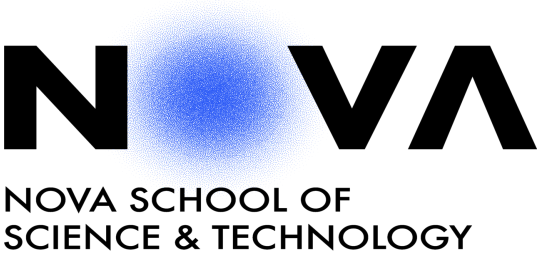


Engenharia de Software (MIEI) – 2022/2023 – 1º Semestre

Relatório do Projeto - Fase 2

Engenharia de Software

Realizado por:

Rodrigo Mesquita, nº57372, turno P3
Ana Gadelha, nº59943, turno P4
Bernardo Carvalho, nº60012, turno P6
Pedro Arruda, nº60663, turno P6
Martim Costa, nº64901, turno P4

GitHub Project Link:
Phase 1: https://github.com/anaGadelha19/ganttproject [1]
Phase 1&2: https://github.com/Martim-Costa/ganttproject
(O projeto encontra-se na pasta Project do master branch)

[1] Foi necessário criar um novo repositório devido a um erro na criação do primeiro que não nos permitia a alteração do branch

Índice

First Phase

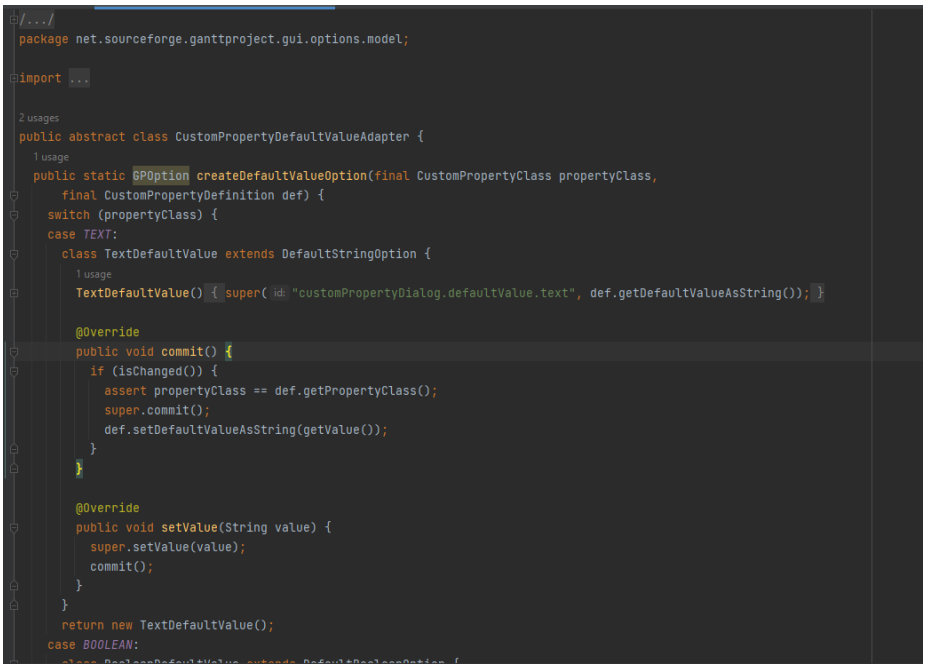
Design Patterns

Adapter Pattern

Pedro Arruda 60633

Location:
\ganttproject\src\main\java\net\sourceforge\ganttproject\gui\options\model\CustomPropertyDefaultValueAdapter.java

This design pattern is an **adapter**. This adapter facilitates communication between two other classes.



```
/.../
package net.sourceforge.ganttproject.gui.options.model;

import ...

2 usages
public abstract class CustomPropertyDefaultValueAdapter {
  1 usage
  public static GPOption createDefaultValueOption(final CustomPropertyClass propertyClass,
      final CustomPropertyDefinition def) {
    switch (propertyClass) {
    case TEXT:
      class TextDefaultValue extends DefaultStringOption {
        1 usage
        TextDefaultValue() { super( id: "customPropertyDialog.defaultValue.text", def.getDefaultValueAsString()); }

        @Override
        public void commit() {
          if (isChanged()) {
            assert propertyClass == def.getPropertyClass();
            super.commit();
            def.setDefaultValueAsString(getValue());
          }
        }

        @Override
        public void setValue(String value) {
          super.setValue(value);
          commit();
        }
      }
      return new TextDefaultValue();
    case BOOLEAN:
```

Reviewers:

- None

Singleton Pattern

Pedro Arruda 60663

Location:
\ganttproject\src\main\java\net\sourceforge\ganttproject\GanttGraphicArea.java

This design pattern is a **singleton**. With this singleton the class makes sure there can only be one instance of the object myChartComponentImpl and that it can be accessed globally.

Reviewers:

- Reviewed by Bernardo Carvalho 60012.

Facade Pattern

Pedro Arruda 60663

Location:
\ganttproject\src\main\java\net\sourceforge\ganttproject\gui\TreeUiFacade.java

This design pattern is a **facade**. This interface makes it much simpler to use its subsystems and results in less dependency and communication between the subsystems.

```java
import ...

/**
 * @author dbarashev (Dmitry Barashev)
 */
7 usages  4 implementations
public interface TreeUiFacade<T> {
  4 usages  1 implementation
  Component getTreeComponent();

  1 implementation
  ColumnList getVisibleFields();

  1 implementation
  boolean isVisible(T modelElement);

  1 implementation
  boolean isExpanded(T modelElement);

  2 usages  1 implementation
  void setExpanded(T modelElement, boolean value);

  1 implementation
  void applyPreservingExpansionState(T modelElement, Predicate<T> callable);
  /**
   * Modifies the selected node(s) of the tree
   *
   * @param clear
   *          when true, it first clears the previous selection. When false the
   *          current selection gets extended
   * @param modelElement
   *          to be selected
   */
```

Reviewers:

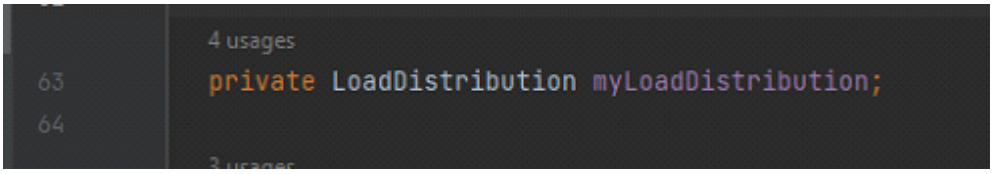
- Reviewed by Martim Costa 64901.

Singleton pattern

Bernardo Carvalho nº60012

Class location:
\ganttproject\src\main\java\net\sourceforge\ganttproject\resource\HumanResource.java

This design pattern is a singleton because the variable myloadDistribution is only initiated on the get method which returns it.




```
public LoadDistribution getLoadDistribution() {
  if (myLoadDistribution == null) {
    myLoadDistribution = new LoadDistribution( resource: this);
  }
  return myLoadDistribution;
}
```

Reviewers:

- Reviewed by Martim Costa 64901.

Memento pattern

Bernardo Carvalho nº60012

Class location:
\ganttproject\src\main\java\net\sourceforge\ganttproject\undo\UndoableEditImpl.java

The following design pattern is known as Memento Pattern. It is used in order to return an object to one of its previous states.

```
dbarashev +2
@Override
public void undo() throws CannotUndoException {
  try {
    restoreDocument(myDocumentBefore);
    if (projectDatabaseTxn != null) {
      try {
        projectDatabaseTxn.undo();
      } catch (ProjectDatabaseException e) {
        GPLogger.log(e);
      }
    }
  } catch (DocumentException | IOException e) {
    undoRedoExceptionHandler(e);
  }
}
```

Reviewers:

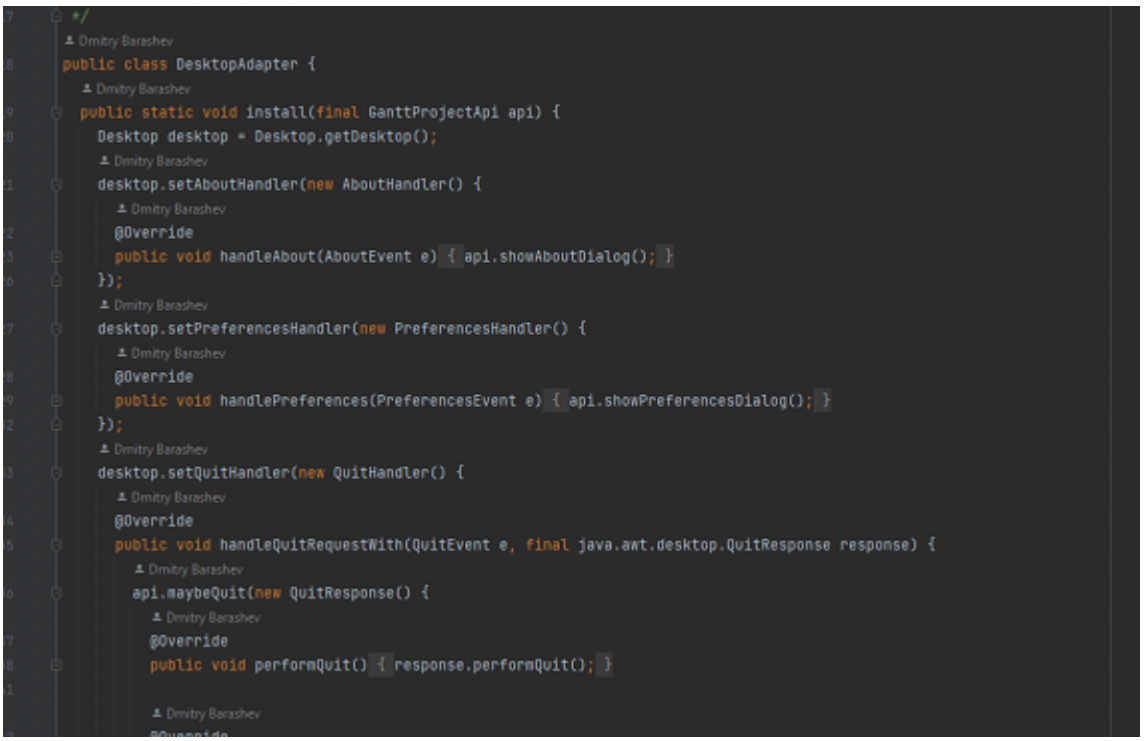
- Reviewed by Pedro Arruda 60663.

Adapter pattern

Bernardo Carvalho nº60012

Class location:
\ganttproject/blob/master/biz.ganttproject.desktop/src/biz/ganttproject/desktop/DesktopAdapter.java

This code pattern is the Adapter Pattern which is used to facilitate the communication between two systems of the program.



```
*/
Dmitry Barashev
public class DesktopAdapter {
  Dmitry Barashev
  public static void install(final GanttProjectApi api) {
    Desktop desktop = Desktop.getDesktop();
    Dmitry Barashev
    desktop.setAboutHandler(new AboutHandler() {
      Dmitry Barashev
      @Override
      public void handleAbout(AboutEvent e) { api.showAboutDialog(); }
    });
    Dmitry Barashev
    desktop.setPreferencesHandler(new PreferencesHandler() {
      Dmitry Barashev
      @Override
      public void handlePreferences(PreferencesEvent e) { api.showPreferencesDialog(); }
    });
    Dmitry Barashev
    desktop.setQuitHandler(new QuitHandler() {
      Dmitry Barashev
      @Override
      public void handleQuitRequestWith(QuitEvent e, final java.awt.desktop.QuitResponse response) {
        Dmitry Barashev
        api.maybeQuit(new QuitResponse() {
          Dmitry Barashev
          @Override
          public void performQuit() { response.performQuit(); }

          Dmitry Barashev
```

Reviewers:

- Reviewed by Rodrigo Mesquita 57372.

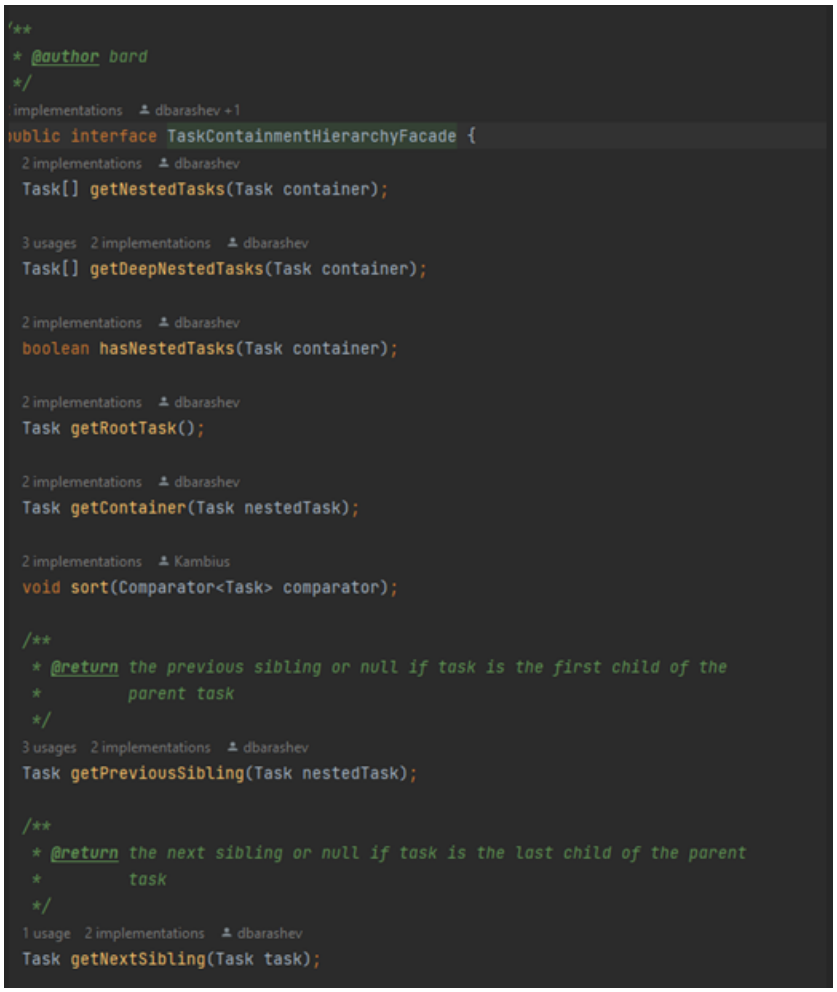
Facade

Ana Gadelha 59943

Class Location:

ganttproject/src/main/java/net/sourceforge/ganttproject/task/TaskContainmentHierarchyFacade.java

This design pattern is a Façade. What provides a simplified interface to a set of interfaces in a subsystem, therefore it hides the complexities of the subsystems.



```
/**
 * @author bard
 */
implementations  dbarashev + 1
public interface TaskContainmentHierarchyFacade {
    2 implementations  dbarashev
    Task[] getNestedTasks(Task container);

    3 usages  2 implementations  dbarashev
    Task[] getDeepNestedTasks(Task container);

    2 implementations  dbarashev
    boolean hasNestedTasks(Task container);

    2 implementations  dbarashev
    Task getRootTask();

    2 implementations  dbarashev
    Task getContainer(Task nestedTask);

    2 implementations  Kambius
    void sort(Comparator<Task> comparator);

    /**
     * @return the previous sibling or null if task is the first child of the
     *         parent task
     */
    3 usages  2 implementations  dbarashev
    Task getPreviousSibling(Task nestedTask);

    /**
     * @return the next sibling or null if task is the last child of the parent
     *         task
     */
    1 usage  2 implementations  dbarashev
    Task getNextSibling(Task task);
```

Reviewers:

- Reviewed by Pedro Arruda 60663.

