.MIN_VALUE;
    for (int i = 0; i < points.length / 2; i++) {
        leftX = Math.min(leftX, points[i * 2]);
        rightX = Math.max(rightX, points[i * 2]);
        topY = Math.min(topY, points[i * 2 + 1]);
        bottomY = Math.max(bottomY, points[i * 2 + 1]);

        myPointsX[i] = points[i * 2];
        myPointsY[i] = points[i * 2 + 1];
    }
    myLeftX = leftX;
    myRightX = rightX;
    myTopY = topY;
    myBottomY = bottomY;
}

(...)

public static class Rectangle extends Polygon {
    public Paint myPaint;

    private Rectangle(int leftx, int topy, int width, int height) {
        super(leftx, topy, leftx + width, topy + height);
    }

    public Paint getBackgroundPaint() {
        return myPaint;
    }

    public void setBackgroundPaint(Paint paint) {
        myPaint = paint;
    }
}

```

```

public static class Rhombus extends Polygon {
    private Rhombus(int leftx, int topy, int diagWidth, int diagHeight) {
        super(
            leftx, topy + diagHeight / 2,
            leftx + diagWidth / 2, topy,

            leftx + diagWidth, topy + diagHeight / 2,
            leftx + diagWidth / 2, topy + diagHeight
        );
    }
}

```

```

public static class Arrow extends Shape {
    public static Arrow NONE = new Arrow(0, 0);
    public static Arrow FINISH = new Arrow(7, 3);
    private final int myLength;
    private final int myWidth;

    (...)

```

```

public static class Line extends Shape {
    private final int myStartX;

    private final int myStartY;

    private final int myFinishX;

    private final int myFinishY;

    private Arrow myArrow = Arrow.NONE;

    (...)

```

```

public static class Label
    (...)

```

```

public static class Text extends Shape {
    (...)

```

```

public static class TextGroup {
    (...)

```

Location on the code base:

code/biz.ganttproject.core/src/main/java/biz/ganttproject/core/chart/canvas/Canvas.java

Code Smell identification:

Large Class- The Canvas class has several nested classes making this class very large (703 lines).

Although the subclasses have their role in the operations of the top class this nesting, mainly due to the dimension that the class acquires, makes the code confusing and hard to read and there are many different objects to be represented and specified in this class making it very complex.

Refactoring proposal:

Separate the nested classes into their own classes and files and follow a hierarchy of related classes, eg. Line is a subclass of Shape.

11. Joana Cruz's Code Smell 2

Reviewer: Gonalo Rodrigues

```
public class ShapeConstants {  
    public static final ShapePaint TRANSPARENT = new ShapePaint(4, 4, new int[]{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0});  
    public static final ShapePaint DEFAULT = new ShapePaint(4, 4, new int[]{1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1});  
    public static final ShapePaint CROSS = new ShapePaint(4, 4, new int[]{0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0});  
    public static final ShapePaint VERT = new ShapePaint(4, 4, new int[]{1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0});  
    public static final ShapePaint HORZ = new ShapePaint(4, 4, new int[]{0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1});  
    public static final ShapePaint GRID = new ShapePaint(4, 4, new int[]{1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1});  
    public static final ShapePaint ROUND = new ShapePaint(4, 4, new int[]{0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0});  
    public static final ShapePaint NW_TRIANGLE = new ShapePaint(4, 4, new int[]{1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0});  
    public static final ShapePaint NE_TRIANGLE = new ShapePaint(4, 4, new int[]{0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0});  
    public static final ShapePaint SW_TRIANGLE = new ShapePaint(4, 4, new int[]{0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0});  
    public static final ShapePaint SE_TRIANGLE = new ShapePaint(4, 4, new int[]{0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1});  
    public static final ShapePaint DIAMOND = new ShapePaint(4, 4, new int[]{0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0});  
    public static final ShapePaint DOTS = new ShapePaint(4, 4, new int[]{1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0});  
    public static final ShapePaint DOT = new ShapePaint(4, 4, new int[]{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0});  
    public static final ShapePaint SLASH = new ShapePaint(4, 4, new int[]{1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1});  
    public static final ShapePaint BACKSLASH = new ShapePaint(4, 4, new int[]{0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0});  
    public static final ShapePaint THICK_VERT = new ShapePaint(4, 4, new int[]{0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1});  
    public static final ShapePaint THICK_HORZ = new ShapePaint(4, 4, new int[]{0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0});  
    public static final ShapePaint THICK_GRID = new ShapePaint(4, 4, new int[]{0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0});  
    public static final ShapePaint THICK_SLASH = new ShapePaint(4, 4, new int[]{1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0});  
    public static final ShapePaint THICK_BACKSLASH = new ShapePaint(4, 4, new int[]{0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1});  
    public static ShapePaint[] PATTERN_LIST;  
  
    public ShapeConstants() {  
    }  
    static {  
        PATTERN_LIST = new ShapePaint[]{TRANSPARENT, DEFAULT, CROSS, VERT, HORZ, GRID, ROUND, NW_TRIANGLE, NE_TRIANGLE, SW_TRIANGLE, SE_TRIANGLE, DIAMOND, DOTS, DOT, SLASH, BACKSLASH, THICK_VERT, THICK_HORZ, THICK_GRID, THICK_SLASH, THICK_BACKSLASH};  
    }  
}
```

Location on the code base:

code/biz.ganttproject.core/src/main/java/biz/ganttproject/core/chart/canvas/render/ShapeConstants.java

Code Smell identification:

Class of constants instead of enumeration - This class defines a sort of list of constants instead of using, per example, an enumeration, it's a class that, in addition to the constants themselves, has an array (PATTERN_LIST) of these constants which is difficult to navigate, because to access a particular constant we would have to know it's index in the array.

Refactoring proposal:

Transform this class into an enumeration or define the constants in the classes that use them.

12. Joana Cruz's Code Smell 3

Reviewer: Gonalo Gomes

```
public void drawDependencies(Collection<Connector> connectors) {
    if (this.myChartApi.getBarHeight() != this.myBarHeight) {
        this.myFinishArrow = new Canvas.Arrow((int)(0.7F * (float)this.myChartApi.getBarHeight()),
(int)(0.3F * (float)this.myChartApi.getBarHeight()));
        this.myBarHeight = this.myChartApi.getBarHeight();
    }
    Canvas primitiveContainer = this.myOutputCanvas;
    Iterator var3 = connectors.iterator();
    while(var3.hasNext()) {
        Connector connector = (Connector)var3.next();
        Connector.Vector dependantVector = connector.getEnd();
        Connector.Vector dependeeVector = connector.getStart();
        String lineStyle = connector.getStyleName();
        Point first;
        Point third;
        if (dependeeVector.getHProjection().reaches(dependantVector.getHProjection().getPoint())) {
            first = new Point(dependeeVector.getPoint().x, dependeeVector.getPoint().y);
            int xEntry = dependantVector.getPoint().x;
            int yEntry = dependantVector.getPoint().y;
            third = new Point(xEntry, dependeeVector.getPoint().y);
            Point third = new Point(xEntry, yEntry);
            primitiveContainer.createLine(first.x, first.y, third.x, third.y).setStyle(lineStyle);

            Canvas.Line secondLine = primitiveContainer.createLine(third.x, third.y, third.x, third.y);
            secondLine.setStyle(lineStyle);
            secondLine.setArrow(this.myFinishArrow);
        } else {
            Point second;
            if (dependantVector.getHProjection().reaches(dependeeVector.getHProjection().getPoint(3))) {
                first = dependeeVector.getPoint(3);
                second = new Point(first.x, dependantVector.getPoint().y);
                primitiveContainer.createLine(dependeeVector.getPoint().x, dependeeVector.getPoint().y,
first.x, first.y).setStyle(lineStyle);
                primitiveContainer.createLine(first.x, first.y, second.x, second.y).setStyle(lineStyle);
                Canvas.Line line = primitiveContainer.createLine(second.x, second.y,
dependantVector.getPoint().x, dependantVector.getPoint().y);
                line.setStyle(lineStyle);
                line.setArrow(this.myFinishArrow);
            } else {
```

```
first = dependeeVector.getPoint(10);  
second = dependantVector.getPoint(10);
```

(...)

Location on the code base:

code/biz.ganttproject.core/src/main/java/biz/ganttproject/core/chart/scene/gantt/DependencySceneBuilder.java

Code Smell identification:

Large method- The drawDependencies() method like others in this class is quite large (52 lines) making it difficult to understand the method or even identify possible problems in the code.

Refactoring proposal:

Create some auxiliary methods to use in the body of this method to divide it into smaller tasks.

13. Gonçalo Gomes's Code Smell 1

Reviewer: Bárbara Correia

```
@Override  
public TimeUnit createTimeUnit(Date date) {  
    // TODO Only works if myBaseDate is not a TimeUnitFunctionOfDateImpl!  
    // (Quarter -> Month -> Day fails!)  
    return new ParameterizedTimeUnitImpl(date);  
}
```

Location on the code base:

biz.ganttproject.core\src\main\java\biz\ganttpproject\core\time\TimeUnitFunctionOfDateImpl.java

Code Smell identification:

Reminder comment Smell - There is a TODO comment that warns that the method only works under certain circumstances

Refactoring proposal:

A precondition should be implemented or the method should be fixed to work properly under all circumstances.

The comment should be removed, though only if the problem is fixed

14. Gonçalo Gomes's Code Smell 2

Reviewer: Joana Cruz

```
/**
 * @author bard
 */
public class TimeUnitDateFrameableImpl extends TimeUnitImpl {
    private final DateFrameable myFramer;

    public TimeUnitDateFrameableImpl(String name, TimeUnitGraph timeUnitGraph, TimeUnit atomUnit,
    DateFrameable framer) {
        super(name, timeUnitGraph, atomUnit);
        myFramer = framer;
    }

    @Override
    public Date adjustRight(Date baseDate) {
        return myFramer.adjustRight(baseDate);
    }

    @Override
    public Date adjustLeft(Date baseDate) {
        return myFramer.adjustLeft(baseDate);
    }

    @Override
    public Date jumpLeft(Date baseDate) {
        return myFramer.jumpLeft(baseDate);
    }
}
```

Location on the code base:

biz.ganttproject.core\src\main\java\biz\ganttpproject\core\time\

Code Smell identification:

Lack of comments - the code snippet presents no comments indicating what it does and how, making interpretation harder.

Refactoring proposal:

Comment the code so that others can better understand what it does

15. Gonçalo Gomes's Code Smell 3

Reviewer: Guilherme Santana

