# Software Engineering 2
# SUPD REPORT

## Status Update Report

| | |
|---|---|
| **Team number:** | 0309 |

| **Team member 1** | |
|---|---|
| **Name:** | Muhammed Akinci |
| **Student ID:** | 11720479 |
| **E-mail address:** | a11720479@unet.univie.ac.at |

| **Team member 2** | |
|---|---|
| **Name:** | Maxim Bogoutdinov |
| **Student ID:** | 01468382 |
| **E-mail address:** | a01468382@unet.univie.ac.at |

| **Team member 3** | |
|---|---|
| **Name:** | Ema Dupovac-Kilincarslan |
| **Student ID:** | 01268309 |
| **E-mail address:** | a01268309@unet.univie.ac.at |

| **Team member 4** | |
|---|---|
| **Name:** | Lala Mammadova |
| **Student ID:** | 01652485 |
| **E-mail address:** | a01652485@unet.univie.ac.at |

# 1 Design Draft

## 1.1 Design Approach and Overview

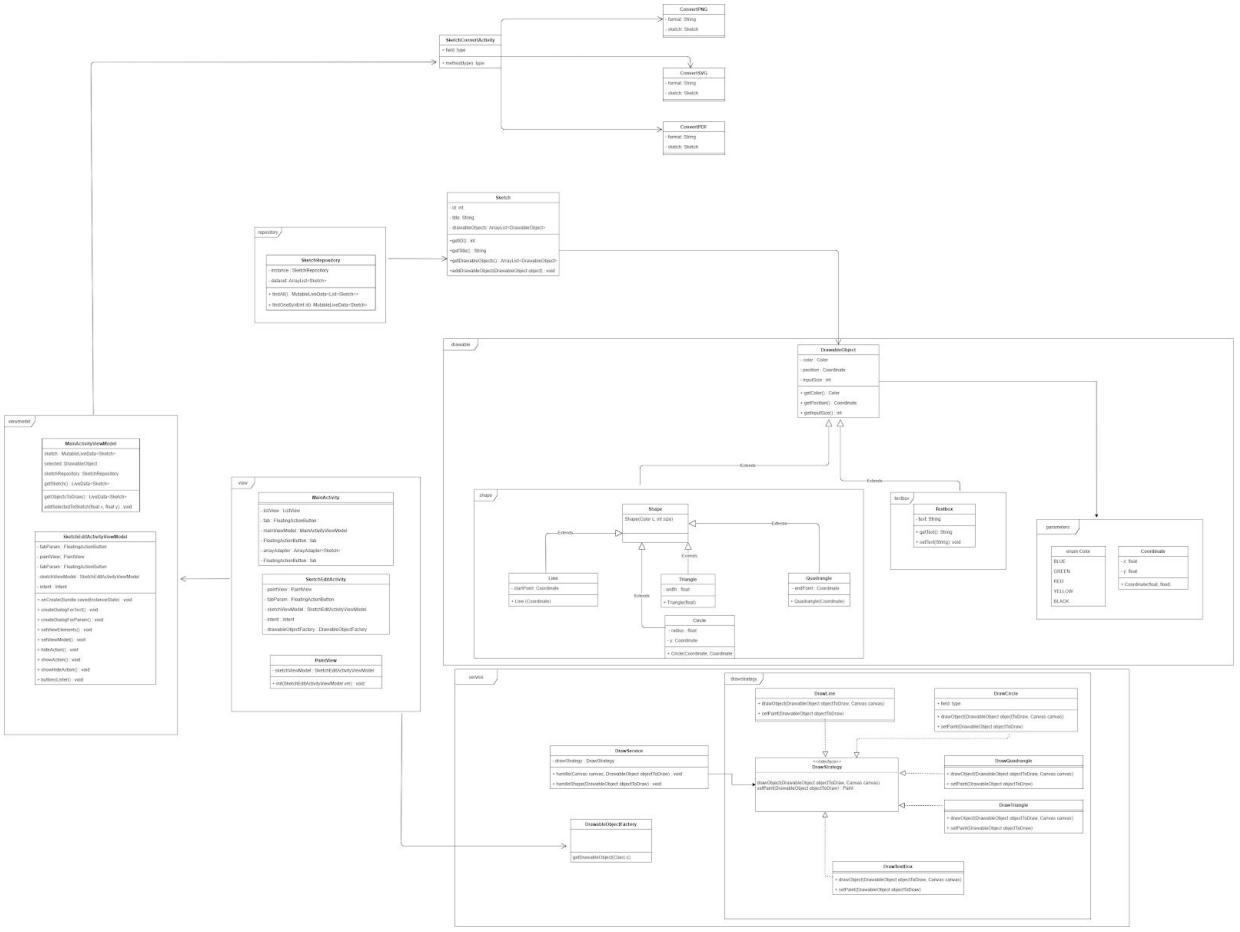
In our first online meeting we discussed the assignment sheet and talked about possible solutions for the project. It was a long discussion about the assignment sheet and understanding the requirements set there. So we used brainstorming as a technique to collect different ideas about functionalities and layout implementations that had to be done.