Factory Method

Ana Gadelha 59943

Class Location:

ganttproject/src/main/java/net/sourceforge/ganttproject/parser/ParserFactory.java

ganttproject/src/main/java/net/sourceforge/ganttproject/GanttProject.java

This design pattern is a Factory Method. It defines an interface class for creating an object but it lets the subclasses decide which class to instantiate.

Reviewers:

- Reviewed by Bernardo Carvalho 60012.
- Reviewed by Rodrigo Mesquita 57372.

Singleton

Ana Gadelha 59943

Class Location:

ganttproject/src/main/java/net/sourceforge/ganttproject/GanttOptions.java

This design pattern is a **Singleton**, because this class has only one instance and it provides a global point of access to it.

Reviewers:

- Reviewed by Martim Costa 64901.

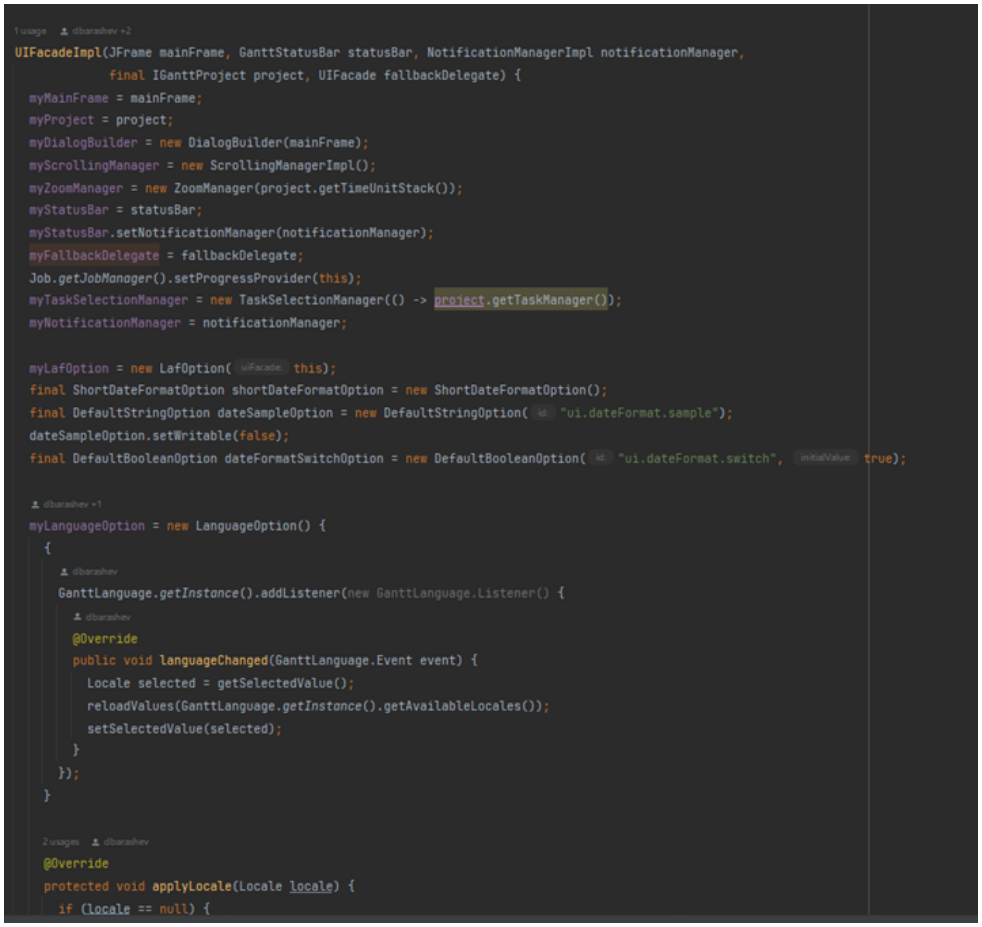
Facade Martim Costa 64901

Interface location: \ganttproject\src\main\java\net\sourceforge\ganttproject\gui\UIFacade

Class location: \ganttproject\src\main\java\net\sourceforge\ganttproject\UIFacadeImpl

This design pattern is a **Facade**. This Facade provides a unified interface between the different managers (like the notification manager, task selection manager, zoom manager, ...) and other classes too. This way is much simpler to use the subsystems that would be a lot more complex without this Facade.

```java
/*...*/
package net.sourceforge.ganttproject.gui;

import ...

/**
 * @author bard
 */
4 implementations  dbarashev +3
public interface UIFacade {
  4 usages
  ImageIcon DEFAULT_LOGO = new ImageIcon(UIFacade.class.getResource( name: "/icons/big.png"));

  1 implementation  dbarashev +2
  interface Dialog {
    1 implementation  dbarashev
    void show();

    1 implementation  dbarashev
    void hide();

    1 implementation  dbarashev
    void layout();

    1 implementation  dbarashev
    void center(Centering centering);
    1 usage  1 implementation  Dmitry Barashev
    void onShown(Runnable onShown);
    2 usages  1 implementation  Dmitry Barashev
    void onClosed(Runnable onClosed);
    //void resize();
  }

  16 usages  dbarashev
  public enum Centering {
    3 usages
    SCREEN, WINDOW
  };

  14 usages  dbarashev
  public enum Choice {
    3 usages
    YES, NO, CANCEL, OK
```



```
1 usage  dbarashev +2
UIFacadeImpl(JFrame mainFrame, GanttStatusBar statusBar, NotificationManagerImpl notificationManager,
             final IGanttProject project, UIFacade fallbackDelegate) {
  myMainFrame = mainFrame;
  myProject = project;
  myDialogBuilder = new DialogBuilder(mainFrame);
  myScrollingManager = new ScrollingManagerImpl();
  myZoomManager = new ZoomManager(project.getTimeUnitStack());
  myStatusBar = statusBar;
  myStatusBar.setNotificationManager(notificationManager);
  myFallbackDelegate = fallbackDelegate;
  Job.getJobManager().setProgressProvider(this);
  myTaskSelectionManager = new TaskSelectionManager(() -> project.getTaskManager());
  myNotificationManager = notificationManager;

  myLafOption = new LafOption( uiFacade: this);
  final ShortDateFormatOption shortDateFormatOption = new ShortDateFormatOption();
  final DefaultStringOption dateSampleOption = new DefaultStringOption( id: "ui.dateFormat.sample");
  dateSampleOption.setWritable(false);
  final DefaultBooleanOption dateFormatSwitchOption = new DefaultBooleanOption( id: "ui.dateFormat.switch",  initialValue: true);

  dbarashev +1
  myLanguageOption = new LanguageOption() {
    {
      dbarashev
      GanttLanguage.getInstance().addListener(new GanttLanguage.Listener() {
        dbarashev
        @Override
        public void languageChanged(GanttLanguage.Event event) {
          Locale selected = getSelectedValue();
          reloadValues(GanttLanguage.getInstance().getAvailableLocales());
          setSelectedValue(selected);
        }
      });
    }

    2 usages  dbarashev
    @Override
    protected void applyLocale(Locale locale) {
      if (locale == null) {
```

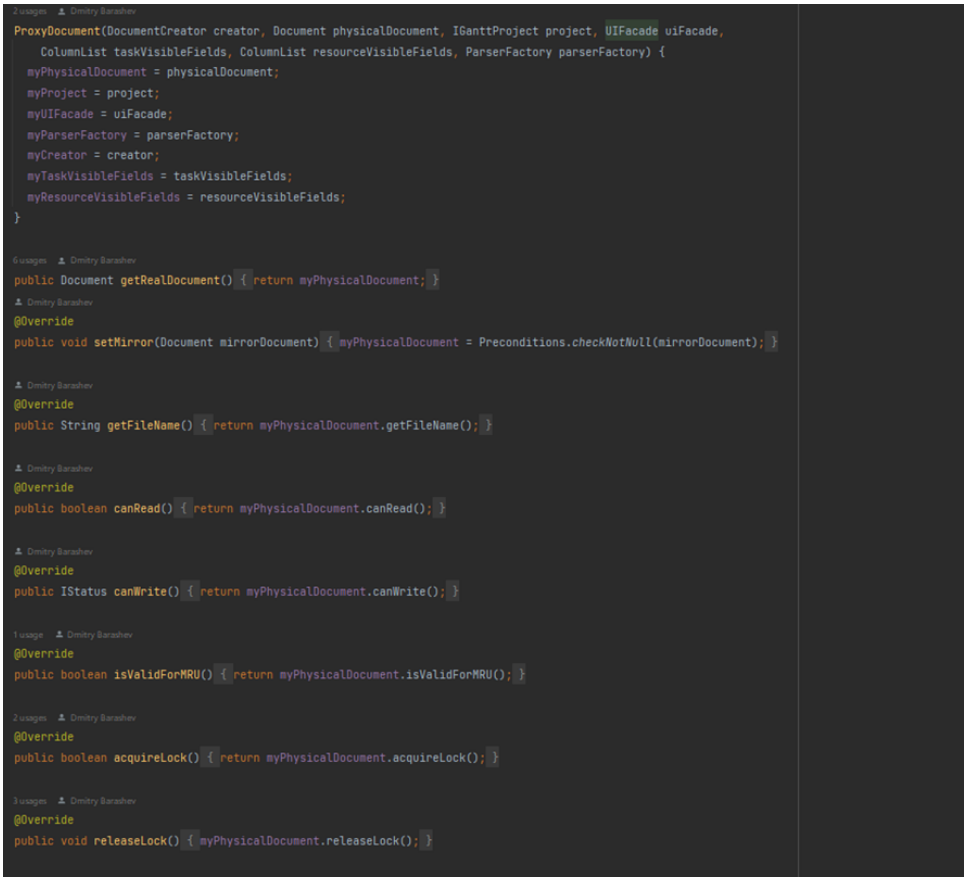
Reviewers:

- Reviewed by Ana Gadelha 59943.

Proxy Martim Costa 64901

Location:
\ganttproject\src\main\java\net\sourceforge\ganttproject\document\ProxyDocument

This design pattern is a Proxy (as the name of the class suggests). It performs the same tasks as the "original" class but in this case is used to not access directly to a document (it has a function called "getRealDocument" that will redirect to the "original" document, for example it is used in the storage of the documents).



```
2 usages  Dmitry Barashev
ProxyDocument(DocumentCreator creator, Document physicalDocument, IGanttProject project, UIFacade uiFacade,
    ColumnList taskVisibleFields, ColumnList resourceVisibleFields, ParserFactory parserFactory) {
  myPhysicalDocument = physicalDocument;
  myProject = project;
  myUIFacade = uiFacade;
  myParserFactory = parserFactory;
  myCreator = creator;
  myTaskVisibleFields = taskVisibleFields;
  myResourceVisibleFields = resourceVisibleFields;
}

6 usages  Dmitry Barashev
public Document getRealDocument() { return myPhysicalDocument; }
 Dmitry Barashev
@Override
public void setMirror(Document mirrorDocument) { myPhysicalDocument = Preconditions.checkNotNull(mirrorDocument); }

 Dmitry Barashev
@Override
public String getFileName() { return myPhysicalDocument.getFileName(); }

 Dmitry Barashev
@Override
public boolean canRead() { return myPhysicalDocument.canRead(); }

 Dmitry Barashev
@Override
public IStatus canWrite() { return myPhysicalDocument.canWrite(); }

1 usage  Dmitry Barashev
@Override
public boolean isValidForMRU() { return myPhysicalDocument.isValidForMRU(); }

2 usages  Dmitry Barashev
@Override
public boolean acquireLock() { return myPhysicalDocument.acquireLock(); }

3 usages  Dmitry Barashev
@Override
public void releaseLock() { myPhysicalDocument.releaseLock(); }
```

Reviewers:

- Reviewed by Rodrigo Mesquita 57372.

Singleton

Martim Costa 64901

Location: \ganttproject\src\main\java\net\sourceforge\ganttproject\gui\GattLookAndFeels

This design pattern is a Singleton. This way the class ensures no other instance is created, by intercepting requests to create new objects and provides the sole way to access the instance. It has a protected constructor, and it has the public method that instantiates the class “if” it is not already instantiated.

```
protected GanttLookAndFeels() {
  infoByClass = new HashMap<String, GanttLookAndFeelInfo>();
  infoByName = new HashMap<String, GanttLookAndFeelInfo>();
  LookAndFeelInfo[] lookAndFeels = UIManager.getInstalledLookAndFeels();
  for (int i = 0; i < lookAndFeels.length; i++) {
    GanttLookAndFeelInfo info = new GanttLookAndFeelInfo(lookAndFeels[i]);
    addLookAndFeel(info);
  }
}

1 usage  dbarashev
protected void addLookAndFeel(GanttLookAndFeelInfo info) {
  if (info.getName().startsWith("Kunststoff") && System.getProperty("os.name").startsWith("Mac")) {
    System.err.println("LookAndFeel not added (Kunststoff is ignored on MacOS).");
  } else {
    if (!infoByClass.containsKey(info.getClassName())) {
      infoByClass.put(info.getClassName(), info);
      infoByName.put(info.getName(), info);
    } else {
      System.err.println("LookAndFeel " + info + "(" + info.getClassName() + ") already installed.");
    }
  }
}

2 usages  dbarashev
public GanttLookAndFeelInfo getInfoByClass(String className) { return infoByClass.get(className); }

3 usages  dbarashev
public GanttLookAndFeelInfo getInfoByName(String name) { return infoByName.get(name); }

1 usage  dbarashev
public GanttLookAndFeelInfo getDefaultInfo() {
  GanttLookAndFeelInfo info = getInfoByClass(UIManager.getSystemLookAndFeelClassName());
  if (null == info)
    info = getInfoByClass(UIManager.getCrossPlatformLookAndFeelClassName());
  return info;
}

1 usage  dbarashev
public GanttLookAndFeelInfo[] getInstalledLookAndFeels() {
  GanttLookAndFeelInfo[] lookAndFeels = new GanttLookAndFeelInfo[0];
  return infoByClass.values().toArray(lookAndFeels);
}
```

```
5 usages  dbarashev
public static GanttLookAndFeels getGanttLookAndFeels() {
  if (singleton == null) {
    singleton = new GanttLookAndFeels();
  }
  return singleton;
}
}
```

Reviewers:

- Reviewed by Bernardo Carvalho 60012.

Proxy Rodrigo Mesquita 57372

Class location:

\ganttproject\ganttproject\src\main\java\net\sourceforge\ganttproject\document\ReadOnlyProxyDocument.java

This pattern is a Proxy. It allows a proxy class to represent a real class, in this case, this proxy class represents read only documents (verified with the canRead() and canWrite() methods).

Reviewers:

- **None**

Singleton Rodrigo Mesquita 57372

Class location:

\Java\jdk-17.0.2\lib\src.zip!\jdk.jdi\com\sun\tools\jdi\TargetVM.java

This design pattern is a Singleton because that is the only instance in the EventController class.

```
241     private EventController eventController() {
242         if (eventController == null) {
243             eventController = new EventController();
244         }
245         return eventController;
246     }
```

Reviewers:

- Reviewed by Pedro Arruda 60663.

Facade

Rodrigo Mesquita 57372

Class location:

\ganttproject\ganttproject\src\main\java\net\sourceforge\ganttproject\chart\mouse\MouseInteraction.java

This design pattern is a Facade, it provides a simplified interface that hides the complexity of a complex subsystem.

Reviewers:

- Reviewed by Ana Gadelha 59943.