```
public TimeDuration parseDuration(String lengthAsString) throws ParseException {  
    int state = 0;  
    StringBuffer valueBuffer = new StringBuffer();  
    Integer currentValue = null;  
    TimeDuration currentLength = null;  
    lengthAsString += " ";  
    for (int i = 0; i < lengthAsString.length(); i++) {  
        char nextChar = lengthAsString.charAt(i);  
        if (Character.isDigit(nextChar)) {  
            switch (state) {  
                case 0:  
                    if (currentValue != null) {  
                        throw new ParseException(lengthAsString, i);  
                    }  
                    state = 1;  
                    valueBuffer.setLength(0);  
                case 1:  
                    valueBuffer.append(nextChar);  
                    break;  
                case 2:  
                    TimeUnit timeUnit = findTimeUnit(valueBuffer.toString());  
                    if (timeUnit == null) {  
                        throw new ParseException(lengthAsString, i);  
                    }  
                    assert currentValue != null;  
                    TimeDuration localResult = createLength(timeUnit, currentValue.floatValue());  
                    if (currentLength == null) {  
                        currentLength = localResult;  
                    } else {  
                        if (currentLength.getTimeUnit().isConstructedFrom(timeUnit)) {  
                            float recalculatedLength = currentLength.getLength(timeUnit);  
                            currentLength = createLength(timeUnit, localResult.getValue() +  
recalculatedLength);  
                        } else {  
                            throw new ParseException(lengthAsString, i);  
                        }  
                    }  
                }  
            }  
        }  
    }  
}
```

```

        }
    }
    state = 1;
    currentValue = null;
    valueBuffer.setLength(0);
    valueBuffer.append(nextChar);
    break;
}
} else if (Character.isWhitespace(nextChar)) {
    switch (state) {
    case 0:
        break;
    case 1:
        currentValue = Integer.valueOf(valueBuffer.toString());
        state = 0;
        break;
    case 2:
        TimeUnit timeUnit = findTimeUnit(valueBuffer.toString());
        if (timeUnit == null) {
            throw new ParseException(lengthAsString, i);
        }
        assert currentValue != null;
        TimeDuration localResult = createLength(timeUnit, currentValue.floatValue());
        if (currentLength == null) {
            currentLength = localResult;
        } else {
            if (currentLength.getTimeUnit().isConstructedFrom(timeUnit)) {
                float recalculatedLength = currentLength.getLength(timeUnit);
                currentLength = createLength(timeUnit, localResult.getValue() +
recalculatedLength);
            } else {
                throw new ParseException(lengthAsString, i);
            }
        }
        state = 0;
        currentValue = null;
        break;
    }
} else {

```



```

switch (state) {
case 1:
    currentValue = Integer.valueOf(valueBuffer.toString());
case 0:
    if (currentValue == null) {
        throw new ParseException(lengthAsString, i);
    }
    state = 2;
    valueBuffer.setLength(0);
case 2:
    valueBuffer.append(nextChar);
    break;
}
}
}
if (currentValue != null) {
    currentValue = Integer.valueOf(valueBuffer.toString());
    TimeUnit dayUnit = findTimeUnit("d");
    currentLength = createLength(dayUnit, currentValue.floatValue());
}
return currentLength;
}

```

Location on the code base:

biz.ganttproject.core\src\main\java\biz\ganttpproject\core\time\impl\GPTimeUnitStack

Code Smell identification:

Large Method - The method has 94 lines and is hard to read and understand.

Refactoring proposal:

The method should be divided into smaller, more concise sub-methods.

C. Design Patterns

1. Bárbara Correia Design Pattern 1

Reviewer: Gonçalo Rodrigues

```
public class UndoManagerImpl implements GPUndoManager {  
    @Override  
    public boolean canUndo() {  
        return mySwingUndoManager.canUndo();  
    }  
  
    (...)  
  
    @Override  
    public void die() {  
        if (swingEditImpl != null) {  
            swingEditImpl.die();  
        }  
        if (mySwingUndoManager != null) {  
            mySwingUndoManager.discardAllEdits();  
        }  
        fireUndoReset();  
    }  
}
```

Location on the code base:

code\ganttproject\src\main\java\net\sourceforge\ganttproject\undo\UndoManagerImpl.java

Pattern identification:

Command pattern. This command pattern encapsulates a request as an object of its own. In this code Snippet are used methods from mySwingUndoManager, for example. The method die(), for example, calls the method die() from de class swingEditImpl.

2. Bárbara Correia's Design Pattern 2

Reviewer: Guilherme Santana

```
public interface TaskContainmentHierarchyFacade {  
    Task[] getNestedTasks(Task container);  
  
    Task[] getDeepNestedTasks(Task container);  
  
    boolean hasNestedTasks(Task container);  
  
    Task getRootTask();  
  
    Task getContainer(Task nestedTask);  
  
    (...)  
}
```

Location on the code base:

code\ganttproject\src\main\java\net\sourceforge\ganttproject\task\TaskContainmentHierarchyFacade.java

Pattern identification:

Facade Method. The TaskContainmentHierarchyFacade interface implements a lot of methods used to alter the state of other classes that together form a subsystem of the class.

3. Bárbara Correia's Design Pattern 3

Reviewer: Joana Cruz

on DialogBuilder.java:

```
OkAction proxy = new OkAction() {  
    // These two steps handel the case when focus is somewhere in text input  
    // and user hits Ctrl+Enter  
    // First we want to move focus to OK button to allow focus listeners, if any,  
    // to catch focusLost event  
    // Second, we want it to happen before original OkAction runs  
    // So we wrap original OkAction into proxy which moves focus and schedules "later" command  
    // which call the original action. Between them EDT sends out focusLost events.  
    (...)  
};  
_btn.setAction(proxy);  
nextButton = _btn;
```

on OkAction.java:

```
public abstract class OkAction extends GPAction {  
    private boolean isDefault = true;  
  
    public OkAction() {  
        this("ok");  
    }  
  
    public OkAction(String key) {  
        super(key);  
    }  
  
    public boolean isDefault() {  
        return isDefault;  
    }  
  
    protected void setDefault(boolean value) {  
        isDefault = value;  
    }  
}
```

Location on the code base:

code\ganttproject\src\main\java\net\sourceforge\ganttproject\DialogBuilder.java

code\ganttproject\src\main\java\net\sourceforge\ganttproject\action\OkAction.java

Pattern identification:

Proxy pattern. The class OkAction acts as a simplified, or lightweight version, of the original object, the class GPAction.

4. Gonçalo Rodrigues's Design Pattern 1

Reviewer: Joana Cruz

```
(...)  
public class DefaultDateOption extends GPAbstractOption<Date> implements DateOption  
(...)  
public class DefaultDoubleOption extends GPAbstractOption<Double> implements DoubleOption  
(...)  
public class DefaultEnumerationOption<T> extends GPAbstractOption<String> implements EnumerationOption  
(...)  
public class DefaultFileOption extends DefaultStringOption implements FileOption  
(...)  
public class DefaultFontOption extends GPAbstractOption<FontSpec> implements FontOption  
(...)  
public class DefaultIntegerOption extends GPAbstractOption<Integer> implements IntegerOption  
(...)  
public class DefaultMoneyOption extends GPAbstractOption<BigDecimal> implements MoneyOption  
(...)  
public class DefaultStringOption extends GPAbstractOption<String> implements StringOption  
(...)  
(Entre outras classes)
```

Location on the code base:

code/biz.ganttproject.core/src/main/java/biz/ganttproject/core/option

Pattern identification:

Template Method: The "GPAbstractOption" class is extended by most classes in the "Option" folder. The GPAbstractOption class implements some of the same methods for all its subclasses but these implement other methods that have different implementations, which vary from class to class according to the purpose and needs of the client. This way you reuse code (it's in the superclass instead of in all the subclasses, repeated) and makes the code more flexible and versatile in case you need to make any adjustments or changes.

5. Gonalo Rodrigues's Design Pattern 2

Reviewers: Joana Cruz & Gonalo Rodrigues

(on TaskManagerImpl.java)

```
private final TaskImpl myRoot;

(...)

private TaskImpl createRootTask() {
    Calendar c = CalendarFactory.newCalendar();
    Date today = c.getTime();
    TaskImpl root = new GanttTask(null, CalendarFactory.createGanttCalendar(today), 1, this, -1, "");
    root.setStart(CalendarFactory.createGanttCalendar(today));
    root.setDuration(createLength(getConfig().getTimeUnitStack().getDefaultTimeUnit(), 1));
    root.setExpand(true);
    root.setName("root");
    return root;
}

(...)

public Date getProjectStart() {
    if (myTaskMap.isEmpty()) {
        return myRoot.getStart().getTime();
    }

    Result result = getAlgorithmCollection().getProjectBoundsAlgorithm().getBounds(tasksToActivities(myTaskMap.getTasks()));
    return result.lowerBound;
}
```

(on PertChart.java)

```
TaskManager myTaskManager;

(...)

public Date getStartDate() {
    return myTaskManager.getProjectStart();
}

(...)
```

(on GanttTask.java)

```
(...)

public class GanttTask extends TaskImpl implements Serializable {
    public GanttTask(String name, GanttCalendar start, long length, TaskManagerImpl taskManager, int taskID, @NotNull String taskUid) {
        super(taskManager, taskID, taskUid);
        setName(name);
        setStart(start);
        setDuration(taskManager.createLength(length));
        enableEvents(true);
    } (...)
```

Location on the code base:

code\ganttproject\src\main\java\net\sourceforge\ganttproject\task\TaskManagerImpl.java
code\org.ganttproject.chart.pert\src\main\java\org\ganttproject\chart\pert\PertChart.java
code\ganttproject\src\main\java\net\sourceforge\ganttproject\GanttTask.java

Pattern identification:

Command Pattern- The "TaskManagerImpl.java" class works as a command class among many other classes, including the "PertChart.java" class and the "GanttTask.java" class and these two classes, although they never have access to the other, are able to communicate with each other. through the "TaskManagerImpl.java" class.

6. Gonçalo Rodrigues's Design Pattern 3

Reviewer: Joana Cruz & Bárbara Correia

(on UIFacade.java)

```
public interface UIFacade {  
    (... +/- 120 linhas de código ...)  
}
```

(on UIFacadeImpl.java)

```
class UIFacadeImpl extends ProgressProvider implements UIFacade {  
    private final JFrame myMainFrame;  
    private final ScrollingManager myScrollingManager;  
    private final ZoomManager myZoomManager;  
    private final GanttStatusBar myStatusBar;  
    private final UIFacade myFallbackDelegate;  
    private final TaskSelectionManager myTaskSelectionManager;  
    private final List<GPOptionGroup> myOptionGroups = Lists.newArrayList();  
    private final GPOptionGroup myOptions;  
    private final LafOption myLafOption;  
    private final GPOptionGroup myLogoOptions;  
    private final DefaultFileOption myLogoOption;  
    private final NotificationManagerImpl myNotificationManager;  
    private final TaskView myTaskView = new TaskView();  
    private final DialogBuilder myDialogBuilder;  
    private final Map<String, Font> myOriginalFonts = Maps.newHashMap();  
    private final List<Runnable> myOnUpdateComponentTreeUiCallbacks = Lists.newArrayList();  
    private float myLastScale = 0;
```

Location on the code base:

code\ganttpproject\src\main\java\net\sourceforge\ganttpproject\gui\UIFacade.java

code\ganttpproject\src\main\java\net\sourceforge\ganttpproject\UIFacadeImpl.java

Pattern identification:

Facade Method: The "UIFacade" interface implements many methods that aim to change the state of other classes, which together form the subsystem of the class that the "UIFacade" interface implements, the "UIFacadeImpl.java" class. We can admit that most of the variables of the "UIFacadeImpl.java" class (above, in the 'Code Spinnet' section) are instances of its subclasses.

7. Guilherme Santana's Design Pattern

Reviewer: Gonalo Gomes

Code1:

(...)

