Software Engineering 2  
SUPD REPORT

Status Update Report

|  |  |
| --- | --- |
| **Team number:** | 0310 |

|  |  |
| --- | --- |
| **Team member 1** | |
| **Name:** | Christian Orlowski |
| **Student ID:** | 12024267 |
| **E-mail address:** | a12024267@unet.univie.ac.at |

|  |  |
| --- | --- |
| **Team member 2** | |
| **Name:** | Jonas Michael Speiser |
| **Student ID:** | 12012545 |
| **E-mail address:** | a12012545@unet.univie.ac.at |

|  |  |
| --- | --- |
| **Team member 3** | |
| **Name:** | Sandra Tomeschek |
| **Student ID:** | 01504450 |
| **E-mail address:** | a01504450@unet.univie.ac.at |

|  |  |
| --- | --- |
| **Team member 4** | |
| **Name:** | Leyla Durdyyeva |
| **Student ID:** | 01576824 |
| **E-mail address:** | a01576824@unet.univie.ac.at |

|  |  |
| --- | --- |
| **Team member 5** | |
| **Name:** | Felix Achim Hubert Lindau |
| **Student ID:** | 11948883 |
| **E-mail address:** | a11948883@unet.univie.ac.at |

Dear students,

This template document suggests an outline for the required contents of the SUPD report. The included descriptions and examples are supposed to help you write a clear report that documents and presents your actual solution well. Please remove this additional text (similarily written as this text) and exemplary material before you submit your report!

The actual SUPD report document can be based on this template, or can be written using an arbitrary text editing program such as Latex, LibreOffice and the like, as long as the required material (described in the assignment document) is contained.

In general, write the report in such a manner as to provide all information to a third party who is not involved in the design and development and unfamiliar with the exact tasks of the semester project assignment. Think of this third party as a company that roughly wants to offer a software product as described in the assignment document and commissioned your team to start this software engineering project and produce a viable project which follows best practices, is of high quality, and is worth to be funded further after the SUPD deadline.

The SE2 Team.

# Design Draft

## Design Approach and Overview - Chris

Describe your design approach and how you arrived at your solution. Typically you will start with an initial solution that you refine in an iterative way by means of re-factorization. Use at least class diagrams and the used technology stack for documenting:

* major design decisions (e.g., design alternatives). Discuss and explain your design decisions also regarding the overall class layout. Possibly include design descriptions/class diagrams showing (a part of) the design before a refactorization step and/or alternatively considered (parts of) designs.
* a design overview of the current state of your solution at SUPD. Note that used design patterns should be well visible in this design overview (e.g., in the class diagrams). Regarding design patterns you may cross-reference to Section 1.2 for the details.

Carefully check all your UML diagrams for syntactical and semantical correctness!

### Class Diagrams - Leyla

Our class diagram follows the Model-View-Viewmodel pattern. Classes colored in green represent our View, classes in blue belong to the ViewModel and Classes colored in purple belong to the Model. Classes in grey will be implemented later.

### Technology Stack - Felix

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Website link | Version | Purpose and Reasoning |
| ColorPicker | https://github.com/kristiyanP/colorpicker | 1.1.10 | Is implementing the color palette for the color change of objects, which is linked to the floating button in the bottom left corner. Considered alternatives where the Holopicker (by M. Schweiz) and ColorPickerView (by skydoves). Holopicker was not fitting to our design so well, and the documentation of ColorPickerView was not as extensive as the ColorPicker library. |

## Design Patterns - Leyla

### Design Pattern 1: Factory Method Pattern

The Factory Method Pattern is useful for delegating the creation of new objects to a dedicated class. This is useful in order to better split the code and avoid bugs. The required attributes and constructor of a concrete object might also change over time and having the creation logic in one determined class makes it easier to address the changes.

In our project we needed to implement the Factory Method pattern to create objects of GraphicalElement type from our Model. These objects are being later displayed on the View.

This design pattern relates to the Functional Requirements FR1-FR4, requiring new Graphical Elements to be created and displayed on the View. The creation happens after the user selects a graphical element. The creation is an intermediary step between the data saved in the Model and the representation of the data in the View.