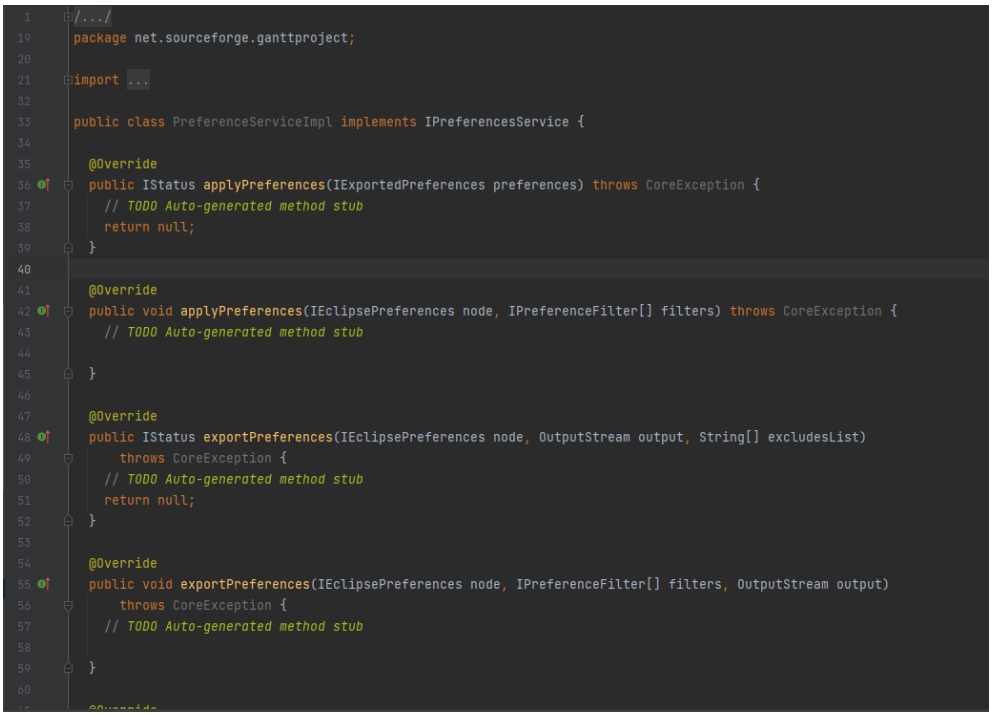
Code Smells

Dead Code

Pedro Arruda 60633

Location:
\ganttproject\src\main\java\net\sourceforge\ganttproject\PreferenceServiceImpl.java

This class is **dead code** due to it not being used and therefore should be removed to reduce code size and bring simpler support.



```
/.../
package net.sourceforge.ganttproject;

import ...

public class PreferenceServiceImpl implements IPreferencesService {

  @Override
  public IStatus applyPreferences(IExportedPreferences preferences) throws CoreException {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public void applyPreferences(IEclipsePreferences node, IPreferenceFilter[] filters) throws CoreException {
    // TODO Auto-generated method stub

  }

  @Override
  public IStatus exportPreferences(IEclipsePreferences node, OutputStream output, String[] excludesList)
      throws CoreException {
    // TODO Auto-generated method stub
    return null;
  }

  @Override
  public void exportPreferences(IEclipsePreferences node, IPreferenceFilter[] filters, OutputStream output)
      throws CoreException {
    // TODO Auto-generated method stub

  }
```

Reviewers:

- Reviewed by Bernardo Carvalho 60012.

No Comments

Pedro Arruda 60663

Location:
\ganttproject\src\main\java\net\sourceforge\ganttproject\TreeTableCellEditorImpl.java

This entire class has **no comments** making it very difficult for anyone to understand what the methods/class are doing or should be doing.

```
/.../
package net.sourceforge.ganttproject;

import ...

3 usages
class TreeTableCellEditorImpl implements TableCellEditor {
  11 usages
  private final DefaultCellEditor myProxiedEditor;
  5 usages
  private final JTable myTable;

  1 usage
  TreeTableCellEditorImpl(DefaultCellEditor proxiedEditor, JTable table) {
    assert proxiedEditor != null;
    myProxiedEditor = proxiedEditor;
    myTable = table;
  }

  @Override
  public Component getTableCellEditorComponent(JTable arg0, Object arg1, boolean arg2, int arg3, int arg4) {
    final Component result = myProxiedEditor.getTableCellEditorComponent(arg0, arg1, arg2, arg3, arg4);
    if (result instanceof JTextComponent) {
      ((JTextComponent) result).selectAll();
      //myFocusCommand = createSelectAllCommand((JTextComponent)result);
    }
    return result;
  }

  @Override
  public Object getCellEditorValue() { return myProxiedEditor.getCellEditorValue(); }
```

Reviewers:

- Reviewed by Ana Gadelha 59943.

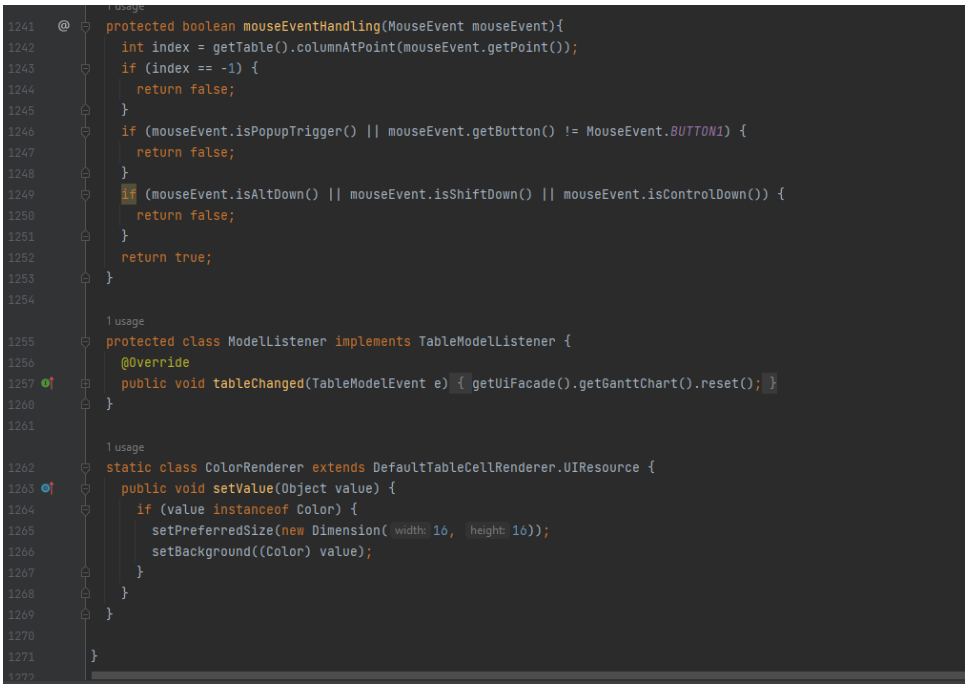
God Class

Pedro Arruda 60663

Location:

\ganttproject\src\main\java\net\sourceforge\ganttproject\TreeTableCellEditorImpl.java

Same class as before, this class has a **really large size** (1270 lines), turning it into a **god class** due to the amount of responsibility it has and making it even more difficult and exhausting to understand. This class should, for example, be divided into different classes, each with its specific functionalities.



```
1241  @   protected boolean mouseEventHandling(MouseEvent mouseEvent){
1242        int index = getTable().columnAtPoint(mouseEvent.getPoint());
1243        if (index == -1) {
1244          return false;
1245        }
1246        if (mouseEvent.isPopupTrigger() || mouseEvent.getButton() != MouseEvent.BUTTON1) {
1247          return false;
1248        }
1249        if (mouseEvent.isAltDown() || mouseEvent.isShiftDown() || mouseEvent.isControlDown()) {
1250          return false;
1251        }
1252        return true;
1253      }
1254
                1 usage
1255      protected class ModelListener implements TableModelListener {
1256        @Override
1257        public void tableChanged(TableModelEvent e) { getUiFacade().getGanttChart().reset(); }
1260      }
1261
                1 usage
1262      static class ColorRenderer extends DefaultTableCellRenderer.UIResource {
1263        public void setValue(Object value) {
1264          if (value instanceof Color) {
1265            setPreferredSize(new Dimension( width: 16,  height: 16));
1266            setBackground((Color) value);
1267          }
1268        }
1269      }
1270
1271  }
```

Reviewers:

- Reviewed by Martim Costa 64901.

Unnecessary commentary

Bernardo Carvalho nº60012

Class location:
\ganttproject\src\main\java\net\sourceforge\ganttproject\GanttProject.java

code smell - A lot of unnecessary commentary in this method.

```
1 usage   dbarashev
private MouseListener getStopEditingMouseListener() {
  if (myStopEditingMouseListener == null)
    dbarashev
    myStopEditingMouseListener = new MouseAdapter() {
      // @Override
      // public void mouseClicked(MouseEvent e) {
      // if (e.getSource() != bNew && e.getClickCount() == 1) {
      // tree.stopEditing();
      // }
      // if (e.getButton() == MouseEvent.BUTTON1
      // && !(e.getSource() instanceof JTable)
      // && !(e.getSource() instanceof AbstractButton)) {
      // Task taskUnderPointer =
      // area.getChartImplementation().findTaskUnderPointer(e.getX(),
      // e.getY());
      // if (taskUnderPointer == null) {
      // getTaskSelectionManager().clear();
      // }
      // }
      // }
    };
  return myStopEditingMouseListener;
}
```

Reviewers:

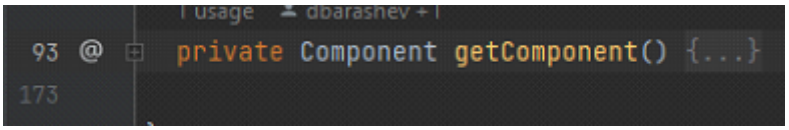
- Reviewed by Pedro Arruda 60663.

Massive method

Bernardo Carvalho nº60012

Class location:
\ganttproject\src\main\java\net\sourceforge\ganttproject\gui\AbstractPagesDialog.java

code smell- Massive method, perhaps there is a better way to distribute it in a group of smaller methods.



Reviewers:

- Reviewed by Ana Gadelha 59943.

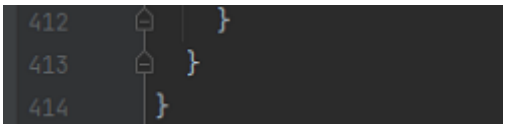
Very big Class

Bernardo Carvalho nº60012

Class location:

\ganttproject\src\main\java\net\sourceforge\ganttproject\gui\FileChooserPageBase.java

code smell-This class has over 400 lines of code and it’s not even the system. Perhaps it should be spread out between smaller classes.



Reviewers:

- Reviewed by Martim Costa 64901.
- Reviewed by Rodrigo Mesquita 57372.

Dead Code Ana Gadelha 59943

Class Location:

ganttproject/src/main/java/net/sourceforge/ganttproject/action/project/CloudProjectActionBase.java

This class is Dead Code because it is never used, it should be removed, to simplify the code and reduce the code size.

Reviewers:

- Reviewed by Martim Costa 64901.

Long Parameter List Ana Gadelha 59943

Class Location:

ganttproject/src/main/java/net/sourceforge/ganttproject/chart/gantt/GanttChartController.java

This constructor has too many arguments, which can indicate that this class has more functionalities than what it should have.

A way to make it simpler would be to aggregate some of the arguments into a class, and then the constructor would only have to receive one argument instead of receiving them individually.

```
1 usage   Dmitry Barashev +2
public GanttChartController(IGanttProject project, UIFacade uiFacade, ChartModelImpl chartModel,
                            ChartComponentBase chartComponent, ChartViewState chartViewState,
                            TaskTableChartConnector taskTableConnector,
                            Supplier<TaskTableActionConnector> taskTableActionFacade) {
  super(project, uiFacade, chartModel, chartComponent);
  myChartViewState = chartViewState;
  myTaskManager = project.getTaskManager();
  myChartModel = chartModel;
  myMouseListener = new MouseListenerImpl( chartImplementation: this, uiFacade, chartComponent, taskTableActionFacade);
  myMouseMotionListener = new MouseMotionListenerImpl( chartImplementation: this, uiFacade, chartComponent);
  mySelectionManager = uiFacade.getTaskSelectionManager();
  mySelection = new GanttChartSelection(myTaskManager, mySelectionManager);
  myTaskTableConnector = taskTableConnector;
  myTaskTableConnector.getVisibleTasks().addListener(
      (ListChangeListener<Task>) c -> SwingUtilities.invokeLater(this::reset)
  );
  myTaskTableConnector.getTableScrollOffset().addListener(
      (ChangeListener<? super Number>) (wtf, old, newValue) -> SwingUtilities.invokeLater(() -> {
        getChartModel().setVerticalOffset(newValue.intValue());
        reset();
      })
  );
```

Reviewers:

- Reviewed by Bernardo Carvalho 60012.

Unnecessary Comments

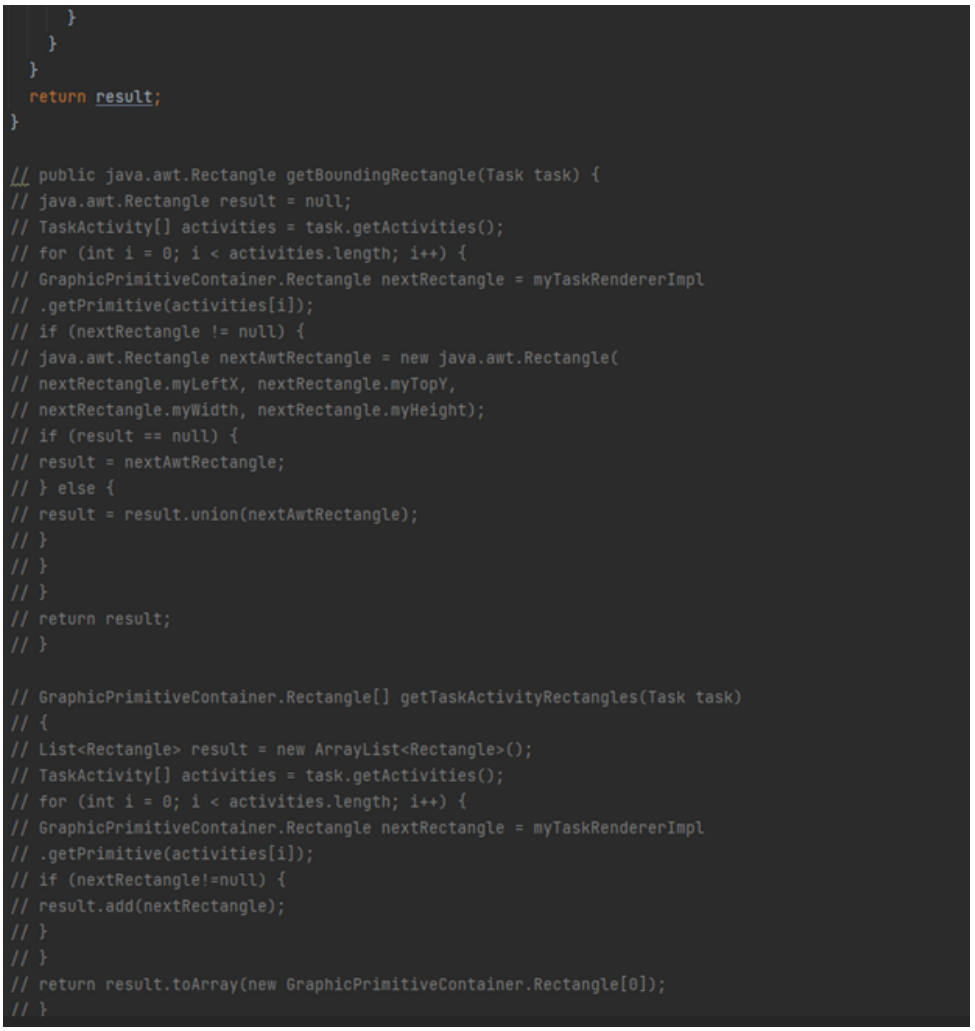
Ana Gadelha 59943

Class Location:

ganttproject/src/main/java/net/sourceforge/ganttproject/chart/ChartModelImpl.java

This class has a lot of unnecessary comments, which can be confusing, and also mean that the method is too complex, what causes the need to exaggerate on the comments.