```
public interface TreeUiFacade<T> {  
    Component getTreeComponent();  
    ColumnList getVisibleFields();  
    boolean isVisible(T var1);  
    boolean isExpanded(T var1);  
    void setExpanded(T var1, boolean var2);  
    void applyPreservingExpansionState(T var1, Predicate<T> var2);  
    void setSelected(T var1, boolean var2);  
    void clearSelection();  
    void makeVisible(T var1);  
    GAction getNewAction();  
    GAction getPropertiesAction();  
    GAction getDeleteAction();  
    void startDefaultEditing(T var1);  
    void stopEditing();  
    AbstractAction[] getTreeActions();  
}  
(...)
```

Code2:

```
public interface ResourceTreeUIFacade extends TreeUiFacade<HumanResource> {  
    AbstractAction getMoveUpAction();  
    AbstractAction getMoveDownAction();  
    TimelineChart.VScrollController getVScrollController();  
}
```

Location on the code base:

code1:

ganttproject/bin/main/net/sourceforge/ganttproject/gui/TreeUiFacade.class

code2:

ganttproject/bin/main/net/sourceforge/ganttproject/gui/ResourceTreeUIFacade.class

Pattern identification:

Facade – The interface ResourceTreeUIFacade, an extension of the interface TreeUiFacade, in an encapsulation of a subsystem with four classes. This interface acts as a point of entry into this subsystem.

8. Guilherme Santana's Design Pattern 2

Reviewer: Gonçalo Rodrigues

```
(...)  
boolean contains(Task var1);  
public interface Factory {  
    TaskContainmentHierarchyFacade createFacade();  
}  
}
```

Location on the code base:

ganttproject/bin/main/net/sourceforge/ganttproject/task/TaskContainmentHierarchyFacade.class

Pattern identification:

Factory method - This interface hides the creation of the instance

TaskContainmentHierarchyFacade. This interface is useful in the class FacadeFactoryImpl nested in the class TaskManagerImpl

9. Guilherme Santana's Design Pattern 3

Reviewer: Bárbara Correia

```
public class CommandLineExportApplication {

    private LoggerApi logger;

    public CommandLineExportApplication() {

        throw new Error("Unresolved compilation problems: \n\tThe import biz.ganttproject.LoggerApi cannot be resolved\n\tThe import net.sourceforge.ganttproject.IGanttProject cannot be resolved\n\tLoggerApi cannot be resolved to a type\n\tThe method create(String) from the type GPLogger refers to the missing type LoggerApi\n\tIGanttProject cannot be resolved to a type\n\tIGanttProject cannot be resolved to a type\n\tLoggerApi cannot be resolved to a type\n\tConsoleProgressProvider cannot be resolved to a type\n");

    }

    public boolean export(Args var1, IGanttProject var2, UIFacade var3) {

        throw new Error("Unresolved compilation problem: \n\tIGanttProject cannot be resolved to a type\n");

    }

    boolean export(Exporter var1, Args var2, IGanttProject var3, UIFacade var4) {

        throw new Error("Unresolved compilation problems: \n\tIGanttProject cannot be resolved to a type\n\tLoggerApi cannot be resolved to a type\n\tConsoleProgressProvider cannot be resolved to a type\n");

    }

    public static class Args {

        @Parameter(
            names = {"-export"},
            description = "Export format"
        )

        public String exporter;

        @Parameter(
            names = {"-stylesheet"},
            description = "Stylesheet used for export"
        )

        public String stylesheet;

        @Parameter(
            names = {"-chart"},
            description = "Chart to export (resource or gantt)"
        )

        public String chart;

        @Parameter(
            names = {"-zoom"},
            description = "Zoom scale to use in the exported charts"
        )

        public Integer zooming;

        @Parameter(
            names = {"-o", "-out"},
            description = "Output file name",
            converter = FileConverter.class
        )

    }

}
```

```

public File outputFile;

@Parameter(
    names = {"-expand-resources"},
    description = "Expand resource nodes on the resource load chart"
)

public boolean expandResources;

@Parameter(
    names = {"-expand-tasks"},
    description = "Expand all tasks nodes on the Gantt chart",
    arity = 1
)

public boolean expandTasks;


public Args() {
    throw new Error("Unresolved compilation problems: \n\tThe import biz.ganttproject.LoggerApi cannot be resolved\n\tThe import net.sourceforge.ganttproject.IGanttProject cannot be resolved\n\tLoggerApi cannot be resolved to a type\n\tThe method create(String) from the type GPLogger refers to the missing type LoggerApi\n\tIGanttProject cannot be resolved to a type\n\tIGanttProject cannot be resolved to a type\n\tLoggerApi cannot be resolved to a type\n\tConsoleProgressProvider cannot be resolved to a type\n");
}

}

}

code2:
} else {
    appBuilder.whenDocumentReady(project -> {
        var executor = Executors.newSingleThreadExecutor();
        executor.submit(() -> {
            var cliApp = new CommandLineExportApplication();
            cliApp.export(appBuilder.getCliArgs(), project, ((GanttProject) project).getUIFacade());
            GanttProject.doQuitApplication(true);
        });
        return Unit.INSTANCE;
    });
}

```

Location on the code base:

code1:

code/ganttproject/src/main/java/net/sourceforge/ganttproject/application/MainApplication.java

code2:

ganttproject/bin/main/net/sourceforge/ganttproject/export/CommandLineExportApplication.class

Pattern identification:

Command Pattern – the class MainApplication – the invoker – invokes an object from CommandLineExportApplication to complete a task. This class CommandLineExportApplication can be considered as a command manager because manipulates and invokes the commands needed to finish this method.

10. Joana Cruz Design Pattern 1

Reviewer: Gonalo Gomes:

```
(...)  
private State myState;  
  
(...)  
public void setGraphics(Graphics2D g) {  
    this.myGraphics = g;  
    this.myState = null;  
}  
  
(...)  
public Object getState() {  
    if (this.myState == null) {  
        this.myState = new State(this.myGraphics.getFontRenderContext(), this.myGraphics.getFont());  
    }  
  
    return this.myState;  
}
```

Location on the code base:

code/biz.ganttproject.core/src/main/java/biz/ganttproject/core/chart/render/TextLengthCalculatorImpl.java

Pattern identification:

Singleton – The static class State cannot be instantiated without an instance of TextLengthCalculatorImpl, in this Outer class we can observe that there can only be a single instance of State(myState) and that in the getState() method we only create new instances of State if myState == null, if an instance already exists then we return the myState which is unique.

11. Joana Cruz's Design Pattern 2

Reviewer: Gonalo Rodrigues

```
public class Canvas{
    private ArrayList<Rectangle> myRectangles = new ArrayList<Rectangle>();

    private ArrayList<Line> myLines = new ArrayList<Line>();

    private ArrayList<Text> myTexts = new ArrayList<Text>();

    private Map<Object, Shape> myModelObject2primitive = new WeakHashMap<Object, Shape>();
    (...)
    public static class Shape {
        (...)
        public Color getBackgroundColor() {
            return myBackgroundColor;
        }

        public void setBackgroundColor(Color myBackgroundColor) {
            this.myBackgroundColor = myBackgroundColor;
        }
        (...)
        public static class Polygon extends Shape
        (...)
        public static class Rectangle extends Polygon {
            (...)
            public Paint getBackgroundPaint() {
                return myPaint;
            }

            public void setBackgroundPaint(Paint paint) {
                myPaint = paint;
            }
        }
        (...)
        public static class Rhombus extends Polygon
        (...)
        public static class Arrow extends Shape
        (...)
        public static class Line extends Shape
        (...)
        (Back to Canvas class)
        public Rhombus createRhombus(int leftx, int topx, int diagWidth, int diagHeight) {
            Rhombus rhombus = new Rhombus(leftx, topx, diagWidth, diagHeight);
```



```

        myRhombusIndex.put(rhombus, rhombus.getLeftX(), rhombus.getBottomY(), rhombus.getWidth(),
rhombus.getHeight());
        return rhombus;
    }

    public Rectangle createRectangle(int leftx, int topy, int width, int height) {
        Rectangle result = createDetachedRectangle(leftx, topy, width, height);
        myRectangles.add(result);
        return result;
    }

    public Rectangle createDetachedRectangle(int leftx, int topy, int width, int height) {
        if (width < 0) {
            width = -width;
            leftx = leftx - width;
        }
        return new Rectangle(leftx + myDeltaX, topy + myDeltaY, width, height);
    }

    public Line createLine(int startx, int starty, int finishx, int finishy) {
        Line result = new Line(startx + myDeltaX, starty + myDeltaY, finishx + myDeltaX, finishy + myDeltaY);
        myLines.add(result);
        return result;
    }
    (...)

```

Location on the code base:

code/biz.ganttproject.core/src/main/java/biz/ganttproject/core/chart/canvas/Canvas.java

Pattern identification:

Factory Method- The Canvas class along with the Shape class (starting at line 63) nested in the one previously mentioned, follows with its subclasses a Factory Method Pattern. Shape defines a general shape and then other classes that extend it (directly or indirectly) like Rectangle and Line (among others) represent a specific shape with some common fields and other specific ones of a rectangle and a line, respectively. The Canvas class has then methods that create the various types of shapes that it needs.

12. Joana Cruz's Design Pattern 3

Reviewer: Guilherme Santana

```
public interface SceneBuilder {  
    /**  
     * Resets this builder and sets scene height, which is a height of a user-visible viewport  
     * of a chart  
     *  
     * @param sceneHeight  
     */  
    void reset(int sceneHeight);  
  
    /**  
     * Builds a scene  
     */  
    void build();  
  
    /**  
     * @return canvas instance  
     */  
    Canvas getCanvas();  
}  
  
    (...)  
public abstract class AbstractSceneBuilder implements SceneBuilder  
    (...)  
public class DayGridSceneBuilder extends AbstractSceneBuilder  
    (...)  
public class BottomUnitSceneBuilder extends AbstractSceneBuilder  
    (...)  
(Among other classes)
```

Location on the code base:

code/biz.ganttproject.core/src/main/java/biz/ganttproject/core/chart/scene/
code/biz.ganttproject.core/src/main/java/biz/ganttproject/core/chart/scene/SceneBuilder.java
code/biz.ganttproject.core/src/main/java/biz/ganttproject/core/chart/scene/AbstractSceneBuilder.java
code/biz.ganttproject.core/src/main/java/biz/ganttproject/core/chart/scene/DayGridSceneBuilder.java
code/biz.ganttproject.core/src/main/java/biz/ganttproject/core/chart/scene/BottomUnitSceneBuilder.java

Pattern identification:

Template Method – The SceneBuilder interface along with the abstract class that implements it, AbstractSceneBuilder class, works as the “recipe” for building a scene, that is, they are a template for classes like DayGridSceneBuilder and BottomUnitSceneBuilder which can adapt this construction and algorithm to build a specific genre of scene, in this case a DayGrindScene and a BottomUnitScene. Note that there are still more classes in the "scene" folder that extend AbstractSceneBuilder to create other scene variations through this template.

13. Gonalo Gomes's Design Pattern 1

Reviewer: Brbara Correia