By next time we had a first version of the UML class diagram, which contained the basic logic and elements we had to implement. In the first iteration we just did the basic skeleton of our implementation where we defined classes with basic shapes. At this stage we decided to start coding and expand our project by actually working on it. So we divided the shapes that had to be implemented and started working on them. We always had regular meetings where we discussed our ideas and merged them. And we were parallel also working on our UML class diagram and expanding them.

### 1.1.1 Class Diagrams

Our class diagram contains the classes of our current implementation and the parts of classes for our planned implementation for the DEAD. The snippets for the chosen patterns Strategy pattern and Factory Method Pattern is also described in this document. As was discussed with the Team members, for the project was chosen to use Model-View-ViewModel Architectural pattern. The Project contains as it can be seen from the Class Diagram separate packages such as view, model and viewmodel.

The architecture was implemented in a way, where view is not dependent on any specific model which means the viewmodel is responsible for exposing the data objects from the model so that objects are easily managed and presented.



The Class Diagram is also added to Gitlab as a PNG format.

### 1.1.2 Technology Stack

Besides the standard libraries (Appcompat, Test...) that were included in our project, we used following frameworks for our implementation:

- **Android Jetpack suite** - we use a pack of standard libraries for creating main components of application like ViewModel, Canvas, Paint, Activity.
- **Material components available through Google Maven Repository** - which we used for the layout and specifying the main design. The main UI, where the user can draw, contains a blank canvas with a central floating action button. This floating action button is the primary point of interaction where User can open the menu and use other floating action points to start actions like drawing lines or writing text and also modifying the shapes by changing attributes.

  We decided to use a floating action button as a modern approach to design our app and to have a different layout than the most used and popular paint apps have.

- **Lifecycle-aware components available through Google Maven Repository**

  We used the lifecycle class to process the information about the lifecycle state of the components like sketch itself, which holds all the elements.