A solution to this would be to remove the unnecessary comments and simplify the method so tat it does not need such an explanation.



```
    }
   }
  }
  return result;
}

// public java.awt.Rectangle getBoundingRectangle(Task task) {
// java.awt.Rectangle result = null;
// TaskActivity[] activities = task.getActivities();
// for (int i = 0; i < activities.length; i++) {
// GraphicPrimitiveContainer.Rectangle nextRectangle = myTaskRendererImpl
// .getPrimitive(activities[i]);
// if (nextRectangle != null) {
// java.awt.Rectangle nextAwtRectangle = new java.awt.Rectangle(
// nextRectangle.myLeftX, nextRectangle.myTopY,
// nextRectangle.myWidth, nextRectangle.myHeight);
// if (result == null) {
// result = nextAwtRectangle;
// } else {
// result = result.union(nextAwtRectangle);
// }
// }
// }
// return result;
// }

// GraphicPrimitiveContainer.Rectangle[] getTaskActivityRectangles(Task task)
// {
// List<Rectangle> result = new ArrayList<Rectangle>();
// TaskActivity[] activities = task.getActivities();
// for (int i = 0; i < activities.length; i++) {
// GraphicPrimitiveContainer.Rectangle nextRectangle = myTaskRendererImpl
// .getPrimitive(activities[i]);
// if (nextRectangle!=null) {
// result.add(nextRectangle);
// }
// }
// return result.toArray(new GraphicPrimitiveContainer.Rectangle[0]);
// }
```

Reviewers:

- Reviewed by Pedro Arruda 60663.

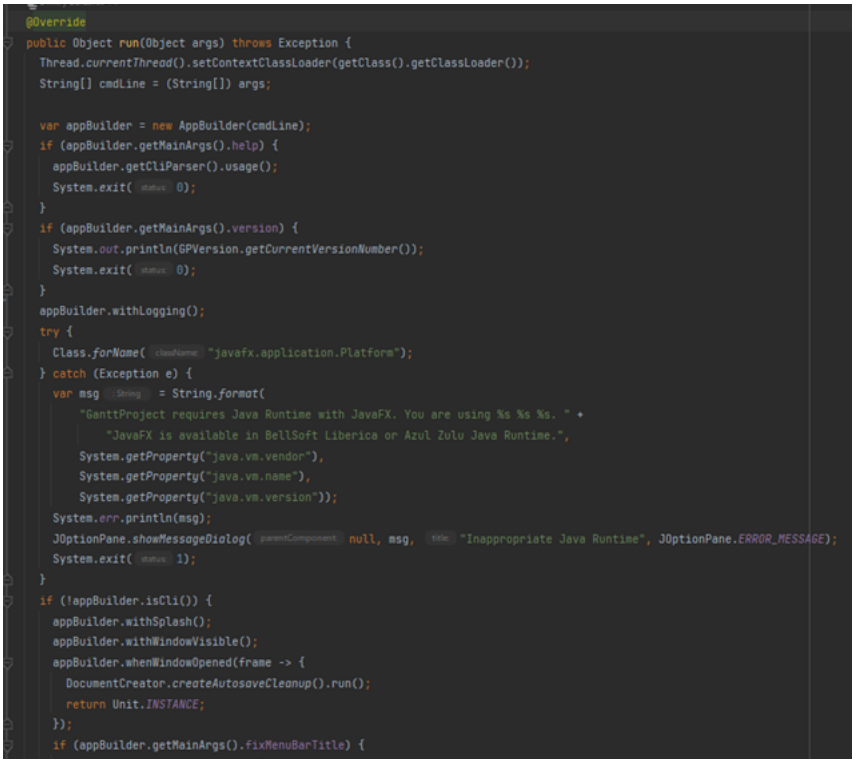
Huge Function Martim Costa 64901

Location:

\ganttproject\src\main\java\net\sourceforge\ganttproject\application\MainApplication

This function has a large size (around 88 lines), making it difficult and exhausting to understand the processes that are happening.

My suggestion to improve and refactor this code is to create smaller methods to decrease the size of this function, making it easier to read and debug.



```
@Override
public Object run(Object args) throws Exception {
  Thread.currentThread().setContextClassLoader(getClass().getClassLoader());
  String[] cmdLine = (String[]) args;

  var appBuilder = new AppBuilder(cmdLine);
  if (appBuilder.getMainArgs().help) {
    appBuilder.getCliParser().usage();
    System.exit( status: 0);
  }
  if (appBuilder.getMainArgs().version) {
    System.out.println(GPVersion.getCurrentVersionNumber());
    System.exit( status: 0);
  }
  appBuilder.withLogging();
  try {
    Class.forName( className: "javafx.application.Platform");
  } catch (Exception e) {
    var msg = String.format(
        "GanttProject requires Java Runtime with JavaFX. You are using %s %s %s. " +
            "JavaFX is available in BellSoft Liberica or Azul Zulu Java Runtime.",
        System.getProperty("java.vm.vendor"),
        System.getProperty("java.vm.name"),
        System.getProperty("java.vm.version"));
    System.err.println(msg);
    JOptionPane.showMessageDialog( parentComponent: null, msg, title: "Inappropriate Java Runtime", JOptionPane.ERROR_MESSAGE);
    System.exit( status: 1);
  }
  if (!appBuilder.isCli()) {
    appBuilder.withSplash();
    appBuilder.withWindowVisible();
    appBuilder.whenWindowOpened(frame -> {
      DocumentCreator.createAutosaveCleanup().run();
      return Unit.INSTANCE;
    });
    if (appBuilder.getMainArgs().fixMenuBarTitle) {
```

```
      try {
        var toolkit = Toolkit.getDefaultToolkit();
        var awtAppClassNameField = toolkit.getClass().getDeclaredField( "awtAppClassName");
        awtAppClassNameField.setAccessible(true);
        awtAppClassNameField.set(toolkit, InternationalizationKt.getRootLocalizer().formatText( "appliTitle"));
      } catch (NoSuchFieldException | IllegalAccessException ex) {
        System.err.println("Can't set awtAppClassName (needed on Linux to show app name in the top panel)");
      }
      return Unit.INSTANCE;
    });
  }
} 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;
  });
}
var files = appBuilder.getMainArgs().file;
if (files != null && !files.isEmpty()) {
  appBuilder.withDocument(files.get(0));
}


Consumer<Boolean> onApplicationQuit = withSystemExit -> {
  synchronized(myLock) {
    myLock.set(withSystemExit);
    myLock.notify();
  }
};
GanttProject.setApplicationQuitCallback(onApplicationQuit);
appBuilder.launch();
try {
  synchronized (myLock) {
    logger.debug( "Waiting until main window closes");
    myLock.wait();
    logger.debug( "Main window has closed");
  }
} catch (InterruptedException ex) {
  ex.printStackTrace();
```

```
} catch (InterruptedException ex) {
  ex.printStackTrace();
}
logger.debug( "Program terminated");
GPLogger.close();
if (myLock.get()) {
  System.exit( 0);
}
return null;
}
```

Reviewers:

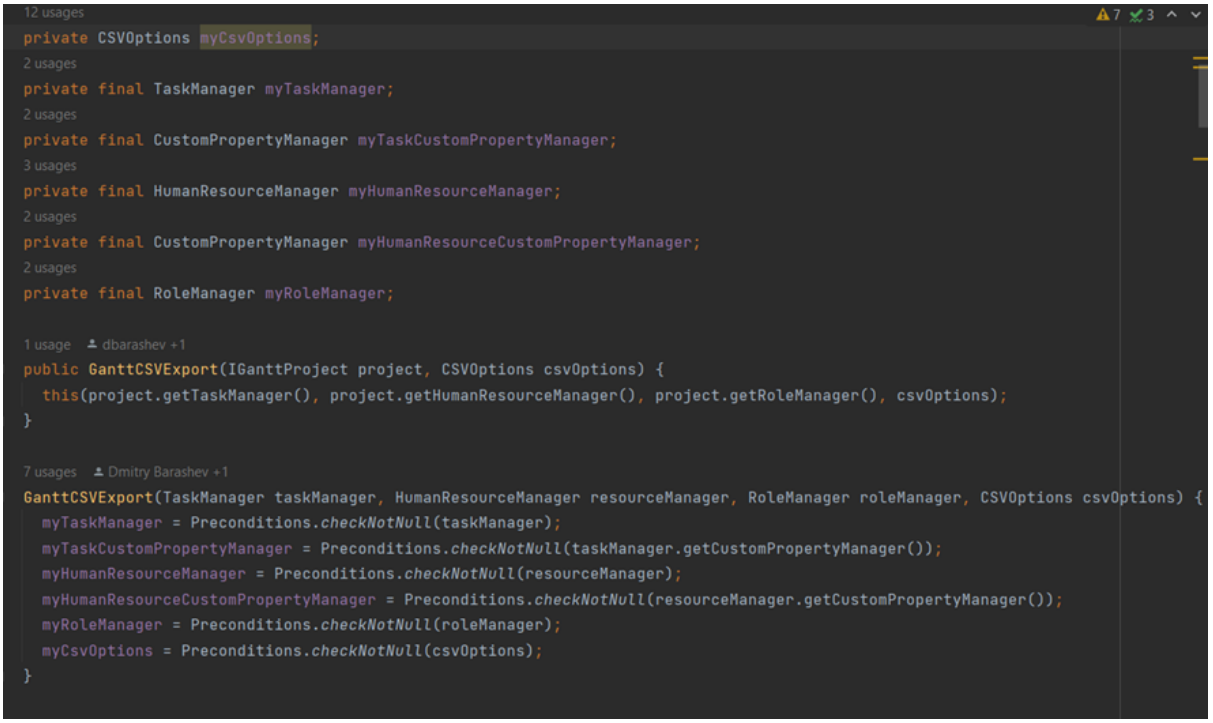
- Reviewed by Bernardo Carvalho 60012.

Excessively long identifiers Martim Costa 64901

Location: \ganttproject\src\main\java\biz\ganttproject\impex\csv\GanttCVSExport

This class has variables with excessively long identifiers, making it harder to read and use.

My suggestion is to reduce the size of the identifiers, using naming conventions of software architecture, reducing the effort to read and understand the code, and enable code reviews to have more focus on more important issues than naming and syntax standards. One example could be deleting the “Human” and using only a “H” and then comment above what the “H” means.



Reviewers:

- Reviewed by Rodrigo Mesquita 57372.

Constructor with too many arguments Martim Costa 64901

This constructor has too many arguments (10), making it hard to debug, and probably meaning that this Class is doing more than it should.

One possible way to improve this is to define a class which holds these parameter values and re-phrase the constructor to accept the class instance as a parameter instead of individually requesting the parameters.

```
public GanttGraphicArea(GanttProject app, TaskManager taskManager, ZoomManager zoomManager,
                        GPUndoManager undoManager, TaskTableChartConnector taskTableChartConnector,
                        Supplier<TaskTableActionConnector> taskTableActionFacade) {
  super(app.getProject(), app.getUIFacade(), zoomManager);
  this.setBackground(Color.WHITE);
  this.taskTableChartConnector = taskTableChartConnector;
  this.taskTableActionFacade = taskTableActionFacade;
  myUndoManager = undoManager;

  myChartModel = new ChartModelImpl(getTaskManager(), app.getTimeUnitStack(), app.getUIConfiguration());
  myChartModel.addOptionChangeListener(this::repaint);
  myStateDiffOptions = createBaselineColorOptions(myChartModel, app.getUIConfiguration());
  //this.tree = ttree;
  myViewState = new ChartViewState( chart: this, app.getUIFacade());
  app.getUIFacade().getZoomManager().addZoomListener(myViewState);

  super.setStartDate(CalendarFactory.newCalendar().getTime());
  var timerBarrier = new TimerBarrier( intervalMillis: 1000);
  timerBarrier.await((unit) -> {
    SwingUtilities.invokeLater(this::reset);
    return Unit.INSTANCE;
  });
  taskManager.addTaskListener(createTaskListenerWithTimerBarrier(timerBarrier));
  myPublicHolidayDialogAction = new ProjectCalendarDialogAction(getProject(), getUIFacade());
  getProject().getTaskCustomColumnManager().addListener(this);
  initMouseListeners();
}
```

Reviewers:

- Reviewed by Rodrigo Mesquita 57372.

No Comments

Rodrigo Mesquita 57372

Smell Code Location:

\ganttproject\ganttproject\src\main\java\net\sourceforge\ganttproject\calendar\CalendarEditorPanel.java

This class has 513 lines of code with many methods for different purposes, but there are no comments to explain their functionalities, which makes it hard and exhaustive to understand the code to someone new or even to the original developer.

```
2 usages  dbarashev
private static void reload(GPCalendar calendar, List<CalendarEvent> events, TableModelImpl model) {
  int size = events.size();
  events.clear();
  model.fireTableRowsDeleted(0, size);
  events.addAll(Collections2.filter(calendar.getPublicHolidays(), recurring(model.isRecurring())));
  model.fireTableRowsInserted(0, events.size());
}

2 usages  dbarashev
public JComponent createComponent() {
  JTabbedPane tabbedPane = new JTabbedPane();
  tabbedPane.addTab(GanttLanguage.getInstance().getText( key: "calendar.editor.tab.oneoff.title"),
      createNonRecurringComponent());
  tabbedPane.addTab(GanttLanguage.getInstance().getText( key: "calendar.editor.tab.recurring.title"),
      createRecurringComponent());
  myOnCreate.run();
  return tabbedPane;
}

1 usage  dbarashev
private Component createRecurringComponent() {
  DateFormat dateFormat = GanttLanguage.getInstance().getRecurringDateFormat();
  AbstractTableAndActionsComponent<CalendarEvent> tableAndActions = createTableComponent(myRecurringModel, date
  JPanel result = AbstractTableAndActionsComponent.createDefaultTableAndActions(tableAndActions.getTable(), tab
```

```
1 usage  Dmitry Barashev +1
@Override
protected void onDeleteEvent() {
  var selected :int[] = table.getSelectedRows();
  Arrays.sort(selected);
  for (int i = selected.length - 1; i >= 0; i--) {
    var selectedRow :int = selected[i];
    if (selectedRow < tableModel.getRowCount() - 1) {
      tableModel.delete(selectedRow);
    }
  }
}

 dbarashev
@Override
protected CalendarEvent getValue(int row) { return tableModel.getValue(row); }
};

Function<List<CalendarEvent>, Boolean> isDeleteEnabled = events -> events.size() != 1 || events.get(0) != null;
tableAndActions.getDeleteItemAction().putValue(AbstractTableAndActionsComponent.PROPERTY_IS_ENABLED_FUNCTION, isDeleteEnabled);
return tableAndActions;
}

 dbarashev
public List<CalendarEvent> getEvents() {
  List<CalendarEvent> result = Lists.newArrayList();
  result.addAll(myOneOffEvents);
  result.addAll(myRecurringEvents);
  return result;
}
```


Reviewers:

- Reviewed by Pedro Arruda 60663.