Diagram

Description automatically generated

Code snippet (GraphicalElementFactory.java):

public static GraphicalElement createElement(EGraphicalElementType type) throws AppException {  
 switch (type) {  
 case *LINE*:  
 return *createLine*();  
 case *CIRCLE*:  
 return *createCircle*();  
 case *DRAWING*:  
 case *COMPOSITE\_SHAPE*:  
 break;  
 case *TRIANGLE*:  
 return *createTriangle*();  
 case *QUADRANGLE*:  
 return *createQuadrangle*();  
 case *TEXT\_FIELD*:  
 return *createText*();  
 default:  
 throw new AppException("Unknown type: " + type);  
 }

private static Circle createCircle() {  
 Circle mCircle = new Circle(new DrawCircleStrategy());  
 Paint mPaint = new Paint(GraphicalElement.*getSelectedPaint*());  
 mCircle.setObjectPaint(mPaint);  
 mCircle.setShapeSize(70);  
 return mCircle;  
}

### Strategy Pattern

The Strategy Pattern is useful when there are multiple algorithms which solve a problem, and they can be used interchangeably, according to the concrete context. It is an alternative to implementing behavioural logic in subclasses of a Context, which has the benefit of separating business logic from Context state and thus making the code easier to read and maintain. Moreover, conditional statements for selecting the behaviour are also avoided thanks to this pattern.

In our project, the Strategy Pattern turned out useful to avoid adding logic for drawing on the Canvas directly in our concrete subclasses of GraphicalElement. The logic for drawing is different for each graphical element, different Strategies have been implemented.

This design pattern relates to the Functional Requirements FR1-FR4, requiring new Graphical Elements to be created and displayed (drawn) on the View. The Strategy pattern is executed when the whole View is being drawn again, by going through each graphical element and invoking the corresponding Strategy object. The drawing is performed by the ViewModel and used to represent the data on the View.

Diagram

Description automatically generated

Code Snippets:

public interface IDrawStrategy {  
 void draw(Canvas canvas, GraphicalElement graphicalElement);  
}

public class DrawTriangleStrategy implements IDrawStrategy {  
 @Override  
 public void draw(Canvas canvas, GraphicalElement graphicalElement) {  
 // in Anlehnung an: https://kylewbanks.com/blog/drawing-triangles-...

float halfWidth = graphicalElement.getShapeSize() / 2;  
 float x = graphicalElement.getxPosition();  
 float y = graphicalElement.getyPosition();  
  
 Path path = new Path();  
 path.moveTo(x, y - halfWidth); // Top  
 path.lineTo(x - halfWidth, y + halfWidth); // Bottom left  
 path.lineTo(x + halfWidth, y + halfWidth); // Bottom right  
 path.lineTo(x, y - halfWidth); // Back to Top  
 path.close();  
  
 canvas.drawPath(path, graphicalElement.getObjectPaint());  
 }  
}

public class DrawTextStrategy implements IDrawStrategy{  
 @Override  
 public void draw(Canvas canvas, GraphicalElement graphicalElement) {  
 canvas.drawText(((Text) graphicalElement).getTextInput(), graphicalElement.getxPosition(), graphicalElement.getyPosition(), graphicalElement.getObjectPaint());  
 }  
}

# Code Metrics - Alle

Provide information on at least the number of packages, lines of code, comment lines of code, number of classes and code bugs with regard to your current state of the implementation. It is recommendable to use a static code analysis tool (e.g. SpotBugs). Discuss your findings.

Code bugs and testing

Before running the final lint-test, the implementation showed around 70 errors. The majority of them were:

* Un-used resources which had piled up over the development process
* Display attributes of UI elements being missing (e.g. focusability)
* The non-local declaration of local variables
* Unused import-statements from the development process
* Usage of hardcoded values instead of flexible resources (strings)

The current implementation shows 21 warnings, whereas Java accounts for 8 and Android of the rest. The majority of Android errors recurs to our use of Resource ID’s in the switch-case statement in the main activity – which should be avoided in this context. We will eliminate this error and come up with another solution.

The rest of the warnings in the Android context recurs to minor aspects, we have considered these hints and came up with a suitable solution in these cases.

We have explicitly implemented three different Unit tests regarding methods in the Class CanvasViewTest for the setting of X and Y, as well as the selection of graphical elements. Through the latter, we gained an insight on the bug of selecting a text size/and or color when there has been no graphical element instantiated. This bug could be fixed. Furthermore, we had multiple phases of code reviews. Each team member checked the other program parts and used the TODO-function of Android studio to make suggestions on code improvement.

# Team Contribution

## Project Tasks and Schedule - Felix

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | **October** | | | | **November** | | | | **December** | | | | **January** | |
| JanPreparation/Planning |  | **1** | **2** | **3** | **4** | **1** | **2** | **3** | **4** | **1** | **2** | **3** | **4** | **1** | **2** |
| Requirements checking |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Task distribution |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Architectural decisions |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Development/ Test | GitLab Setup |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Implementation & Testing |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| SUPD submission |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Further Implementation & Testing |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Implementation | Check on Design Pattern Use |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Final Testing |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Final Code Clean-Up |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| DEAD submission |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

## Distribution of Work and Efforts – Chris & dann alle

Report in a table how you distributed the overall work among team members and how much time was spent by each team member on the tasks.