## 1.2 Design Patterns

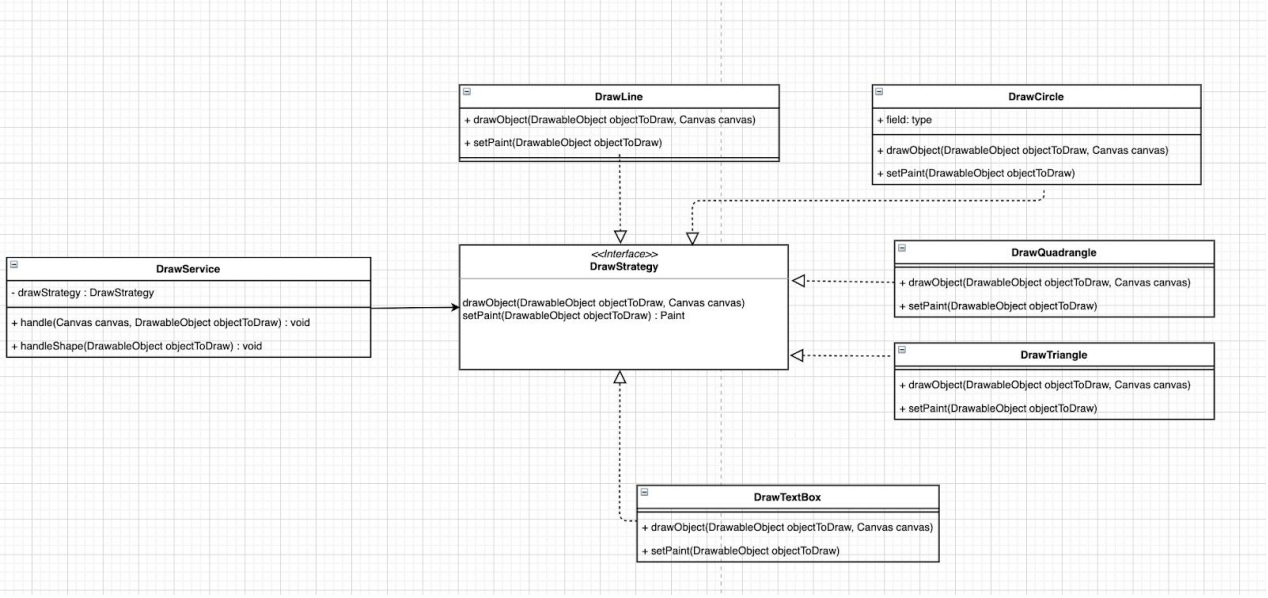
As the main application pattern, we decided to follow "Guide to app architecture" by Google and implement MVVM pattern with the LiveData library. It allows us to separate UI, business logic, data from each other. As an implementation for each Activity was created a separate ViewModel class which manages all logic/connection between UI and entities.

Also, we implemented Strategy and Factory patterns.

### 1.2.1 Strategy Pattern

We used the strategy pattern to implement drawing techniques in Canvas and draw such objects as textbox, circle, linie, triangle and rectangle.
This helps us to extend drawing logic into a separate service where we can manage different drawing algorithms.

The main benefit of using a strategy pattern here is that we can easily create new drawing algorithms when adding new DrawableObjects without changing the main View.



- DrawService defines a type of object to draw and create related algorithms.

```
public void handle(Canvas canvas, DrawableObject objectToDraw) {

    if (objectToDraw instanceof TextBox) {
        drawStrategy = new DrawTextBox();
    }
    if (objectToDraw instanceof Shape) {
        handleShape(objectToDraw);
    }

    if (null != drawStrategy) {
        drawStrategy.drawObject(objectToDraw, canvas);
    } else {
        //todo custom exception DrawableObjectNotDefined
    }
}
```

- Interface describe main functionality

```
public interface DrawStrategy {

    boolean drawObject(DrawableObject objectToDraw, Canvas canvas);
    Paint setPaint(DrawableObject objectToDraw);
}
```