```
package biz.ganttproject.core.time;

import java.text.DateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.Locale;

public abstract class CalendarFactory {
    public static interface LocaleApi {
        Locale getLocale();
        DateFormat getShortDateFormat();
    }

    private static LocaleApi ourLocaleApi;

    public static Calendar newCalendar() {
        return (Calendar) Calendar.getInstance(ourLocaleApi.getLocale()).clone();
    }

    protected static void setLocaleApi(LocaleApi localeApi) {
        ourLocaleApi = localeApi;
    }

    public static GanttCalendar createGanttCalendar(Date date) {
        return new GanttCalendar(date, ourLocaleApi);
    }

    public static GanttCalendar createGanttCalendar(int year, int month, int date) {
        return new GanttCalendar(year, month, date, ourLocaleApi);
    }

    public static GanttCalendar createGanttCalendar() {
        return new GanttCalendar(ourLocaleApi);
    }
}
```

Location on the code base:

biz.ganttproject.core\src\main\java\biz\ganttproject\core\time\CalendarFactory.java

Pattern identification:

Object factory design pattern – The class CalendarFactory implements methods used for the instantiation of other classes. By doing this, it hides certain aspects of the creation of the classes from the places that use them and it also makes the instantiation of these classes independant from those same users.

14. Gonalo Gomes's Design Pattern 2

Reviewer: Joana Cruz

```
package biz.ganttproject.core.time;

/**
 * Created by IntelliJ IDEA.
 *
 * @author bard Date: 31.01.2004
 */
public interface TimeUnit extends DateFrameable {
    public String getName();

    public boolean isConstructedFrom(TimeUnit unit);

    /**
     * @return number of atoms used to create current TimeUnit
     * @throws UnsupportedOperationException
     *         if current TimeUnit does not have constant number of atoms
     */
    public int getAtomCount(TimeUnit atomUnit);

    /** @return the TimeUnit which is used to build the current TimeUnit */
    public TimeUnit getDirectAtomUnit();

    public int DAY = 0;
}
```

Location:

biz.ganttproject.core\src\main\java\biz\ganttpproject\core\TimeUnit.java

biz.ganttproject.core\src\main\java\biz\ganttpproject\core\TimeUnitImpl.java

Pattern identification:

Template design pattern – The TimeUnit interface and the respective implementation of TimeUnitImpl are a template of a time unit that is redefined in several other classes

as needed

15. Gonalo Gomes's Design Pattern 3

Reviewer: Guilherme Santana

```
public class FramerImpl implements DateFrameable {
    private final int myCalendarField;

    public FramerImpl(int calendarField) {
        myCalendarField = calendarField;
    }

    @Override
    public Date adjustRight(Date baseDate) {
        Calendar c = CalendarFactory.newCalendar();
        c.setTime(baseDate);
        clearFields(c);
        c.add(myCalendarField, 1);
        return c.getTime();
    }

    private void clearFields(Calendar c) {
        for (int i = myCalendarField + 1; i <= Calendar.MILLISECOND; i++) {
            c.clear(i);
        }
    }

    @Override
    public Date adjustLeft(Date baseDate) {
        Calendar c = CalendarFactory.newCalendar();
        c.setTime(baseDate);
        clearFields(c);
        // Date beforeClear = c.getTime();
        // if (beforeClear.compareTo(c.getTime())==0) {
        // c.add(Calendar.MILLISECOND, -1);
        // }
        return c.getTime();
    }

    @Override
    public Date jumpLeft(Date baseDate) {
        Calendar c = CalendarFactory.newCalendar();
        c.setTime(baseDate);
        c.add(myCalendarField, -1);
    }
}
```