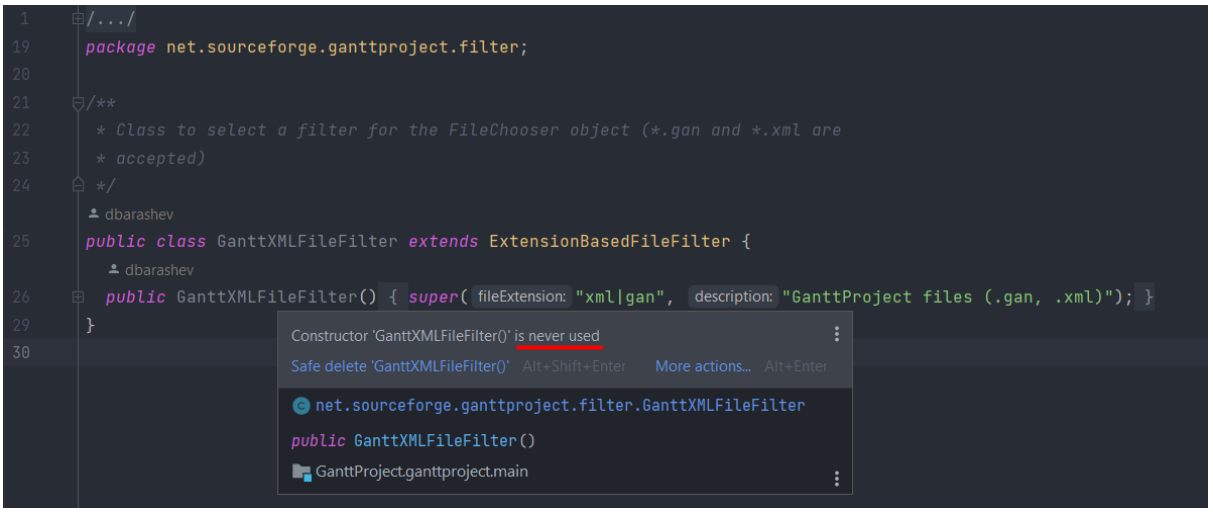
Dead Code

Rodrigo Mesquita 57372

Smell Code Location:

\ganttproject\ganttproject\src\main\java\net\sourceforge\ganttproject\filter\GanttXMLFileFilter.java

This is an example of Dead Code. In this case, the class and its constructor are never used



Reviewers:

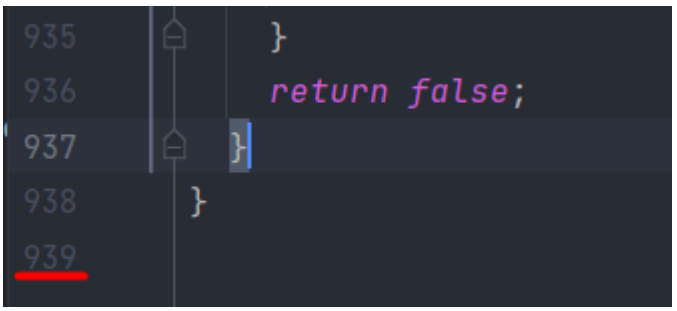
- Reviewed by Ana Gadelha 59943.

Large Class Rodrigo Mesquita 57372

Smell code location:

\ganttproject\ganttproject\src\main\java\net\sourceforge\ganttproject\task\algorithm\DependencyGraph.java

This is an example of a Large Class. This is a result of accumulating responsibilities that require extensive comments (another code smell) to document where in the code of the class certain functionalities exist.



Reviewers:

- Reviewed by Martim Costa 64901.

Second Phase

Codebase Metrics

Martin Package Metrics

Pedro Arruda 60663

Metrics:

- Abstractness (A)
- Afferent couplings (Ca)
- Efferent Couplings (Ce)
- Distance from the main sequence (D)
- Instability (I)

Introduction

In this report I will talk about the set of metrics that I thought would be useful to find code smells. This set of metrics is called the Martin Package Metrics and it is commonly used to assess the relationship between packages and identify poorly designed ones.

Abstractness (A)

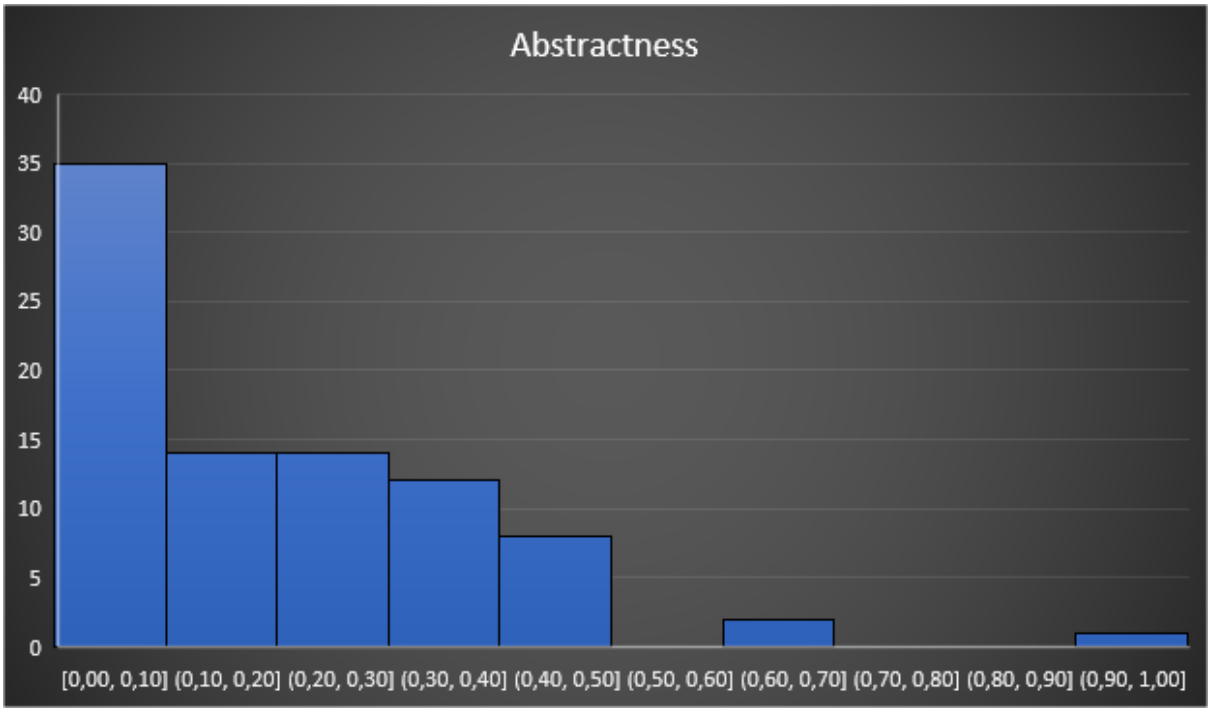
$$A = \frac{N_a}{N_c}$$

N_a Number of Abstract Classes
N_c Number of Classes

Abstractness is the ratio of the number of abstract classes and interfaces in the package to the total number of all classes. This metric is used to measure the degree of abstraction of the package.

Preferred values for the metric A should take extreme values close to 0 or 1:

- Having A = 0 means it is completely concrete.
- Having A = 1 means it is completely abstract.



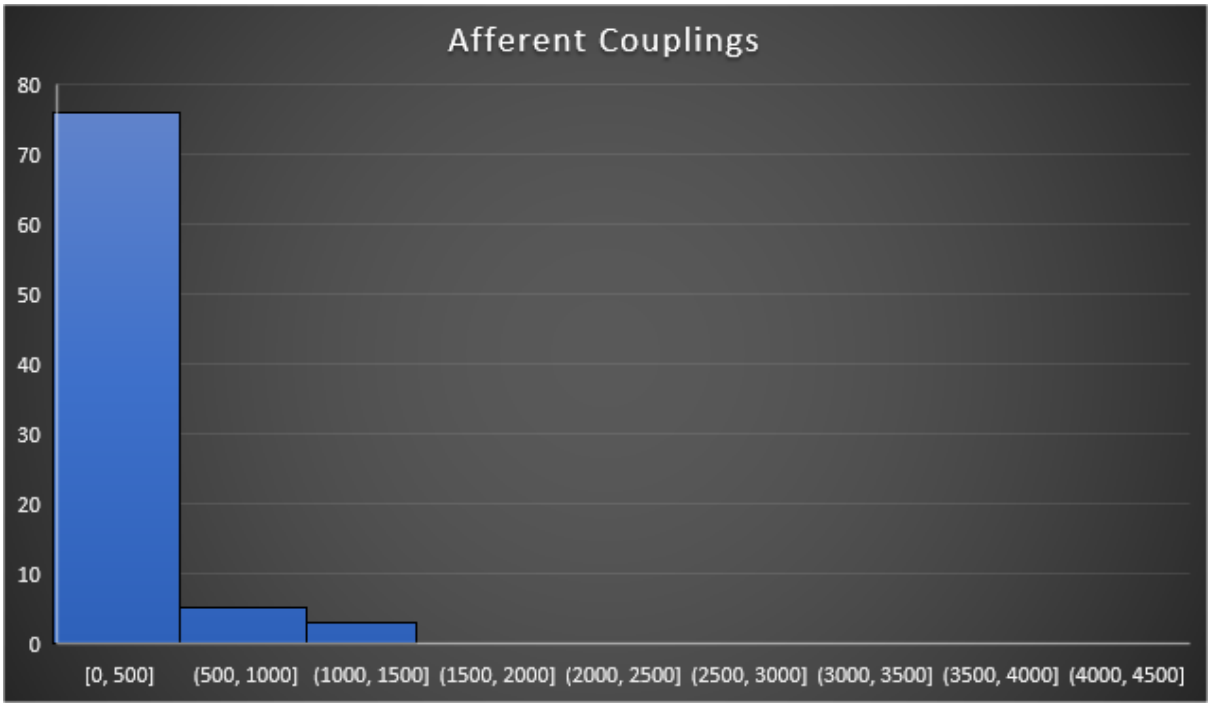
Abstractness Metric Histogram

Looking at the metric histogram, it is possible to see that the most common values are between 0 and 0.10, although, there are also a considerable number of packages with values between 0.10 and 0.50, which might not be a very good thing.

Afferent Couplings (Ca)

This metric calculates the number of Afferent Couplings for each package. An Afferent Coupling is a reference from a class or interface external to the package to a class or interface internal to the package.

- Having high values for Ca means a lot of classes outside the package depend on it.
- Having low values for Ca means few classes outside the package depend on it.



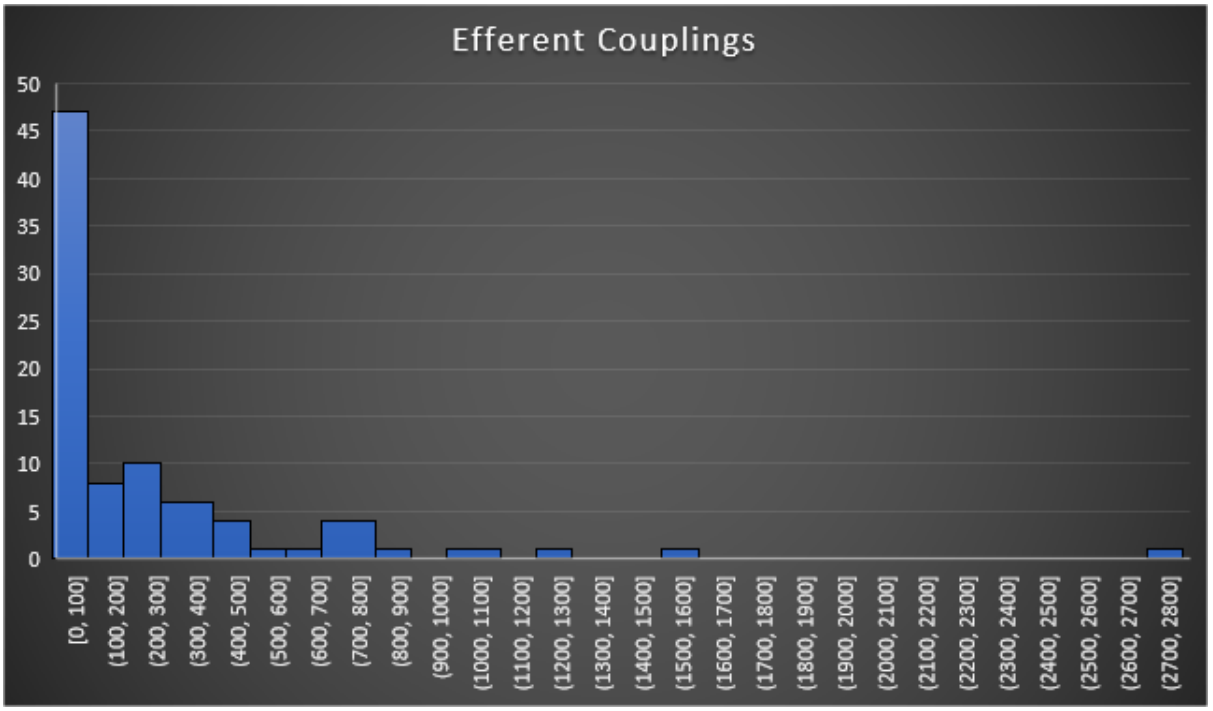
Afferent Couplings Metric Histogram

Looking at the metric histogram, it is possible to see that the most common values are between 0 and 500, which most likely means a good thing since the values are low.

Efferent Couplings (Ce)

Contrary to the Ca metric, Efferent Coupling is the number of classes outside the package on which the package depends upon.

- Having high values for Ce means the package depends on a lot of classes outside of it.
- Having low values for Ce means the package depends on few classes outside of it.



Efferent Couplings Metric Histogram

Looking at the metric histogram, it is possible to see that the most common values are between 0 and 100, which are low values, meaning a good thing. We can also see that more packages have “worse” values when compared to the Ca metric though. It might be worth it to pay attention to those packages so as to not let those values rise more.

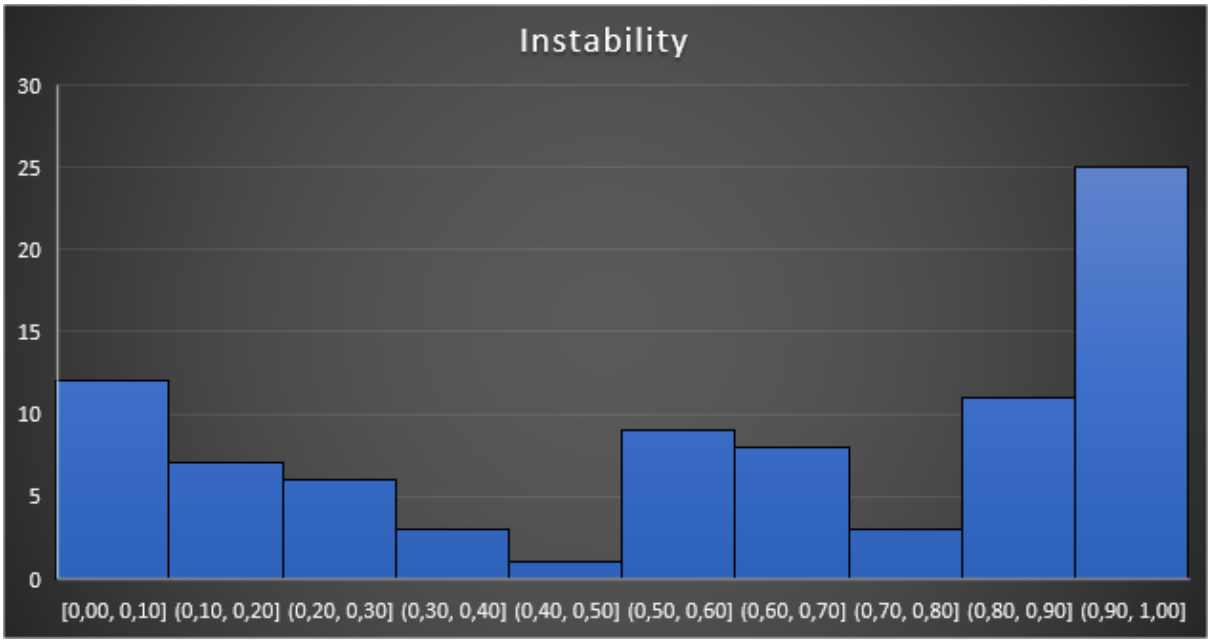
Instability (I)

$$I = \frac{Ce}{Ce + Ca}$$

This metric is used to measure the relative susceptibility of class to changes. According to the definition instability is the ratio of outgoing dependencies to all package dependencies.

Preferred values for the metric I are the same as for the A metric:

- Having I = 0 indicates a stable package.
- Having I = 1 indicates a maximally unstable package.