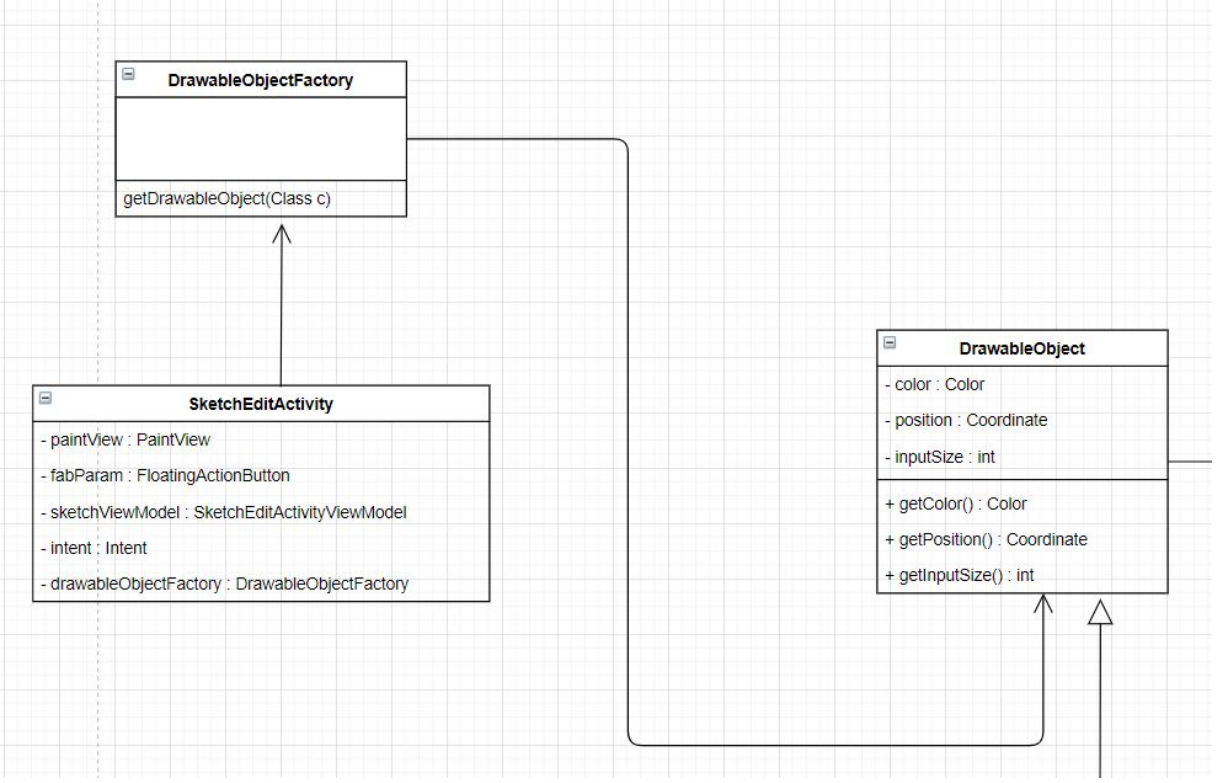
- And for instance algorithm of drawing circle in DrawCircle class that extends DrawStrategy

```
@Override
public boolean drawObject(DrawableObject objectToDraw, Canvas canvas) {
    canvas.drawCircle(
            objectToDraw.getPosition().getX(),
            objectToDraw.getPosition().getY(),
            ((Circle) objectToDraw).getRadius(),
            setPaint(objectToDraw)
    );
    return true;
}

@Override
public Paint setPaint(DrawableObject objectToDraw) {
    Paint mPaint = new Paint();
    mPaint.setColor(objectToDraw.getColor().getAndroidColor());
    mPaint.setAntiAlias(true);
    mPaint.setStyle(Paint.Style.STROKE);
    mPaint.setStrokeWidth(objectToDraw.getInputSize());
    return mPaint;
}
```

### 1.2.2 Factory Pattern

Factory Pattern being one of the most famous design patterns, is also implemented in our project and being the creational pattern it helps us to create an object in its best way.




- We are creating the object without exposing the creation logic to the client and refer to the newly created object. Here in the snapshot we create a DrawableObjectFactory class to generate objects of concrete class based on given information.