```
return c.getTime();
```

Location:

biz.ganttproject.core\src\main\java\biz\ganttpproject\core\time\impl\FramerImpl.java

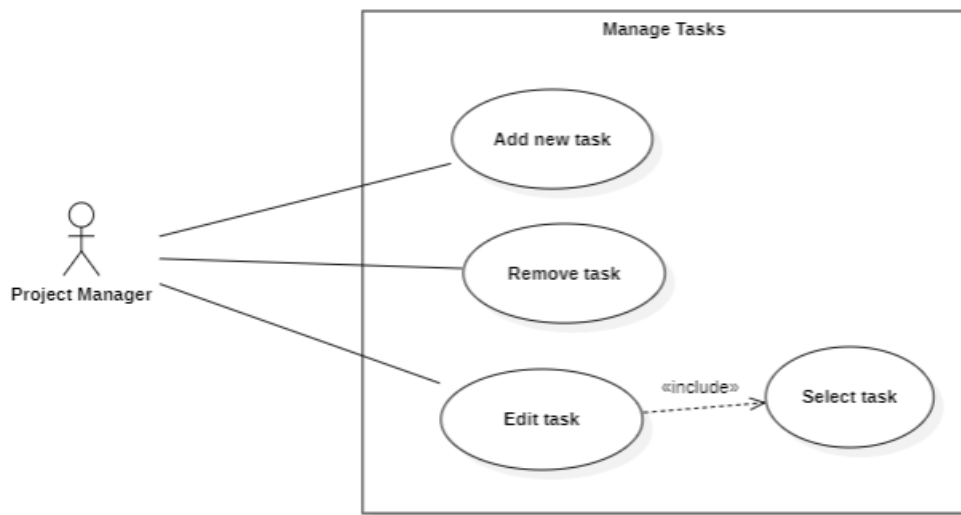
Pattern identification:

Facade design pattern – The class FramerImpl has as its main function to redefine, alter or otherwise modify other classes

D. Use Case Diagrams

1. Bárbara Correia's Use Case

Reviewer: Joana Cruz



Use Case: **Add new task**

Description: The Project Manager creates a new task adding it to the app.

Main actor: Project Manager

Secondary actors: None

Pre-conditions: The name used for the created task does not match any of the names from the already existing tasks.

Post-conditions: The task is now added. The task is now in the app and can be managed.

Use Case: **Removes a task**

Description: The Project Manager selects a task and removes it.

Main actor: Project Manager

Secondary actors: None

Pre-conditions: The selected task exists.

Post-conditions: The task is no longer on the app.

Use Case: **Select a task**

Description: The Project Manager selects a task.

Main actor: Project Manager

Secondary actors: None

Pre-conditions: The selected task exists.

Post-conditions: The task is now selected.

Use Case: **Edit a task**

Description: The Project Manager selects a task and edits it.

Main actor: Project Manager

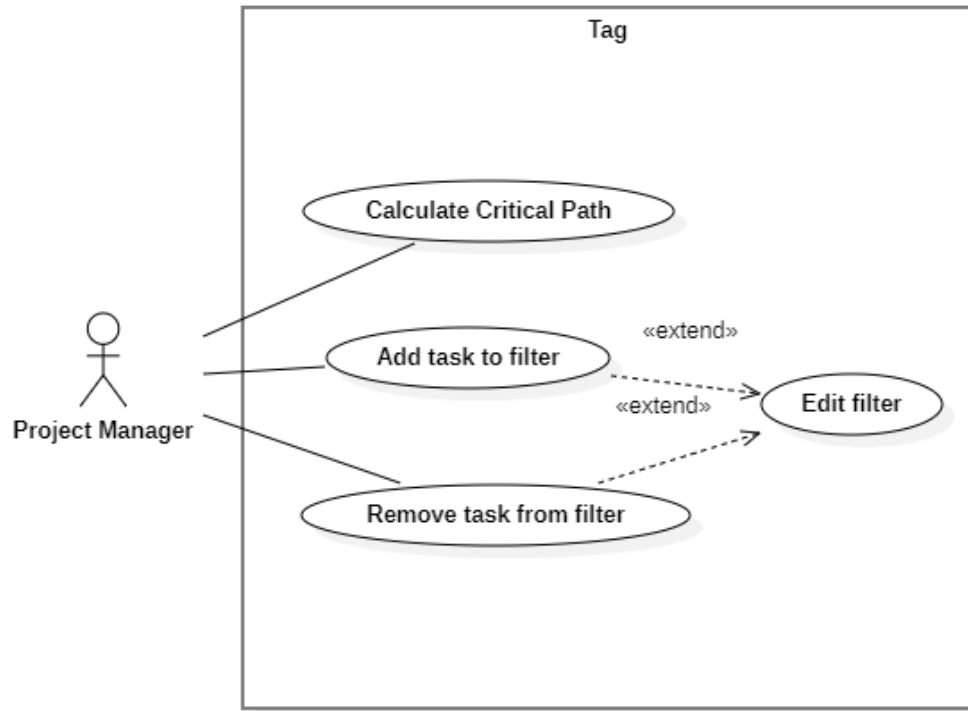
Secondary actors: None

Pre-conditions: The selected task exists.

Post-conditions: The task is now edited.

2. Gonalo Rodrigues's Use Cases

Reviewer: Gonalo Gomes



Use Case: **Add task to filter**

Description: The Project Manager selects a task and adds it to the filter

Main actor: Project Manager

Secondary actors: None

Pre-conditions: The selected task exists and it was not added to the filter.

Post-conditions: The task is now shown when the filter is selected.

Use Case: **Remove a task from the filter.**

Description: The Project Manager selects a task and removes it from the filter.

Main actor: Project Manager

Secondary actors: None

Pre-conditions: The selected task exists and it's on the filter's task list.

Post-conditions: The task doesn't belong to the filter's task list and will not be shown when the filter is selected.

Use Case: **Edit filter**

Description: The Project Manager edit a selected task

Main actor: Project Manager

Secondary actors: None

Pre-conditions: An filter exists

Post-conditions: The filter may be changed or not.

Use Case: **Calculate Critical Path**

Description: The Project Manager calculates the critical path

Main actor: Project Manager

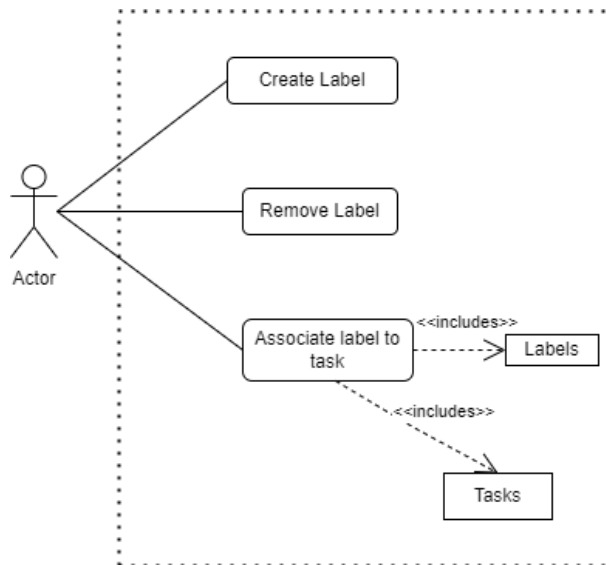
Secondary actors: None

Pre-conditions: At least one task exists

Post-conditions: None

3. Guilherme Santana's Use Case

Reviewer: Gonalo Rodrigues



Use Case: **Create Filter/Label**

ID: 1

Description: Creates a new label that could be associated to a task

Main actor: User

Secondary actor: None

Pre-conditions: None

"Main flow:

1. The case starts when the user select the option "Nova etiqueta" to create a filter/label
2. While the Label/filter name is invalid
 - 2.1 The system asks the user the name of the new label/Filter to associate to the tasks
3. The System creates the new label/filter"

Post-conditions: A new label has been created for the Customer

Alternative flow: InvalidNameFilter

Alternative flow: **Create Filter/Label: InvalidNameFilter**

ID: 1.1

Description: The system informs the User that he or she has entered an invalid name to the label/filter

Main actor: User

Secondary actor: None

Pre-conditions: The User has entered an invalid name Label

"Alternative flow:

1. The alternative flow begins after the step 2.1 of the main flow
2. The system informs the User that he or she entered an invalid Label/Filter name"

Post-conditions: None

Use Case: **Remove Filter/Label**

ID: 1

Description: Removes a existent label that could be associated to a task

Main actor: User

Secondary actor: None

Pre-conditions: The label to be removed need to exists in the system

"Main flow:

The case starts when the user select the option "Apagar etiqueta" to remove a filter/label

The user selects the name of the label/Filter to remove

For each task, the system checks if has the label to be removed

If the label to remove is associated to some tasks

the system removes that filter/label to that task

Else

checks out the next task

The System removes the new label/filter from the system"

Post-conditions: The specified filter/label has removed from the system

Alternative flow: None

Use Case: **Associate Filter/Label to a task**

ID: 1

Description: Associates a existent label to an new or existent task

Main actor: User

Secondary actor: None

Pre-conditions: The label and the associated task need to exist in the system

"Main flow:

The case starts when the user selects the option "Tarefas por etiqueta" to associate a label/filter to a task.

The user selects a label and a task to associate to

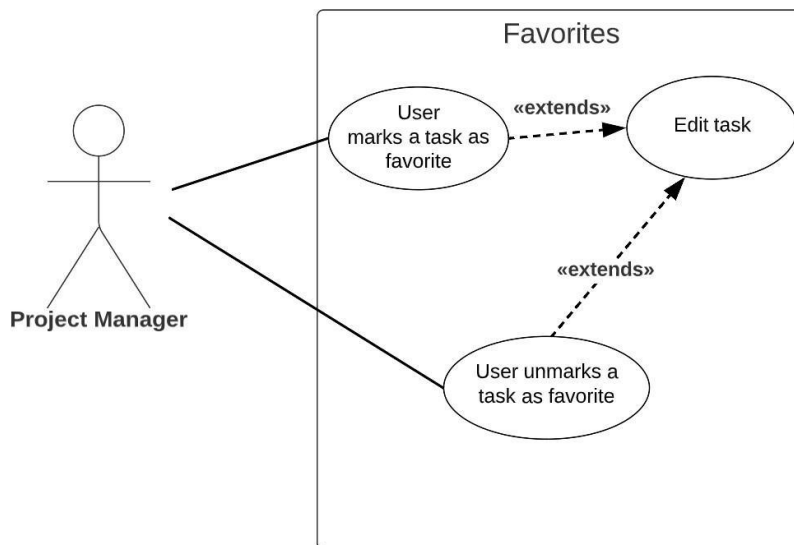
The specified task has a new label associated"

Post-conditions: the specified task has a new filter/label

Alternative flow: None

4. Joana Cruz's Use Case

Reviewer: Guilherme Santana



Use Case: **Project Manager marks a task as favorite**

Description: The Project Manager selects a task and marks it as favorite.

Main actor: Project Manager

Secondary actors: None

Pre-conditions: The selected task exists, and it's not marked as favorite.

Post-conditions: The task is now marked as favorite. The task is now in the favorites tab.

Use Case: **Project Manager unmarks a task as favorite**

Description: The Project Manager selects a task and unmarks it as favorite.

Main actor: Project Manager

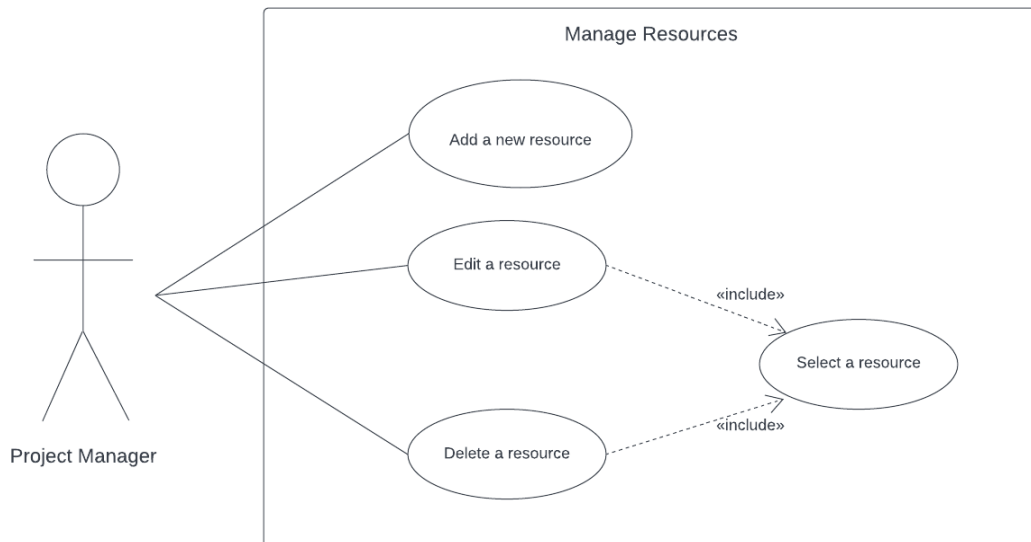
Secondary actors: None

Pre-conditions: The selected task exists; it's marked as favorite and it's in the favorites tab.

Post-conditions: The task is no longer marked as favorite, nor shown in the favorites tab.

5. Gonalo Gomes's Use Case

Reviewer: Brbara Correia



Use Case: **Add a new resource**

Description: The Project Manager creates a new resource adding it to the app.

Main actor: Project Manager

Secondary actors: None

Pre-conditions: None

Post-conditions: The resource is now added. The resource is now in the app and can be managed.

Use Case: **Select a resource**

Description: The Project Manager selects a resource to edit it.

Main actor: Project Manager

Secondary actors: None

Pre-conditions: The selected resource exists.

Post-conditions: The resource is now selected.

Use Case: **Delete a resource**

Description: The Project Manager removes a selected resource.

Main actor: Project Manager

Secondary actors: None

Pre-conditions: The resource exists and is selected.

Post-conditions: The resource is no longer on the app.

Use Case: **Edit a resource**

Description: The Project Manager edits a selected resource.

Main actor: Project Manager

Secondary actors: None

Pre-conditions: The resource exists and is selected.

Post-conditions: The resource is now edited.

E. Metrics Analysis

1. Lines of code

Author: Gonalo Rodrigues

Reviewer: Guilherme Santana

In this analysis, I used the IntelliJ plugin "MetricsReloaded" and the metrics were calculated with a 'Scope'= Whole Project and 'Metrics Profile' = Lines of code metrics (system), on the "favManager" branch.

The analysis will be done by types (methods, classes, interfaces, packages, modules and projects at last).

The lines of code considered are lines with comments, javadoc and code.