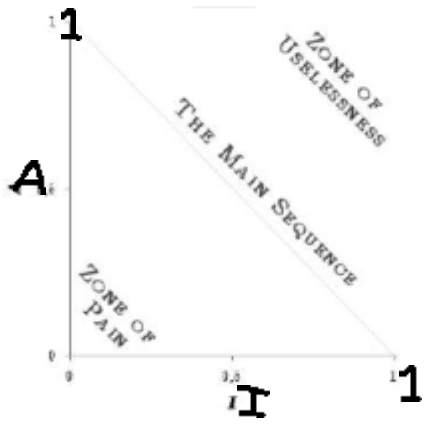


Instability Metric Histogram

Looking at the metric histogram, it is possible to see that the values are very distributed. The fact that there are a decent number of values in the middle might alert us to some poorly designed packages.

Distance from the main sequence (D) $D = |A + I - 1|$

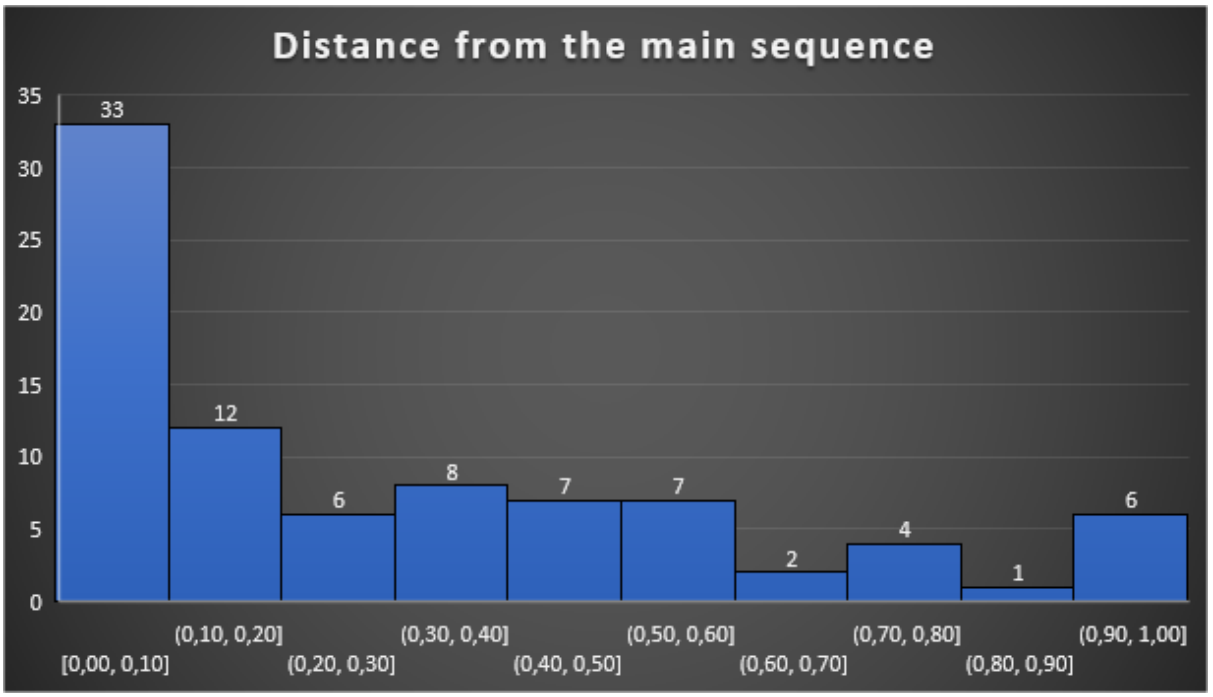
This metric can be considered the most important one, since it calculates the balance between stability and abstractness.



The IA-graph.

The value of D can be interpreted as the distance to the main sequence (which is the line that connects the points A=1; I=0 and A=0; I=1) so its value should be as low as possible.

A D closer to 1 means that the package is poorly designed.



Distance from the main sequence Metric Histogram

Conclusion

When analyzing the last histogram (related to the D metric), we can see that most of the packages seem to be very well designed.

Nevertheless, the number of packages close to the value of 1 led me to investigate these packages further to find out what could be poorly designed.

Martin Packaging Metrics	Abstractnes	Afferent Coupling	Efferent Coupling	Distance from the main sequence	Instability
net.sourceforge.ganttproject.filter	0,0000	9	0	1,0000	0,0000
net.sourceforge.ganttproject.font	0,0000	14	0	1,0000	0,0000
net.sourceforge.ganttproject.shape	0,0000	4	0	1,0000	0,0000
net.sourceforge.ganttproject.util.collect	0,0000	199	0	1,0000	0,0000
org.w3c.util	0,0000	36	0	1,0000	0,0000
net.sourceforge.ganttproject.chart.item	0,0000	71	8	0,8987	0,1013
net.sourceforge.ganttproject.task.hierarchy	0,0000	15	4	0,7895	0,2105
biz.ganttproject.core.chart.canvas	0,2105	766	0	0,7895	0,0000
net.sourceforge.ganttproject.util	0,1111	88	17	0,7270	0,1619
net.sourceforge.ganttproject	0,0000	21	8	0,7241	0,2759
net.sourceforge.ganttproject	0,2857	940	29	0,6844	0,0263
net.sourceforge.ganttproject.gui.zoom	0,2500	62	10	0,6111	0,0577

When looking at the packages with very high values of D (> 0.600) we can see that all of them also have very low values of both abstractness and instability, meaning that they are all either close or located in the “Zone of Pain” – refer to the IA-graph in the section above – meaning that these packages are very rigid. They are difficult to extend because of their lack of abstractness and hard to change because of their responsibilities.

This could be considered a code smell and should be investigated by either turning the packages less rigid or by reconsidering if these packages really are necessary.

Bibliography

- “A Validation of Martin’s Metric” by Sami Hyrynsalmi and Ville Leppänen;
- Future processing blog – “Object-oriented metrics by Robert Martin”;
- Wikipedia – “Software package metrics”

- **Reviewed by: Rodrigo Mesquita 57372**

Dependency Metrics

Ana Gadelha 59943

Introduction

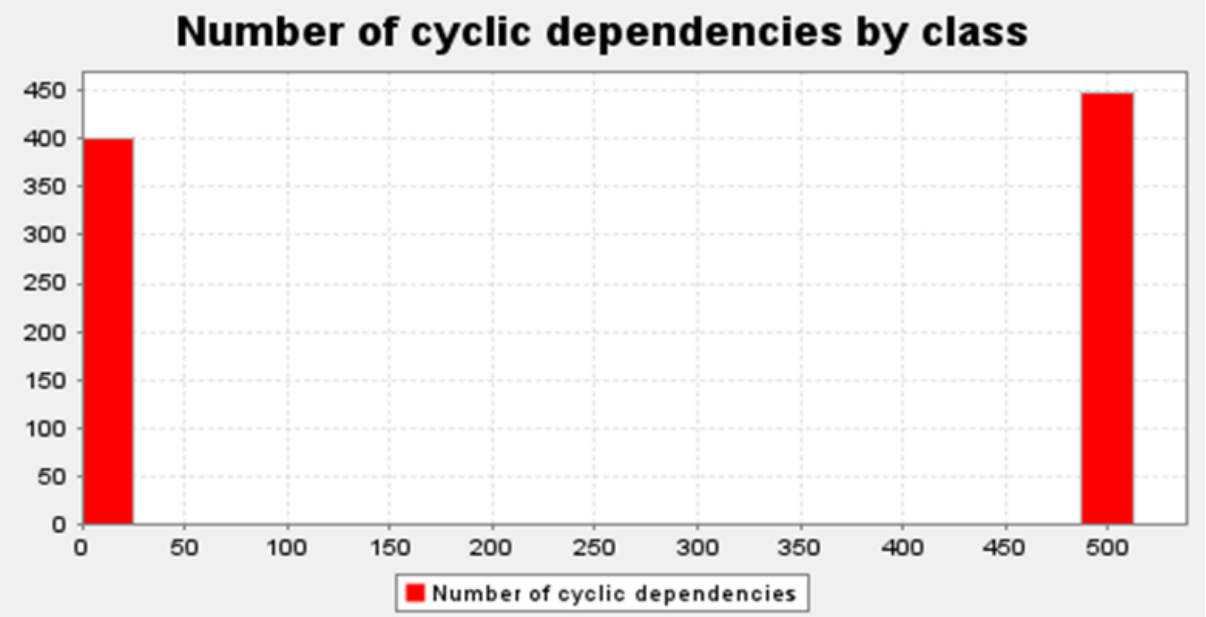
In this report I will analyze the different Dependency Metrics in the classes, because I feel that it has more relevant information then the interfaces, or the packages metrics.

Metrics:

- Cyclic - Number of Cyclic Dependencies
- Dcy - Number of Dependencies
- Dcy* - Number of Transitive Dependencies
- Dpt - Number of Dependents
- Dpt* - Number of Transitive Dependents
- PDcy - Number of Package Dependencies
- PDpt – Number of Dependent Packages

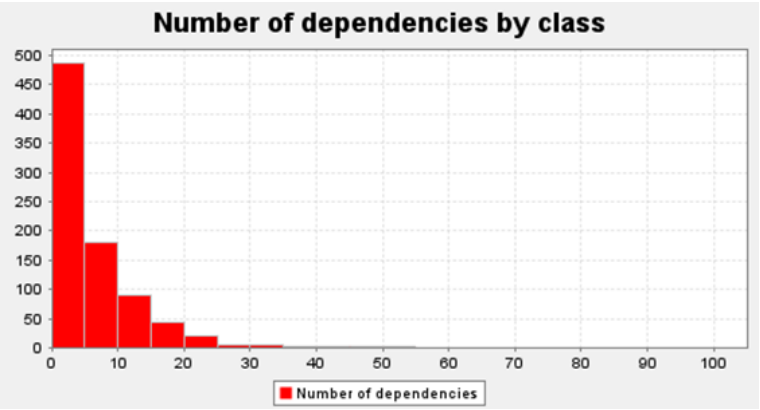
Number of Cyclic Dependencies

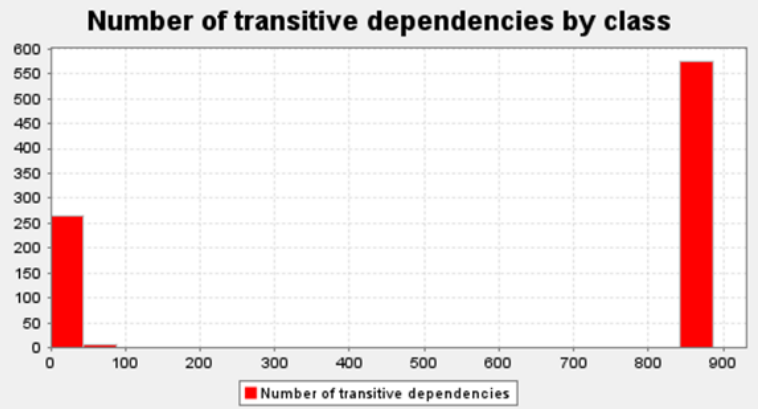
- This dependency calculates the number of classes or interfaces that depend directly or indirectly on another that in turn depends directly or indirectly on the first one. This type of cyclic dependency can result in code that is difficult to test and understand.
- In this histogram we can see that most values are either between 0 and 5 or around 500 which shows that there are a lot of classes that depend on one another, which can become a code smell.



Number of Dependencies and Transitive Dependencies

- These dependencies calculate the number of classes or interfaces which each class directly depends on.
- The Transitive dependencies differ from the normal ones because these also count the number of classes or interfaces which each class indirectly depends on.

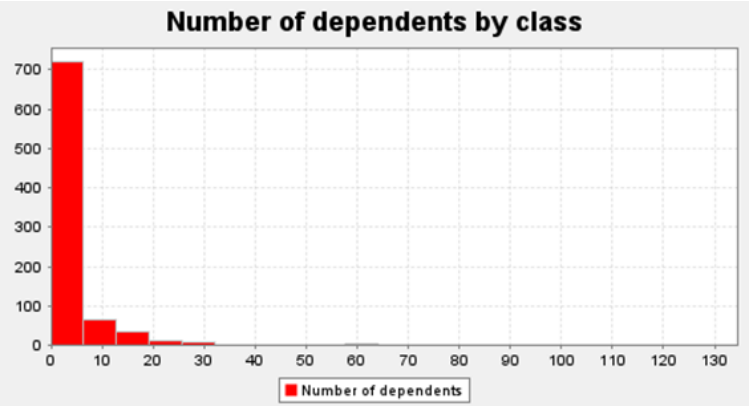


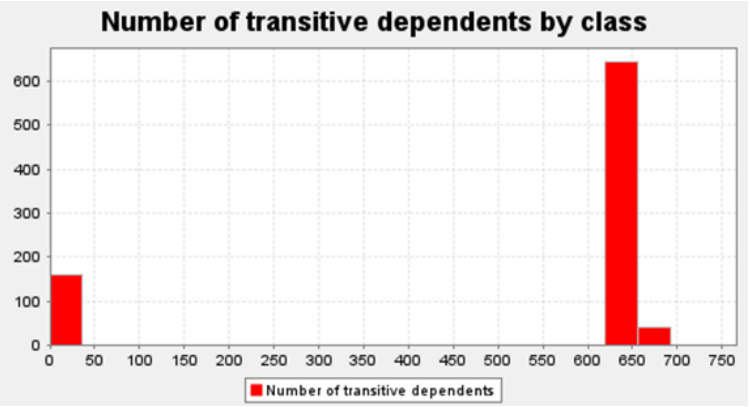


- In the first histogram we can see that the number of dependencies varies mostly between 0 and 10 and then we have a few more between 10 and 25 and the remaining from 30 until one with 100.
- In the second histogram we have the number of transitive dependencies that varies from 0 to 50, in a good number of classes and from 50 to 60, in some classes, and then between 873 and 886 in the remaining.

Number of Dependents and Transitive Dependents

- This dependency calculates the number of classes or interfaces which directly depend on each class.
- Once more the difference between this and the transitive version is that the transitive dependents also count the number of classes or interfaces which indirectly depend on each class.

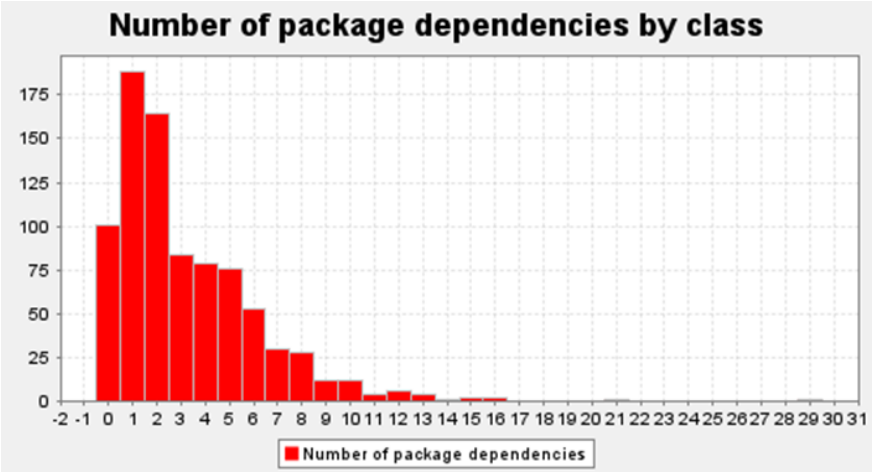




- In the first histogram the values vary mostly between 0 and 6 with some exceptions afterwards between 7 and 96 and with one class that has 128 dependents.
- In the second one we have values between 0 and 43 dependents per class and then there is a jump to values between 642 and 692, and once more there is one class that has a number of 729 dependents.
- By comparing the two we can see that there are drastic changes when we had the indirectly dependents to the count.
- Having too many dependents is a sign of coupled code, which can lead to code smells like the Shotgun Surgery, which basically is when we want to make a change to the code in one place and that change causes a cascade of alterations in other places.