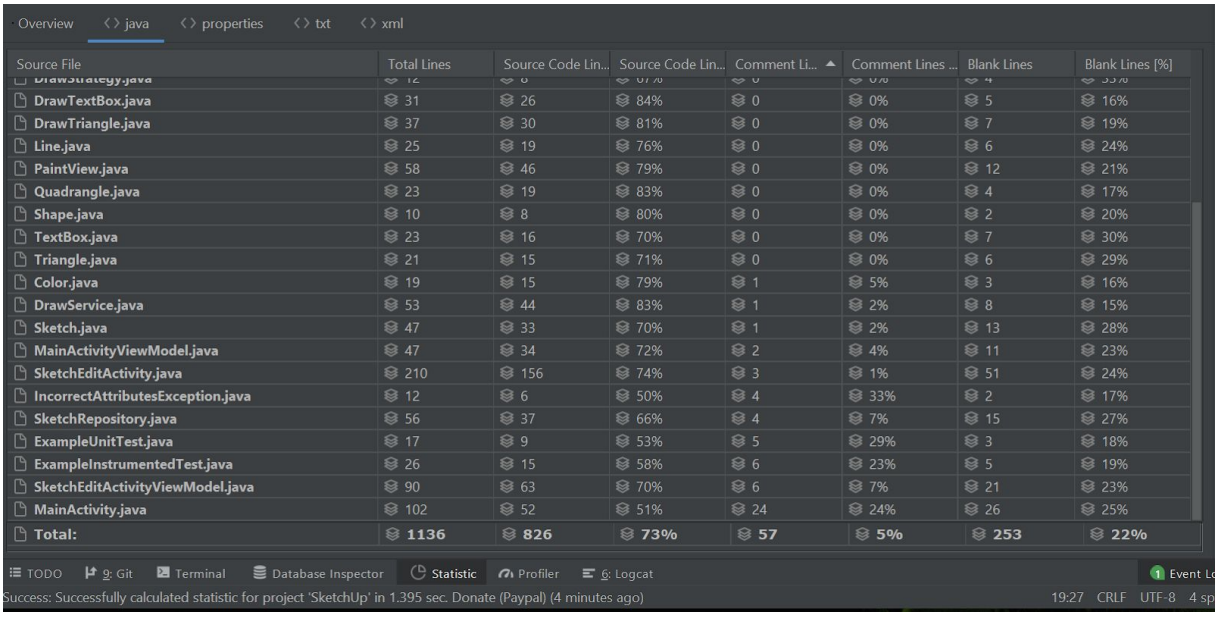
```
public class DrawableObjectFactory {
    public DrawableObject getDrawableObject(Class c) {
        if (c == Line.class) return new Line();
        else if (c == Circle.class) return new Circle();
        else if (c == Quadrangle.class) return new Quadrangle();
        else if (c == Triangle.class) return new Triangle();
        else if (c == TextBox.class) return new TextBox();
        return null;
    }
}
```

- In this method called buttonListener() we get and create the object of the concrete classes, which are extending the abstract Shape class: Line, Circle, Quadrangle, Triangle by passing information of the Class Type through the DrawableObjectFactory method.

```
private void buttonsLister() {
    fabParam.setOnClickListener(view -> createDialogForParam());
    fabText.setOnClickListener(view -> {
        setSelected(drawableObjectFactory.getDrawableObject(TextBox.class));
        createDialogForText();
    });
    fabCircle.setOnClickListener(view -> setSelected(drawableObjectFactory.getDrawableObject(Circle.class)));
    fabTriangle.setOnClickListener(view -> setSelected(drawableObjectFactory.getDrawableObject(Triangle.class)));
    fabQuad.setOnClickListener(view -> setSelected(drawableObjectFactory.getDrawableObject(Quadrangle.class)));
    fabLine.setOnClickListener(view -> setSelected(drawableObjectFactory.getDrawableObject(Line.class)));
}
```

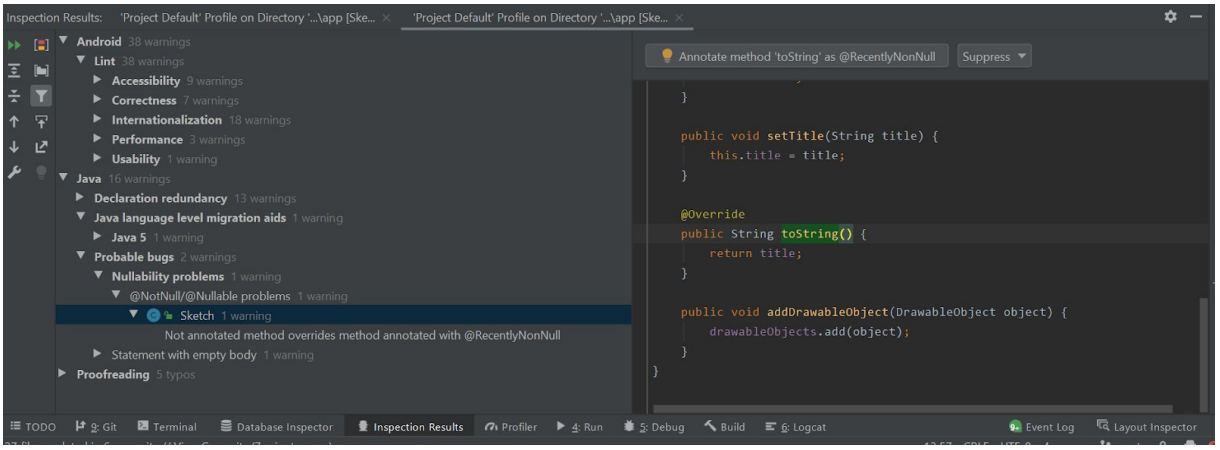
# 2 Code Metrics

Overall the project consists of 5 packages, 27 classes ,57 lines of comments and total number of codes is 1136.