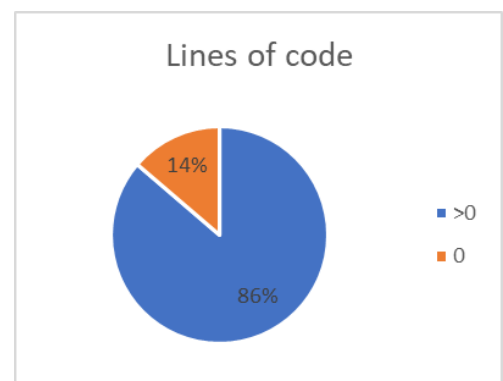
The number of lines helps in finding problems such as long or short comments and long methods/classes/interfaces that can be divided into many small ones but alone may leave unchecked code smells or bad patterns such as data classes, duplicated code, dead code and many others.

a. Methods

In the project, there are 7204 methods and with this amount of methods comes a large amount of bad habits and code.

Among these 7204 methods, 989 have 0 lines of code. This code is useless at the moment.

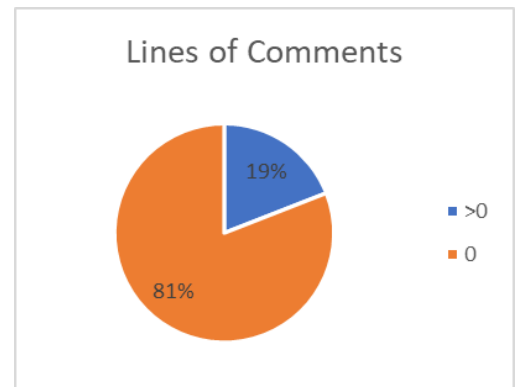
To improve the project quality, I recommend deleting these useless methods or inserting new and useful code. Taking into account the general context of the project, 14% of the methods are empty which represents a small but important amount of methods which need refactoring.



In the project, there are 6267 methods without comments which represents a critical flaw. This number means that 81% of the project's methods are not documented at all and makes understanding the project's files very difficult or even impossible, a problem that our group faced and made our work very difficult.

Documentation is very important to make a project accessible and easy to understand. The GanttProject is an open-source project and the accessibility of the code is an essential part of the puzzle.

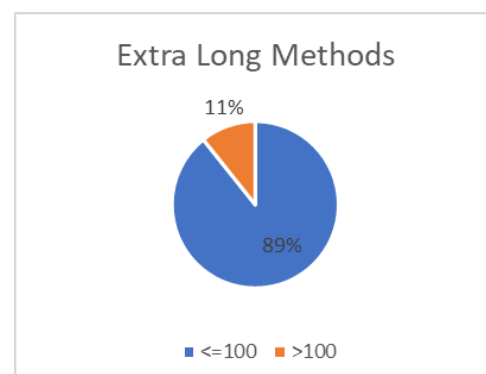
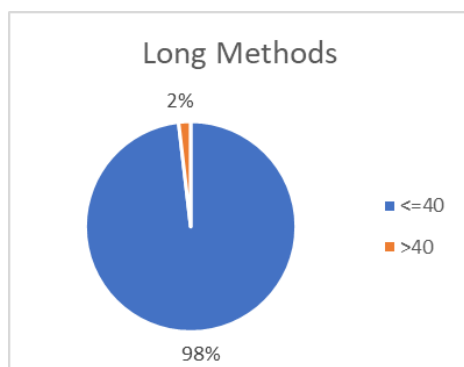
To resolve the issue, I recommend adding comments to the majority of the methods, if not all of them.



Another point evaluated with this metric was the long methods. The convention is that methods have up to 40 lines of code. From this number, most of the methods can be divided into several other methods that should be commented separately, also helping in the improvement of the accessibility of the code for everyone who reads it, a problem already addressed previously.

In this project, 133 of the methods have more than 40 lines of code which are only 2% of the project's methods but among these 133, 2% of these have more than 100 lines of code which can most certainly be divided into several smaller and understandable methods.

I recommend, as stated above, that these methods be split into several others in order to make the code more readable and accessible, with some urgency for methods with more than 100 lines of code. While splitting the methods, we must have caution not to put the objective and understanding of the code at risk. Sometimes a long method can be preferred over a collection of small methods in order to make it easier to understand .

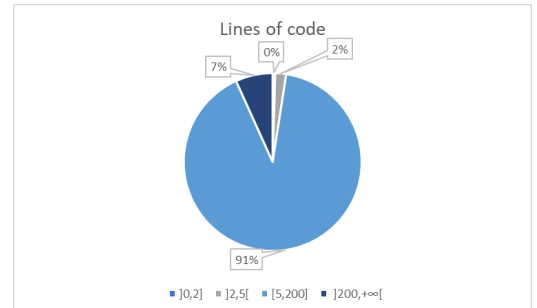


b. Classes

Among many conventions, many define the class' maximum number of lines of code as 200.

In this project there are 859 classes, only 4 have no code, 17 are enums or classes with a single very simple method and 58 with more than 200 lines of code.

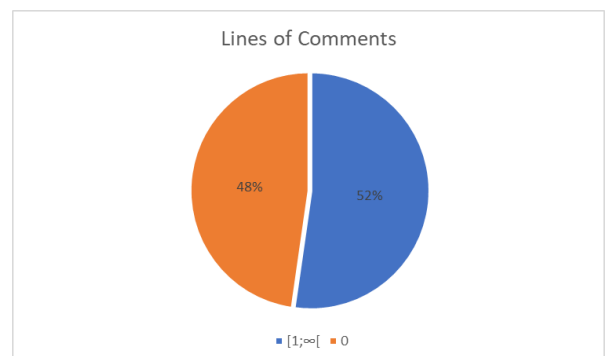
The classes with a lot of lines of code, with a lot of methods or extensive ones, have code that is harder to understand and a limited accessibility.



To resolve the problem of long classes, I recommend shortening the number of lines of code of the classes' methods and dividing the classes into smaller ones. While dividing, we must have caution not to interfere with the objective of the code, the same reason that we have to be cautious when splitting long methods into smaller ones.

As said before, comments are important and crucial on classes and methods in order to provide the most clarity and accessibility to everyone who reads the code.

Therefore I recommend implementing comments on the classes, even though most of the methods' comments are written on the classe's interfaces. The amount of comments must not be very extensive because long comments may have complex explanations that can be resumed into smaller ones.



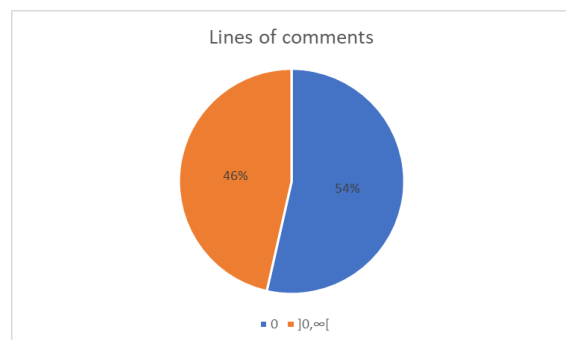
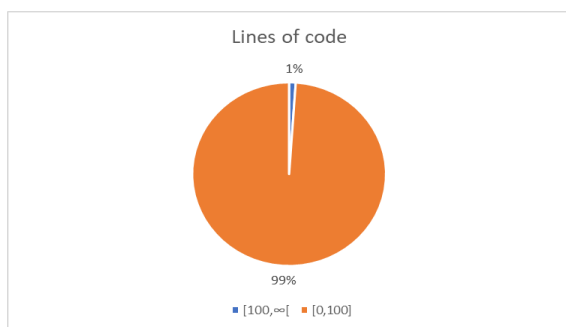
c. Interfaces

In this project there are 183 interfaces, but 98 are not commented. On the other hand, in general they are not very extensive since only 2 have more than 100 lines of code.

The interfaces are an important part of the documentation of every project and should have all the methods commented in order to provide clarifications that may arise while implementing the interface in a class or using a method of a class that implements that interface.

Interfaces should have the comments of the methods in order to ease the reading and understanding but should not be extensive and long.

To improve the project, I recommend that comments are added to the not commented interfaces and that interfaces with lots of lines of code are split into simpler ones, if possible, and extending a main interface.



2. Dependency

Author: Bárbara Correia

Reviewer: Gonçalo Gomes

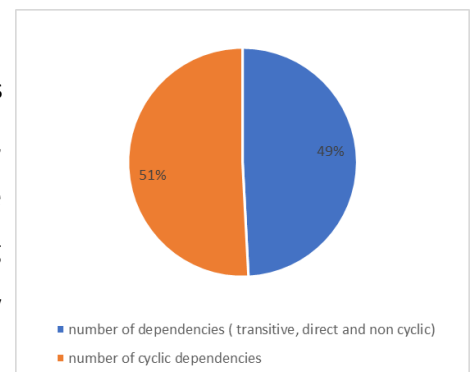
Before analyzing the data retrieved from IntelliJ with the dependency metrics, I will start by clarifying some concepts.

First of all, transitive dependency. This form of dependency manifests itself if A depends on B and B depends on C and therefore A depends on both B and C (indirect).

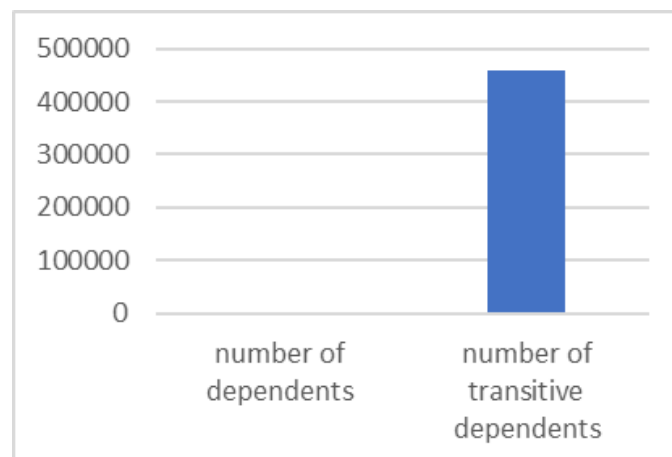
Another important concept to take into account here are cyclic dependencies. A cyclic dependency occurs when two or more architecture components have direct or indirect dependencies on each other and it is a code smell.

a. Classes

In this project there are 241 559 cyclic class dependencies out of 475 361 both transitive and direct dependencies. With this, we know that 51% of transitive dependencies between classes are cyclic. Because this is a code smell, this number is very upsetting as it shows that in regards to dependency, this project is very “smelly”.



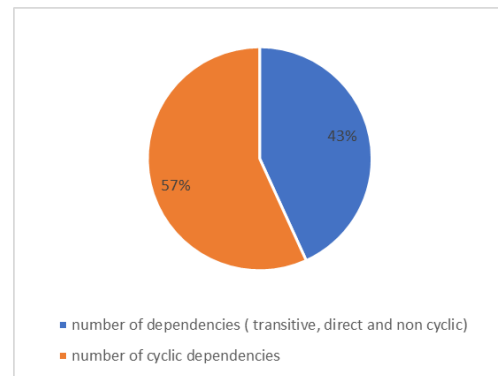
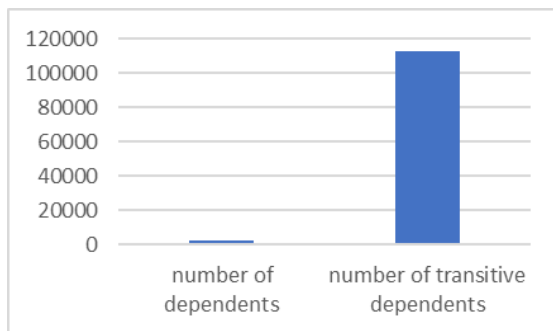
The number of dependents, in turn, is far less high than the number of transitive dependents (3569 to 459356), as shown in the bar diagram below.



b. Interfaces

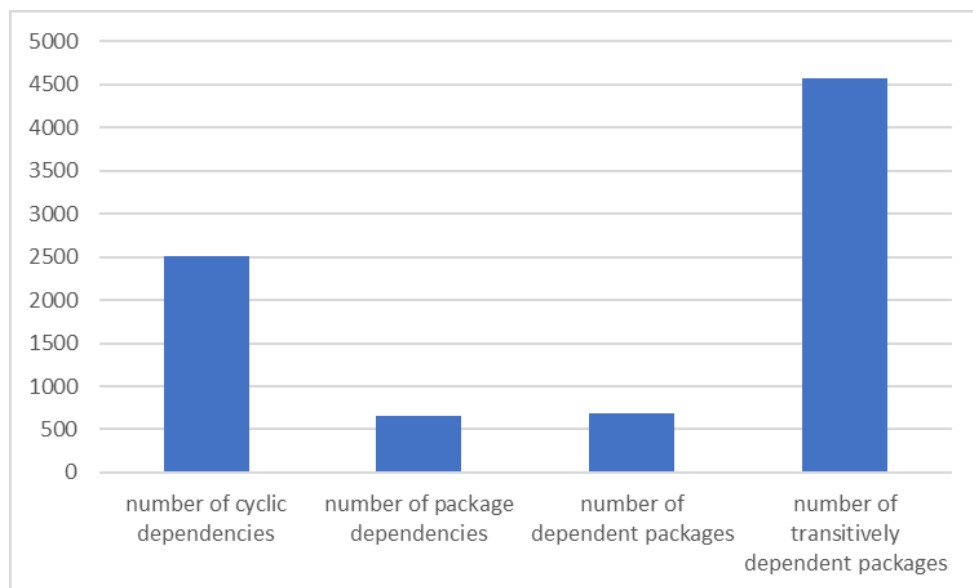
In this project there are 35 785 cyclic interface dependencies out of 63 002 both transitive and direct dependencies. With this, we know that 57% of transitive dependencies between interfaces are cyclic. Because this is a code smell, this number is very upsetting as it once again shows that in regards to dependency, this project is very “smelly”.

The number of dependents, in turn, is far less high than the number of transitive dependents (2674 to 112776), as shown in the bar diagram below.



c. Packages

This project has 2510 cyclic package dependencies, 652 package dependencies, 689 dependent packages and 4576 transitively dependent packages.



d. Conclusion

When dealing with dependencies we should be careful not to originate cyclic dependencies, therefore creating code smells. The most problematic consequence of this kind of dependency from a software design point of view is the tight coupling of the mutually dependent components, which makes it nearly impossible to re-use a single component separately. Another problem is the domino effect that can be caused when a change is made in one component and it unintentionally spreads into other components, producing unwanted changes globally. There can also be security problems and package corruption. Something to add is that it obviously presents a much more complex architectural complexity which can difficult the understanding of the code from an outsider perspective.