Number of Package Dependencies

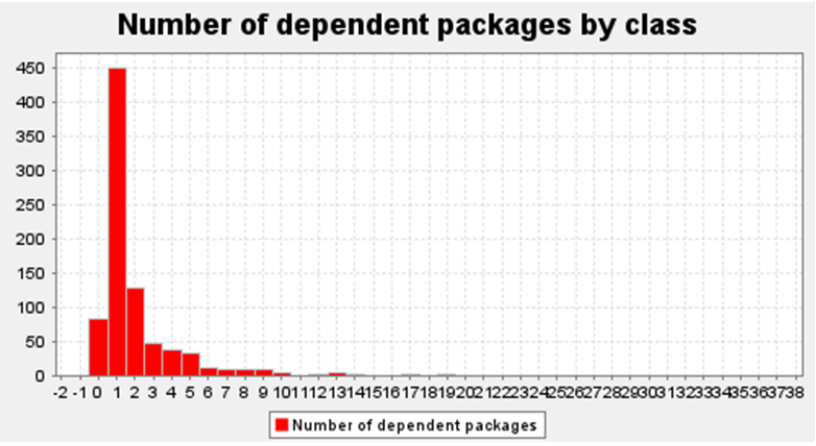
- This dependency calculates the number of packages on which class directly or indirectly depends on.



- In this histogram the values vary mostly between 0 and 10 per class, with some exceptions afterwards of values between 10 and 29.

Number of Dependent Packages

- This dependency calculates the number of packages which directly or indirectly depend on each class.



- In this histogram we can see that the values vary mostly between 0 and 3 but there are classes with values between 3 and 36.
- In comparison with the package dependencies there are not that many changes in the numbers range, only on the quantity of each dependency in each class (the package dependencies are less).

Dependency Metrics and Code Smells

- Too many dependencies between classes, interfaces or packages can be a sign of strong coupling, which we do not want. Low coupling is preferred so that the classes, interfaces, or packages can be as independent as possible.
- This level of coupling can cause code smells like:
    - **Shotgun Surgery** - like mentioned before.
    - **Inappropriate Intimacy** - that is when two or more classes are interlinked with each other too much (for example, by having public fields instead of getters and setters, or public fields that should be private).
    - **Feature Envy** – When a method of one class uses too much the functionalities of another class.

Relation Between the Metrics and the Found Code Smells

- We did not identify any of these types of code smells, but for sure that with the access to these values it would have been easier to find them in the code.

Lines Of Code Metrics

Rodrigo Mesquita 57372

Metrics:

- Comment Lines of Code (CLOC)
- Javadoc Lines of Code (JLOC)
- Lines of Code (LOC)
- Non-Comment Lines of Code (NCLOC)
- Relative Lines of Code (RLOC)

Introduction

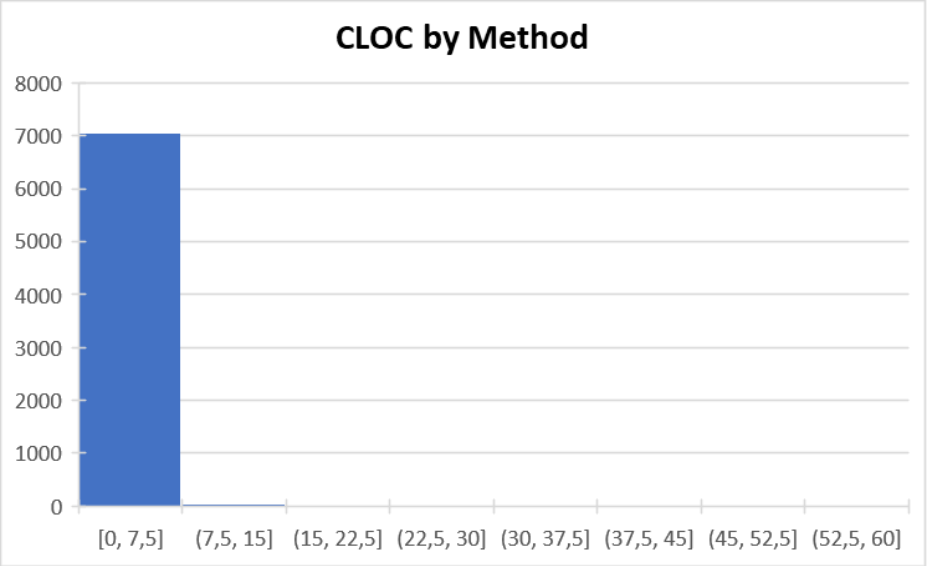
Lines of Code (LOC) metrics' purpose is to measure the size (number of lines) of files containing a computer programming language. Each metric's result helps to understand certain aspects from the written code, for example, too large methods and/or classes, classes/methods that are barely commented on, etc.

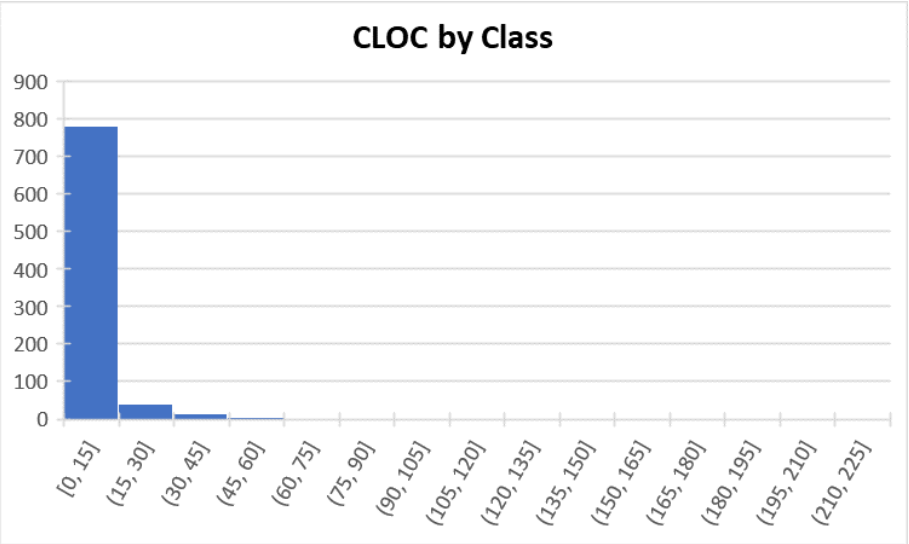
In this document, I will only consider the results obtained through this set of metrics in classes and methods because that is where it would be useful to find one with massive amounts of lines of code or comments.

Comment Lines of Code (CLOC)

Calculates the number of lines of code which contain comments. Whitespace lines are not counted.

- It is usually not good to get big numbers here.
- Very useful for classes and methods.





Thresholds: By analyzing the metric histograms, it's correct to state that the most common number of comments per method are from 0 to 15. In classes, there are from 0 to 45 comments per class. Thus, our thresholds for methods and classes are 20 and 60, respectively

method	CLOC
net.sourceforge.ganttproject.print.PrintPreview.PrintPreview(IGanttProject, UIFacade, Ch	56
net.sourceforge.ganttproject.gui.GanttDialogPerson.getComponent()	28
net.sourceforge.ganttproject.GanttOptions.doSave(OutputStream)	27
net.sourceforge.ganttproject.gui.TextFieldAndFileChooserComponent.showFileChooser(	24
net.sourceforge.ganttproject.GanttTreeTableModel.getColumnName(int)	20
net.sourceforge.ganttproject.chart.SimpleRenderedImage.copyData(WritableRaster)	20

Very few methods exceed our threshold and should be reviewed (specially the first one) to possibly detect an unnecessarily large comment.

class	CLOC
net.sourceforge.ganttproject.chart.SimpleRenderedImage	220
net.sourceforge.ganttproject.GanttOptions	189
net.sourceforge.ganttproject.language.CharSetMap	185
net.sourceforge.ganttproject.GanttProject	129
net.sourceforge.ganttproject.GanttTreeTableModel	124
net.sourceforge.ganttproject.util.BrowserControl	104
org.ganttproject.chart.pert.ActivityOnNodePertChart	89
net.sourceforge.ganttproject.print.PrintPreview	65
net.sourceforge.ganttproject.gui.options.SpringUtilities	63
net.sourceforge.ganttproject.gui.TextFieldAndFileChooserComponent	62

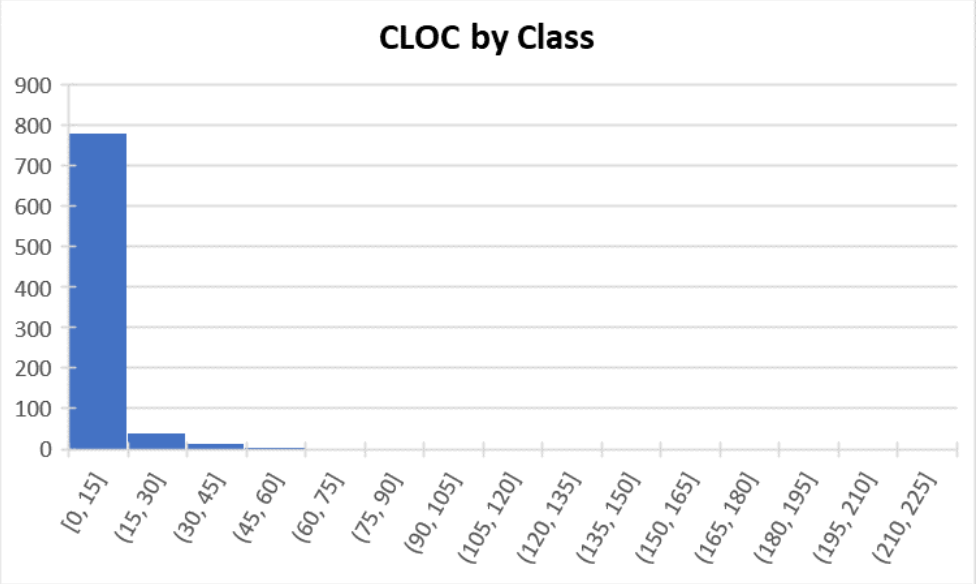
The same goes for some classes that exceeded the threshold.

Javadoc Lines of Code (JLOC)

Calculates the number of lines of code which contain javadoc comments. Whitespace lines are not counted.

method	JLOC
net.sourceforge.ganttproject.gui.options.SpringUtilities.makeCompactGrid(Container, ir	20
net.sourceforge.ganttproject.gui.options.SpringUtilities.makeGrid(Container, int, int, int,	19
net.sourceforge.ganttproject.chart.SimpleRenderedImage.copyData(WritableRaster)	17
net.sourceforge.ganttproject.chart.SimpleRenderedImage.getData(Rectangle)	17
net.sourceforge.ganttproject.chart.SimpleRenderedImage.getData()	16
net.sourceforge.ganttproject.util.DateUtils.parseDate(String)	15

As we can see, the method with the most lines of Javadoc are 20 lines, which in my opinion is not a problem because for some complicated methods it's for the best interest to have a good and detailed resume about it.



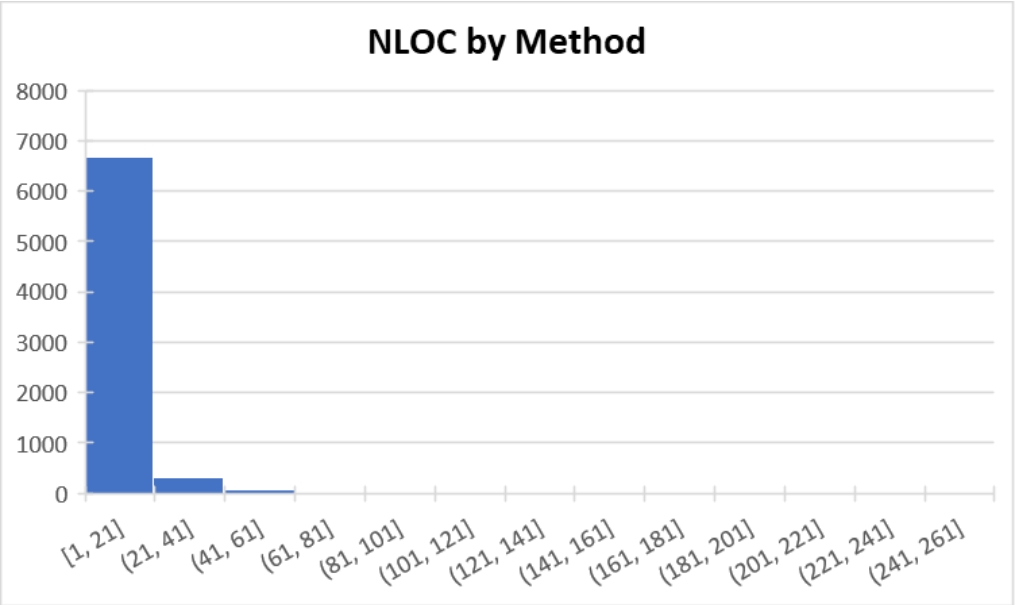
Threshold: the most common number of Javadoc lines per class are from 0 to 45, which gives us a threshold of 60.

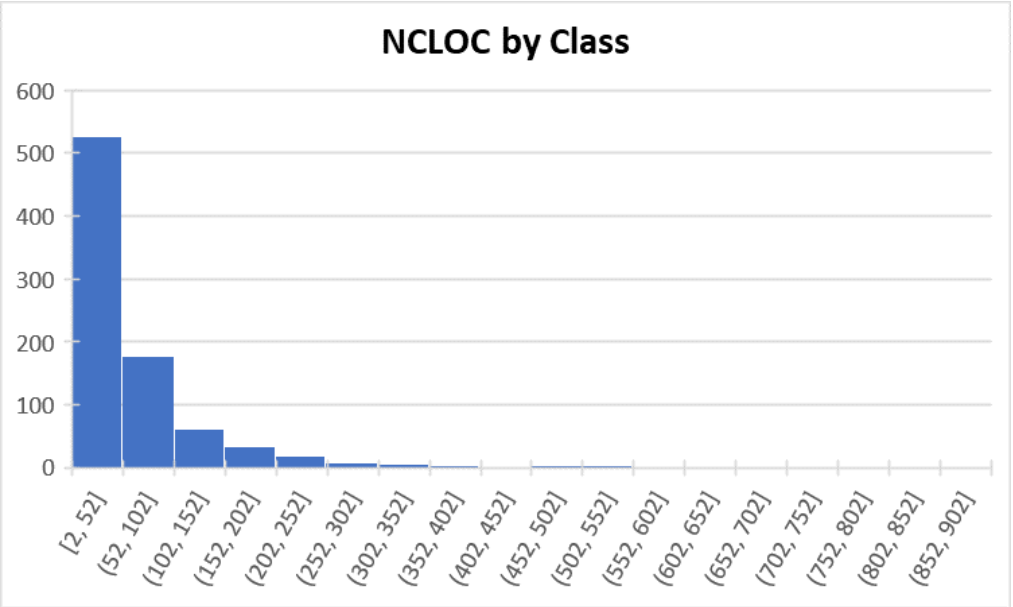
class	JLOC
net.sourceforge.ganttproject.chart.SimpleRenderedImage	207
net.sourceforge.ganttproject.language.CharSetMap	173
net.sourceforge.ganttproject.GanttOptions	149
net.sourceforge.ganttproject.GanttProject	66
org.ganttproject.chart.pert.ActivityOnNodePertChart	58

There are a few excessive cases that can mean the existence of some code smells, for example, large classes. Definitely something that should be checked.

Lines of Code (LOC) and Non-comment Lines of Code (NCLOC)

The LOC metric seems to be kind of useless in this case since we have already looked into CLOC and JLOC. Thus, I will replace LOC with NCLOC.