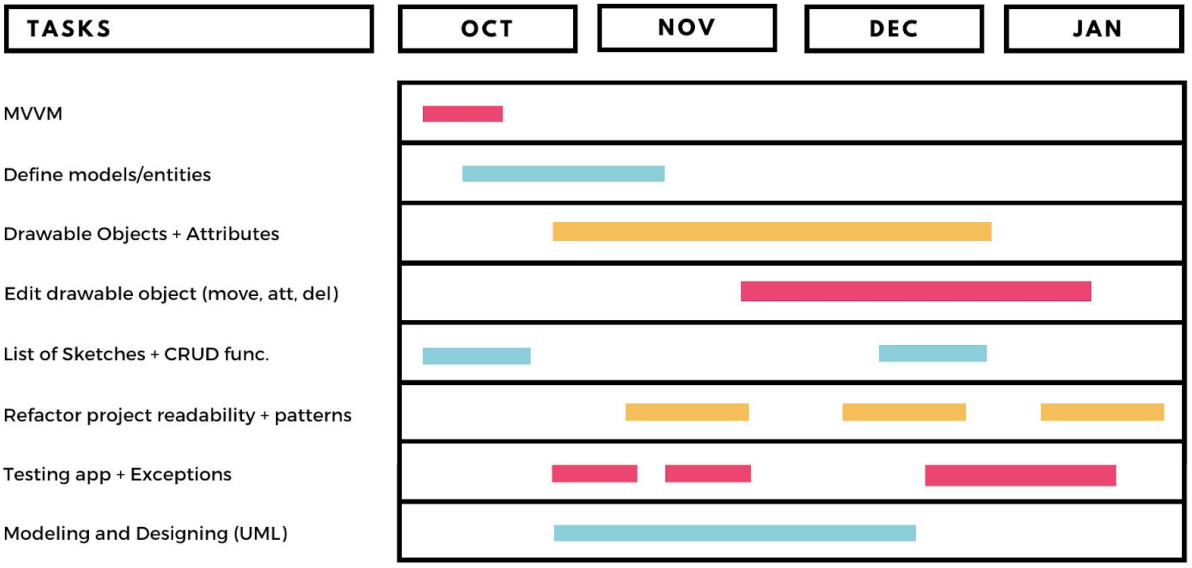
| Source File | Total Lines | Source Code Lin... | Source Code Lin... | Comment Li... | Comment Lines ... | Blank Lines | Blank Lines [%] |
|---|---|---|---|---|---|---|---|
| DrawTextBox.java | 31 | 26 | 84% | 0 | 0% | 5 | 16% |
| DrawTriangle.java | 37 | 30 | 81% | 0 | 0% | 7 | 19% |
| Line.java | 25 | 19 | 76% | 0 | 0% | 6 | 24% |
| PaintView.java | 58 | 46 | 79% | 0 | 0% | 12 | 21% |
| Quadrangle.java | 23 | 19 | 83% | 0 | 0% | 4 | 17% |
| Shape.java | 10 | 8 | 80% | 0 | 0% | 2 | 20% |
| TextBox.java | 23 | 16 | 70% | 0 | 0% | 7 | 30% |
| Triangle.java | 21 | 15 | 71% | 0 | 0% | 6 | 29% |
| Color.java | 19 | 15 | 79% | 1 | 5% | 3 | 16% |
| DrawService.java | 53 | 44 | 83% | 1 | 2% | 8 | 15% |
| Sketch.java | 47 | 33 | 70% | 1 | 2% | 13 | 28% |
| MainActivityViewModel.java | 47 | 34 | 72% | 2 | 4% | 11 | 23% |
| SketchEditActivity.java | 210 | 156 | 74% | 3 | 1% | 51 | 24% |
| IncorrectAttributesException.java | 12 | 6 | 50% | 4 | 33% | 2 | 17% |
| SketchRepository.java | 56 | 37 | 66% | 4 | 7% | 15 | 27% |
| ExampleUnitTest.java | 17 | 9 | 53% | 5 | 29% | 3 | 18% |
| ExampleInstrumentedTest.java | 26 | 15 | 58% | 6 | 23% | 5 | 19% |
| SketchEditActivityViewModel.java | 90 | 63 | 70% | 6 | 7% | 21 | 23% |
| MainActivity.java | 102 | 52 | 51% | 24 | 24% | 26 | 25% |
| Total: | 1136 | 826 | 73% | 57 | 5% | 253 | 22% |