Overall, with my analysis of the dependency metrics of this project, it can be concluded that there are concerning amounts of transitive and cyclic dependencies, which evolve in the problems explained above.

A way to avoid these code smells is to apply design patterns like the observer pattern.

3. Complexity Metrics

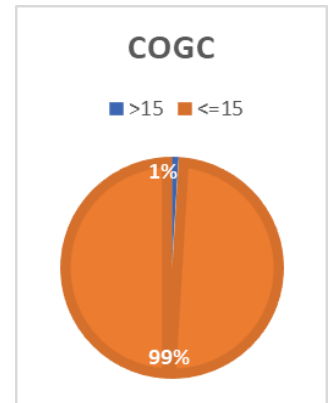
Author: Guilherme Santana

Reviewer: Gonalo Rodrigues

a. Cognitive complexity – CogC

Is a measure of how difficult a unit of code is to intuitively understand. Unlike Cyclomatic Complexity, which determines how difficult your code will be to test, Cognitive Complexity tells you how difficult your code will be to read and understand.

The standard value is below 15. As we can see 1% of the project is over the standard value. That means that from 7367, 70 are above the indicated value.



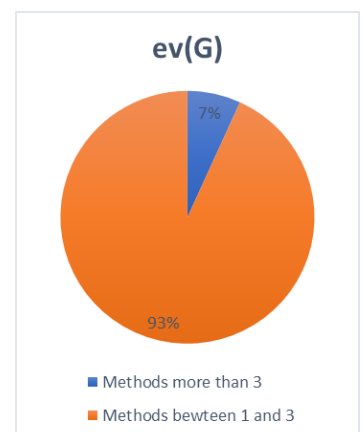
Top 5 with the most cognitive complexity - This methods have the code smell Long Method making them hard to understand.

Method	CogC
net.sourceforge.ganttproject.GanttOptions.GanttXMLOptionsParser.startElement(String, String, Stri	183
net.sourceforge.ganttproject.task.TaskProperties.getProperty(Task, String)	58
biz.ganttproject.core.time.impl.GPTimeUnitStack.parseDuration(String)	53
net.sourceforge.ganttproject.task.TaskManagerImpl.createLength(String)	53
org.w3c.util.DateParser.getCalendar(String)	49

b. Essential cyclomatic complexity – ev(G)

Essential cyclomatic complexity tells how much complexity is left once we have removed the well-structured complexity.

The standard value is between 1 and 3. As we can see 7% of the project is over the standard value. That means that from 6767 methods, 460 are above the indicated value.



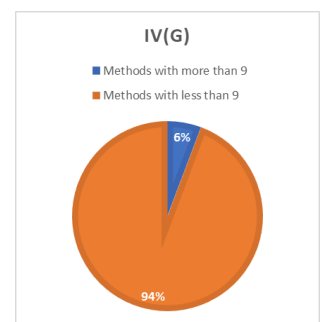
Top 5 with the most Essential Cyclomatic complexity - Methods with this complexity have the code smell Long Method, several big conditional switches, if's conditions or try methods (all of that is a code smell too)

method	ev(G)
org.w3c.util.DateParser.getCalendar(String)	16
biz.ganttproject.core.time.impl.GPTimeUnitStack.parseDuration(String)	12
net.sourceforge.ganttproject.task.TaskManagerImpl.createLength(String)	12
org.ganttproject.impex.htmlpdf.fonts.TTFontCache.FontKey.equals(Object)	12
net.sourceforge.ganttproject.task.TaskManagerImpl.FacadeImpl.compareDocumentOrd	11

c. Design Complexity – iv(G)

It is the measure that calculates the complexity of the design and a section of code, it is related to the interconnection between the "control flow" of methods and calls to other methods.

The standard value is between 1 and 9. As we can see 6% of the project is over the standard value. That means that from 6527 methods, 396 are above the indicated value.



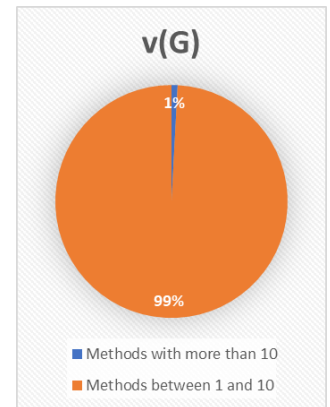
Top 5 with the most Design Complexity - Methods with this complexity have the code smell Long Method, and too many calls to other methods

method	iv(G)
net.sourceforge.ganttproject.GanttOptions.GanttXMLOptionsParser.startElement(String,	78
net.sourceforge.ganttproject.gui.GanttTaskPropertiesBean.applySettings()	28
net.sourceforge.ganttproject.task.TaskManagerImpl.newTaskBuilder()	22
net.sourceforge.ganttproject.io.TaskSaver.writeTask(TransformerHandler, GanttTask, Cu	21
net.sourceforge.ganttproject.parser.TaskTagHandler.loadTask(Attributes)	21











d. Cyclomatic Complexity – v(G)

The V(g) or cyclomatic number is a measure of the complexity of a function which is correlated with difficulty in testing. The standard value is between 1 and 10. A value of 1 means the code has no branching. A function's cyclomatic complexity should not exceed 10.

The standard value is between 1 and 10. As we can see 1% of the project is over the standard value. That means that from 6078 methods, 52 are above the indicated value.



Top 5 with the most Cyclomatic Complexity - Methods with this complexity have the code smell Long Method and several conditional if's (that's a code smell too)

method	v(G)
  net.sourceforge.ganttproject.GanttOptions.GanttXMLOptionsParser.startElement(String,	89
  net.sourceforge.ganttproject.GanttTreeTableModel.getValueAt(Object, int)	28
  net.sourceforge.ganttproject.gui.GanttTaskPropertiesBean.applySettings()	28
  net.sourceforge.ganttproject.parser.TaskTagHandler.loadTask(Attributes)	25
  org.w3c.util.DateParser.getCalendar(String)	25

e. Conclusion

Any method with the code smells long method has at least one of the complexities above. As referenced above to solve this code smell the solution is to divide the method into auxiliary methods reducing its complexity and making it more readable.

4. Martin Packaging Metrics

Author: Joana Cruz

Reviewer: Bárbara Correia

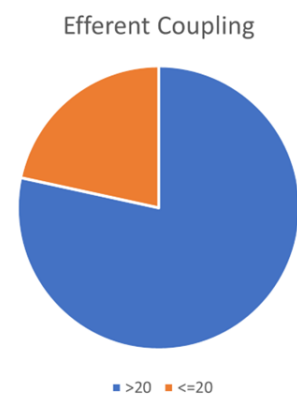
This group of object-oriented metrics was proposed in 1994 and remains popular until now, unlike other object-oriented metrics they don't represent the full set of attributes to assess individual design but instead focus on the relationship and dependency between the packages of the project. These metrics were taken from the test_label branch on commit 0d01f57.

a. Efferent coupling

This metric is defined as the number of classes in a given package which depends on the classes in other packages.

Efferent coupling values higher than 20 indicate instability of a package, and as we can see in graph N most of Gantt Project's packages, 69 in 88 to be more precise, fall on this category. Gantt project reveals serious issues in this matter, the average value for efferent coupling is around 236 which is very far away from the recommended 20 or lower.

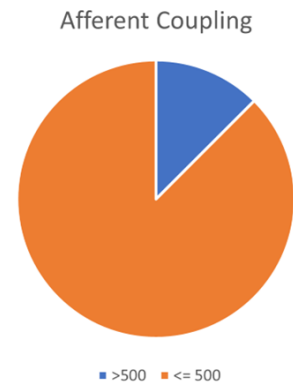
This causes problems with care and development of code. If we have a class in a package A from those 69 packages and alter something from which A depends on a package B we might end up causing unexpected conflicts in class A and have to make changes there too. Which if there's another package C that is highly dependent from A c may also need fixing.



b. Afferent coupling

The Afferent coupling metric complements the Efferent coupling one by analyzing incoming dependencies rather than outgoing ones, it is the number of classes in another package that depend on classes in the package we're in. In different words, this metrics measures the sensitivity of the other packages to changes in this one.

The preferred values for this metrics are between 0 and 500, as we can see (Graph N), most packages in this project respect this range, however there are still a few concerning ones, the `net.sourceforge.ganttproject.task` package has a value of Afferent Coupling of 4 609, and there's 4 more packages, like `biz.ganttproject.core.option` or `net.sourceforge.ganttproject` with a value higher than 1 000.



c. Abstractness

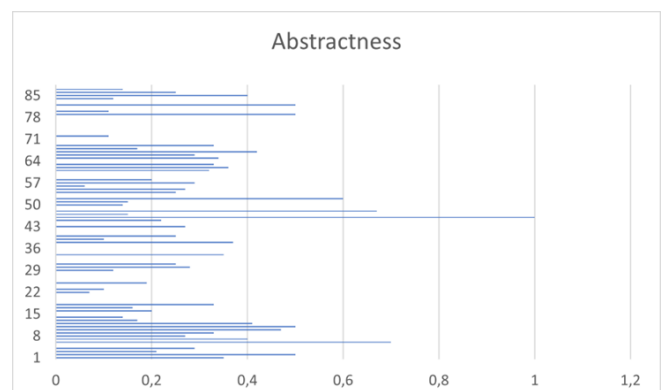
Abstractness is the number of abstract classes compared to the total number of classes in a package.

It is measured according to the formula:

$$A = \frac{T_{abstract}}{T_{abstract} + T_{concrete}}$$

Where $T_{abstract}$ is the number of abstract classes and $T_{concrete}$ the number of concrete ones.

The values of abstractness should have a value extremely close to 0 or extremely close to one. As we can observe there's a few packages with values close to 0 (there's 50 packages with values lower than 0,2), but only one class with a value close to 1, in this case the `net.sourceforge.ganttproject.gui.options.model` abstractness reaches 1.



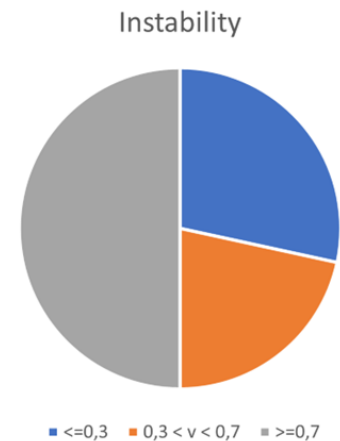
d. Instability

According to definition this metric is the ratio of outgoing dependencies to all package dependencies and its values go from 0 to 1.

It's calculated with the formula:
$$I = \frac{Ce}{Ce + Ca}$$

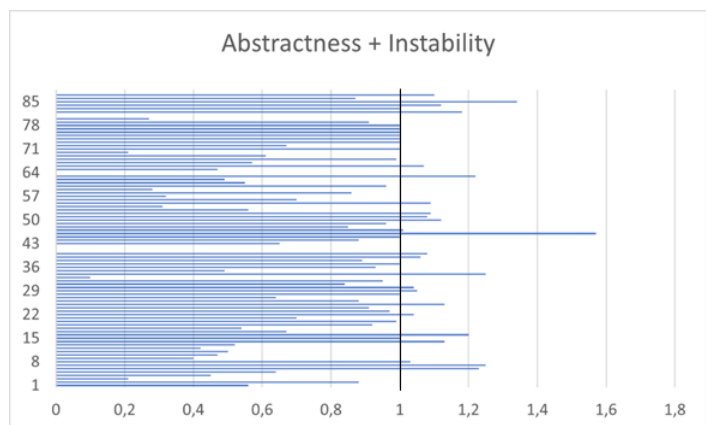
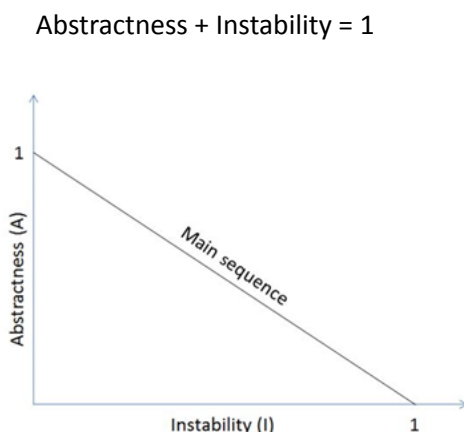
Where *Ce* is *Efferent Coupling* and *Ca* is *Afferent Coupling*.

It is considered that packages should be either very unstable (values from 0,7 to 1) or very stable (0 to 0,3). From the graph on the right, we can see that half the packages of the project are very unstable which is quite concerning although it could or not be compensated by its Abstractness as we'll see right after. There's also more than a quarter of very stable packages (25 from 88).



e. Abstractness and Instability are closely related

This two metrics should sum 1 in the optimal case, as a package that is stable (Instability close to 0) should also be abstract (Abstractness close to 1) because it is dependent on a very low level, whereas packages that are very unstable (Instability close to 1) should have mostly concrete classes (Abstractness close to 1).



As we can see Gantt Project's Abstractness and Instability do not complement each other in most cases, ruffly evaluating Graph N about half the packages have the sum quite close to one but the other half has lower values, which might

mean that there are packages with mostly abstract classes that may be unstable and packages that are very stable but have fewer concrete than abstract classes.

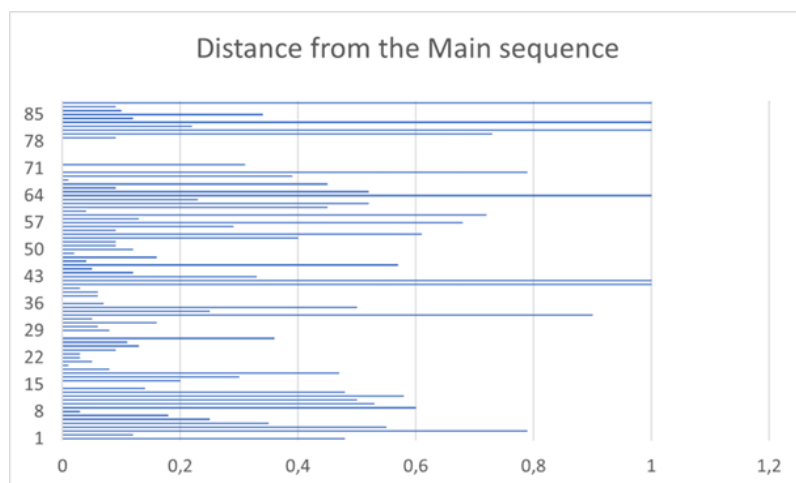
f. Distance from the Main sequence

This metric calls into action the fact that Abstractness and Instability are highly related and measures the balance between stability and abstractness.

It's measured with the formula: $D = |A + I - 1|$

And D may be interpreted in the following way; if we put a given class on a graph of the main sequence its distance from the main sequence will be proportional to the value of D.

This value should be as low as possible so that the components are all quite close to the main sequence.



As we can see this project has some packages with a lower value(35 packages with a value lower than 0,1) however there are some packages(6 with a value higher than 0,9) that have an extremely high value or where it's value it's even 1. These packages might be implementing functionalities that are not directly necessary to the Main sequence of events.

5. Chidamber and Kemerer metrics

Author: Gonalo Gomes

Reviewer: Guilherme Santana

a. WMC - Weighted Methods per Class

This metric is very self explanatory, the number of methods per class. A high number of methods grouped in one class can be a good indicator of complexity, and a high number also makes the class less likely to be reusable in other places, as it is likely very specific to the application it was designed for.

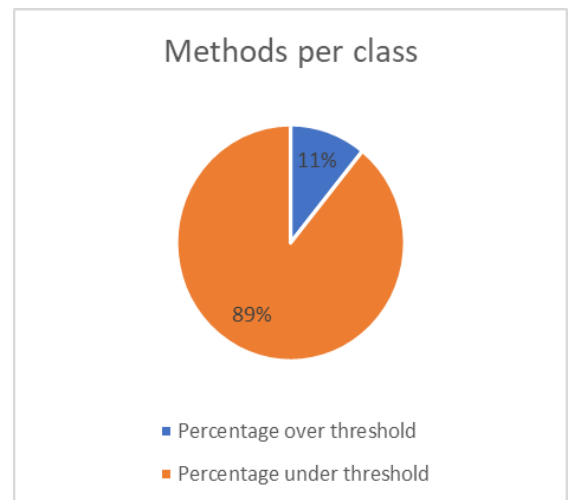
However, many classes actually require a large number of methods to function properly and as such, it is best to evaluate this metric in a percentage of classes with a number of methods above a certain threshold.

In this case, the threshold used was 24 methods per class.

As we can see, the percentage of classes with more methods than recommended sits at 11%, just 1% above the advised 10%.

Given the complexity and vast quantity of functions this program implements, it is not surprising to go over the recommended percentage, and, as such it should not be counted as an error just by itself.

That being said, the classes themselves can be rather complex, even the smaller ones and having so many of them be extremely large (quite a few of them with over 50 implemented methods and some going even over 100) it can be even harder to maintain, expand and improve the code.



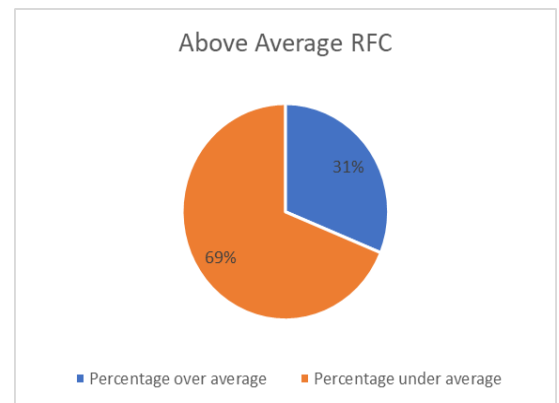
b. RFC Metrics

Author: Gonalo Rodrigues

The “Response for class” is defined by the sum of the number of methods and constructors in the class plus the number of methods and constructors that class may directly call.

Classes with high RFC value have a higher complexity, require higher amounts of integration testing and are less stable.

To lower the RFC of the classes, I recommend that we lower the dependencies of the classes, change the access modifier of variables and methods from public to private or protected.

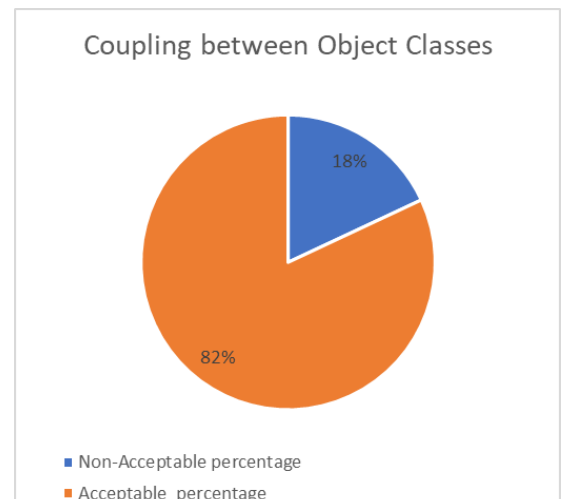


c. CBO - Coupling between Object Classes

Class coupling is a measure defined by the functions of a class that are used by other external classes. This measure helps detect errors like excessive modularity and reuse, meaning classes work better independently from each other and are easier to maintain because of that.

A class starts to be “too coupled” when the CBO rating is above 14.

As we can see, the whole project has almost 20% of its classes above the recommended coupling measure, which is likely a major factor in its extreme complexity and amount of bugs during the expansion.



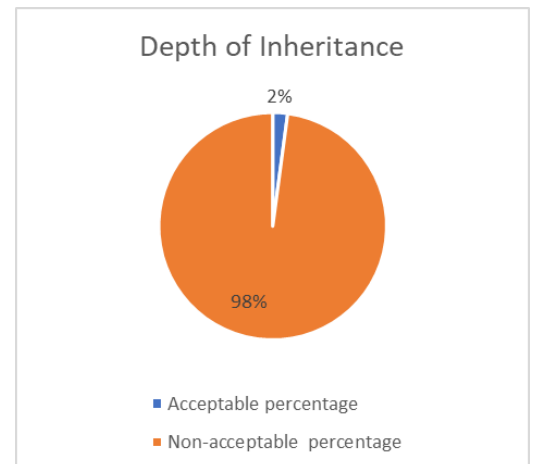
The appearance of bugs was common, as the methods used in one place are likely to also be used elsewhere and the behavior affects more aspects of the program than intended.

A good solution would be to separate the bigger classes into smaller ones or redistribute their methods across the rest of the intervening classes.

d. DIT - Depth Inheritance Metrics

The depth of inheritance metric aims to provide a quantitative way. The deeper a class is in an inheritance tree, the more problems it could cause.

While this project has very few classes that surpass the recommended depth level of 5, the complexity and ambitious goals of the project itself could be to blame. Additionally, all the classes that surpass the limit, do so by small amounts, again, something not entirely unexpected considering the circumstances.



e. NOC - Number of Children

The number of children metric is used to measure the reuse of classes as superclasses of others, being quantified by the number of immediate children of the class in study.

The higher the number the better, given the reuse of code reduces conflicts and takes advantage of all the modularity implemented in the code.

This project has a very low NOC metric overall, the average being 0.45, while the highest being 87, which, while impressive, is also the outlier, given that most classes have 0 children, thus the low average.

This means that the code is very expansive and minor alteration/fixes could require intervention across several classes.

This could be resolved by grouping some of the smaller classes together where it made sense, in order to reduce the spread of the methods, but it should be done carefully in order not to negatively affect the quality of the code and make long methods/classes and increase the coupling between objects.

F. Demo Video

This is the Youtube link of the demo of our demo video:

<https://youtu.be/OlyMLfoFPH8>

G. Conclusion

Throughout the realization of this project, contents related to project management and software quality taught in the lectures were applied to a real project - the GanttProject. We also got to understand more in depth the value of producing good quality code as we experienced difficulties in terms of clarity and accessibility due to various bad practices present in the software (code smells). The group faced several obstacles such as git, the lack of experience when using gradle and the complicated task of inserting good quality code into a project with the before referenced flaws.