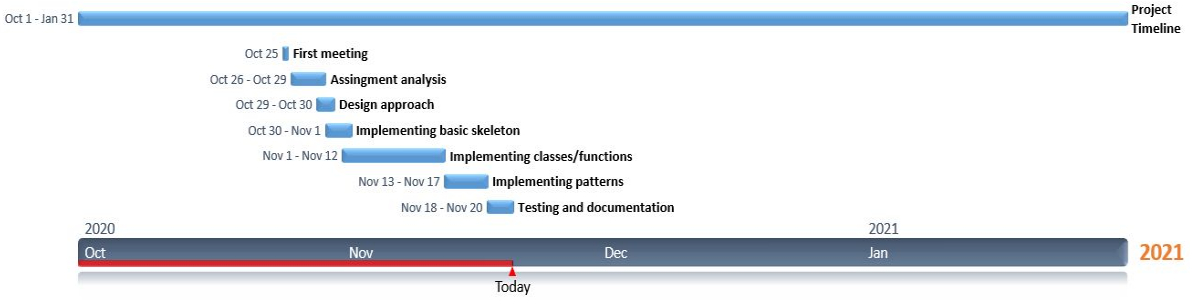
By invoking the **lint** task and entering the command ***gradlew lint*** from the root directory of the project and getting the following results in the snippet from the Inspection Results we were able to see the recommendation concerning Performance, usability, correctness and also probable bugs. There were only warnings concerning the probable bugs.
First was the problem of statement with empty body inspection and the other was the nullability annotation, non - annotated getters of annotated fields.


Inspection Results:
'Project Default' Profile on Directory '...\app [Ske...
'Project Default' Profile on Directory '...\app [Ske...
Android 38 warnings
Lint 38 warnings
Accessibility 9 warnings
Correctness 7 warnings
Internationalization 18 warnings
Performance 3 warnings
Usability 1 warning
Java 16 warnings
Declaration redundancy 13 warnings
Java language level migration aids 1 warning
Java 5 1 warning
Probable bugs 2 warnings
Nullability problems 1 warning
@NotNull/@Nullable problems 1 warning
Sketch 1 warning
Not annotated method overrides method annotated with @RecentlyNonNull
Statement with empty body 1 warning
Proofreading 5 typos
Annotate method 'toString' as @RecentlyNonNull
Suppress
public void setTitle(String title) {
this.title = title;
}
@Override
public String toString() {
return title;
}
public void addDrawableObject(DrawableObject object) {
drawableObjects.add(object);
}
}
TODO
9: Git
Terminal
Database Inspector
Inspection Results
Profiler
4: Run
5: Debug
Build
6: Logcat
Event Log
Layout Inspector

# 3 Team Contribution

## 3.1 Project Tasks and Schedule



## 3.2 Distribution of Work and Effort

| Team-Member | Task | Time used |
|---|---|---|
| Lala Mammadova | UML Class Diagram, SUPD Documentation, Factory Method Pattern implementation, Model Classes and some features for FR3, Code Clean-Up | 40h |
| Ema Dupovac-Kilincarslan | UML Class Diagram,SUPD Documentation,Custom ClassesUser Input for Color and Sizes and Exception Classes | 30h |
| Muhammed Akinci | UMLClass Diagram, SUPD Documentation, Implementation, Factory Method Pattern, FR3 drawing algorithms and base logic, Design and Placement of Buttons | 30h |
| Maxim Bogoutdinov | UML Class Diagram,SUPD Documentation, Strategy Pattern Implementation, MVVM, TextBox drawing | 30h |