Thresholds: with the help of the graphs above I will use a threshold of 80 for methods and 400 for classes.

method	NCLOC
net.sourceforge.ganttproject.print.PrintPreview.PrintPreview(IGanttProject, UIFacade, Ch	258
net.sourceforge.ganttproject.GanttOptions.GanttXMLOptionsParser.startElement(String,	205
net.sourceforge.ganttproject.GanttProject.GanttProject(boolean)	199
net.sourceforge.ganttproject.gui.tableView.ColumnManagerPanel.createComponent()	139
net.sourceforge.ganttproject.chart.ChartModelBase.ChartModelBase(TaskManager, Tim	132
net.sourceforge.ganttproject.chart.overview.ToolbarBuilder.addComboBox(Action[], Ac	117
net.sourceforge.ganttproject.parser.TaskTagHandler.loadTask(Attributes)	115
net.sourceforge.ganttproject.DialogBuilder.createDialog(Component, Action[], String, N	110
org.ganttproject.impex.htmlpdf.XmlSerializer.writeTasks(TaskManager, TransformerHar	110
net.sourceforge.ganttproject.gui.options.model.CustomPropertyDefaultValueAdapter.cr	109
net.sourceforge.ganttproject.GanttOptions.doSave(OutputStream)	107
org.w3c.util.DateParser.getCalendar(String)	106
net.sourceforge.ganttproject.GanttProject.main(String[])	103
net.sourceforge.ganttproject.GanttProject.createToolbar()	102
net.sourceforge.ganttproject.GanttTreeTableModel.getValueAt(Object, int)	101
net.sourceforge.ganttproject.document.webdav.GanttURLChooser.createComponent()	101
net.sourceforge.ganttproject.CustomPropertyManager.PropertyTypeEncoder.decodeTyp	99
net.sourceforge.ganttproject.document.webdav.WebDavOptionPageProvider.buildPage	98
biz.ganttproject.core.time.impl.GPTimeUnitStack.parseDuration(String)	95

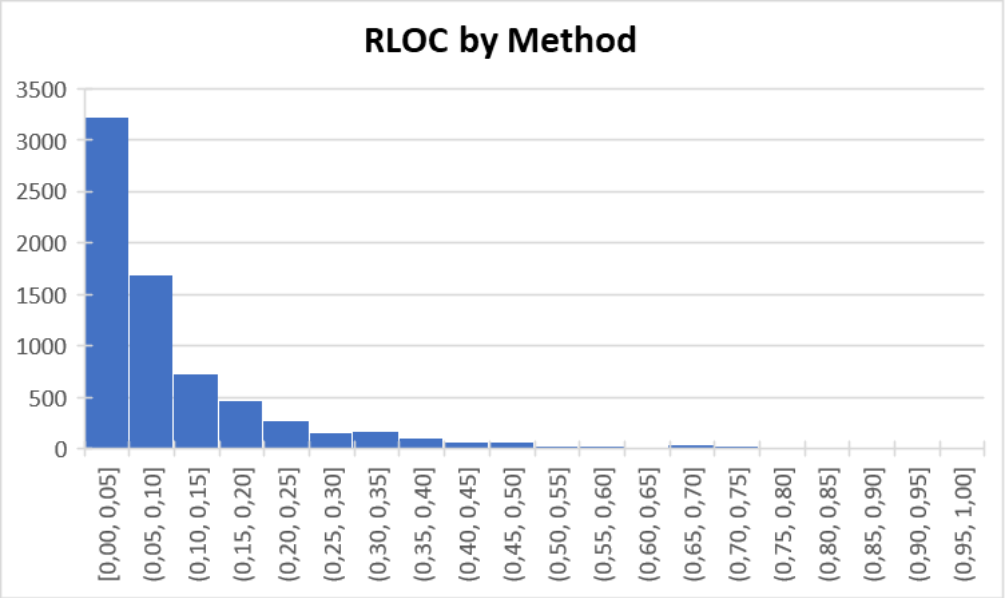
It is undoubtedly necessary to review some methods because they will most likely have a few code smells, especially the long methods one.

class	NCLOC
net.sourceforge.ganttproject.GanttProject	889
net.sourceforge.ganttproject.task.TaskManagerImpl	755
net.sourceforge.ganttproject.task.TaskImpl	603
biz.ganttproject.impex.msproject2.ProjectFileImporter	600
net.sourceforge.ganttproject.gui.options.OptionsPageBuilder	541
org.ganttproject.chart.pert.ActivityOnNodePertChart	525
net.sourceforge.ganttproject.UIFacadeImpl	519
org.ganttproject.impex.htmlpdf.itext.ThemeImpl	501
net.sourceforge.ganttproject.GanttOptions	500
net.sourceforge.ganttproject.GPTreeTableBase	499
net.sourceforge.ganttproject.chart.ChartModelBase	479
net.sourceforge.ganttproject.gui.GanttTaskPropertiesBean	440
net.sourceforge.ganttproject.document.webdav.GanttURLChooser	415
net.sourceforge.ganttproject.print.PrintPreview	400

My opinion for the classes stands the same as for methods.

Relative Lines of Code (RLOC)

Calculates ratio of lines of code for a method to the lines of code for it's containing class, that is, if a method is really close to 100% then the whole class is probably only one method, and such class shouldn't exist. On the other hand, if a method is 0%, the method is empty and should be removed.



Threshold: Due to the histogram results, the threshold for this metric will be 70.

method	RLOC
net.sourceforge.ganttproject.gui.options.model.CustomPropertyDefaultValueAdapter.cr	98,20%
net.sourceforge.ganttproject.document.webdav.WebDavProxyTest.testProxyOptionPars	95,45%
getActivities(Date, Date)	95,00%
biz.ganttproject.core.time.TimeUnitStack.Util.findCommonUnit(TimeUnit, TimeUnit)	92,59%
net.sourceforge.ganttproject.io.GanttChartViewSaver.save(ColumnList, TransformerHan	92,59%
biz.ganttproject.core.table.ColumnList.Immutable.fromList(List<Column>)	92,31%
net.sourceforge.ganttproject.customProperty.CustomPropertyImportTest.testImportDup	92,31%
compare(Task, Task)	92,00%
compare(TaskActivity, TaskActivity)	92,00%
paint(Rectangle)	91,49%

As we can see there are some classes that might only have one method. These classes should be checked and possibly modified.

Code Smells and Lines of Code metrics

After using the Lines of Code metrics, I've concluded that they are useful to find code smells, for example: long methods, too many comments, no comments, large classes, data classes, speculative generality, etc.

While doing this report I found many classes with no comments, methods with more than 200 lines, classes with more than 1000 lines (with the white spaces lines included), comments that were full methods (probably a reminder or a method to be fixed).

Overall, I'm satisfied with the efficiency of these metrics to find code smells.

- **Reviewed by: Pedro Arruda 60663**

Mood Metrics

Bernardo Carvalho 60012

Metrics

- Method Hiding Factor (MHF)
- Attribute Hiding Factor (AHF)
- Method Inheritance Factor (MIF)
- Attribute Inheritance Factor (AIF)
- Polymorphism Factor (PF)
- Coupling Factor (CF)

Introduction

In this report I'll be studying the data collected from the MOOD metrics regarding our project to see if there are any anomalies or if everything is between the expected ranges for each category.

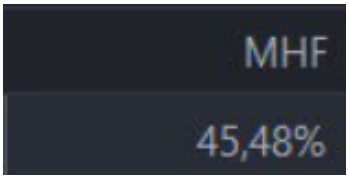
Method Hiding Factor

Method hiding factor measures how methods are encapsulated in a class. Visibility is counted in respect to other classes. MHF represents the average amount of hiding among all classes in the system. A private method is fully hidden.

> MHF = 1 − MethodsVisible
> MethodsVisible = sum(MV) / (C − 1) / Number of methods
> MV = number of other classes where method is visible
> C = number of classes

For each method, MV is counted.

If all methods are private, MHF=100% . If all methods are public, MHF=0%



A low MHF indicates insufficiently abstracted implementation. A large proportion of methods are unprotected and the probability of errors is high. To contrast, a high MHF indicates very little functionality. It may also indicate that the design includes a high proportion of specialized methods that are not available for reuse.

It is considered that a MHF between 10% and 25% is Ideal so as we can see ours is a considerable amount over those values which may indicate less functionality than we would want.

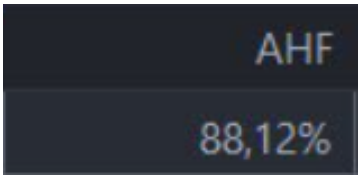
Attribute Hiding Factor

Attribute hiding factor is the same as MHF, but with attributes, and so, it measures how variables are encapsulated in a class. Visibility is also counted in respect to other classes. AHF represents the average amount of hiding among all classes in the system. A private attribute is fully hidden.

> AHF = 1 − AttributesVisible
> AttributesVisible = sum(AV) / (C − 1) / Number of attributes
> AV = number of other classes where attribute is visible
> C = number of classes

For each attribute, AV is counted.

If all attributes are private, AHF=100%. If all attributes are public, AHF=0%.



Ideally, most attributes should be hidden, and thus AHF should always be very high, nothing under 75%. Low values of AHF should trigger attention.

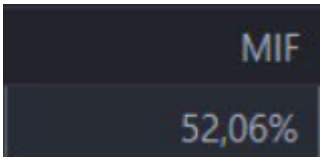
As we can see, although our AHF isn't 100%, it is quite high and thus it looks like everything is in order.

Method Inheritance Factor

Once again both MIF and AIF work in similar ways, but (again) one with methods and the other with attributes. For this reason, both will have similar explanations.

A class that inherits lots of methods from its ancestor classes contributes to a high MIF. A child class that redefines its ancestors' methods and adds new ones contributes to a lower MIF. An independent class that does not inherit and has no children contributes to a lower MIF.

MIF = inherited methods / total methods available in classes



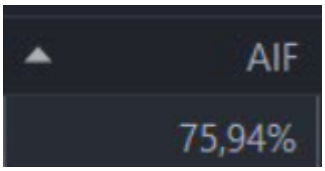
It is said that MIF should be in a reasonable range, not too low nor too high. Too high of a value indicates either superfluous inheritance or too wide member scopes. A low value indicates lack of inheritance or heavy use of Overrides/Shadows.

As we can observe, our MIF is within a good balance between too much inheritance and too little.

Attribute Inheritance Factor

A class that inherits lots of attributes from its ancestor classes contributes to a high AIF. A child class that redefines its ancestors' attributes and adds new ones contributes to a lower AIF. An independent class that does not inherit and has no children contributes to a lower AIF.

AIF = inherited attributes / total attributes available in classes

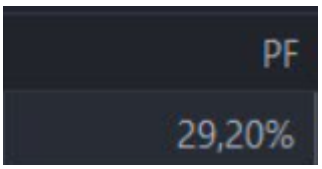


Although AIF behaves very similarly to MIF, it is believed that its range should be significantly lower than MIF, most variables should be declared private and so we can conclude that ours is quite a bit higher than it should be.

Polymorphism Factor

Polymorphism Factor measures the degree of method overriding in the class inheritance tree. It equals the number of actual method overrides divided by the maximum number of possible method overrides. Simply put, PF is the "Overrides factor". The more you use the Overrides keyword, the higher PF.

PF = overrides / sum for each class(new methods * descendants)



There is some conflict between the accepted range for PF, but either way this measurement usually sits at low percentages close to 10% or lower so it is safe to assume that ours is, once again, above usual.

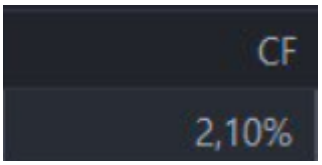
Coupling Factor

Coupling Factor measures the actual couplings among classes in relation to the maximum number of possible couplings.

CF = Actual couplings / Maximum possible couplings

Class A is *coupled to* class B if A calls methods or accesses variables of B.

If no classes are coupled, CF = 0%. If all classes are coupled to all other classes, CF=100%.



An acceptable range for CF is usually between 0 to 11% so ours is well within limits. This means the project does not couple too many classes without reason.

Conclusion

According to MOOD metrics our project has a bit too many overrides and inherited attributes. Furthermore it also has too many private methods leading to less functionality. Although these bad qualities have no specific relation to the code smells we found, they may be harmful in other ways.

The positive side is that the rest of the measurements seem to be within according ranges. The project hides most attributes, as it should, inherits a good balance of methods and couples a good percentage of classes.

Bibliography

- Mood Factors to Assess a Java Program - javatpoint
- MOOD and MOOD2 metrics - Aivosto

- **Reviewed by Ana Gadelha 59943**

Complexity Metrics

Martim Costa 64901

Brief Introduction

In this report I will analyze the Complexity Metrics of the GanttProject, in methods, classes, packages, modules and in the entire project. The metrics are:

- Cogc: Cognitive Complexity

- ev(G): Essential Cyclomatic Complexity

- iv(G): Design Complexity

- v(G): Cyclomatic Complexity

- OCavg: Average Operation Complexity

- OCmax: Maximum Operation Complexity

- WMC - Weighted Method Complexity

- v(G)avg - Average Cyclomatic Complexity

- v(G)tot - Total Cyclomatic Complexity

I will give more importance to the **Cognitive Complexity** and **Cyclomatic Complexity** since the others are strongly related to these two.

Cognitive, Essential Cyclomatic, Design and Cyclomatic Complexities

- Cognitive Complexity (Cogc) is a measure of how difficult a unit of code is to intuitively understand or in other words tells you how difficult your code will be to read and understand. At a method level, 15 for a Cognitive Complexity value is a recommended maximum.

A method's cognitive complexity is based on a few simple rules:

· Code is not considered more complex when it uses shorthand that the language provides for collapsing multiple statements into one.

· Code is considered more complex for each "break in the linear flow of the code". For example loops, switch or case statements, conditionals and others.

· Code is considered more complex when "flow breaking structures are nested". For example conditionals, loops and try/catch blocks.

All of this increases the value of **Cognitive Complexity**.

- Cyclomatic Complexity (v(G)) effectively measures the number of possible independent paths through a method or function. This tells us how complex the method is to test and debug for example. Higher values of **Cyclomatic Complexity** result in the need for a higher number of test cases to comprehensively test a part of code or a function, and if you add that to a more complex part of code, it increases the chances of you adding a defect to the code. Some examples of what increases the **Cyclomatic Complexity** are if/else statements, switch case statements 'try-catch' statements among others. Usually this type of complexity is divided by 4 levels:

- 1 - 10 Simple procedure, little risk
- 11 - 20 More complex, moderate risk
- 21 - 50 Complex, high risk
- > 50 Untestable code, very high risk

The risk is related to how hard it is to test it and how high are the chances of committing a defect.

All the methods start with value 1.

- Essential Cyclomatic Complexity (ev(G)) tells how much complexity is left once we have removed the well-structured complexity**. Essential Cyclomatic Complexity** and **Cyclomatic Complexity** are related.

- Design Complexity (iv(G)) is related to how interlinked a method's control flow is with calls to other methods. It also represents the minimal number of tests necessary to exercise the integration of the method with the methods it calls.

	CogC	ev(G)	iv(G)	v(G)
Total	7741	7516	9896	11194
Average	1.194968	1.22013	1.606494	1.817208

Table 1-Methods

As you can see, at the level of methods (more than 6500) the average for all complexities is under 2, which is a very good average, way below the values considered minimum.

Below are the tables that are related to Packages, Modules and the whole Project. As you can see the values of **Cyclomatic Complexity** referred to the methods are represented in the other tables, which have similar values. **v(G)avg** is **Average Cyclomatic Complexity** and **v(G)tot** is the **Total Cyclomatic Complexity**.

	v(G)avg	v(G)tot
Total	304.559395	11259
Average	1.81538213	135.6506

Table 2- Packages

	v(G)avg	v(G)tot
Total	306.374777	11259
Average	1.81538213	1608.429

Table 3 – Modules

PROJECT METRICS		
	v(G)avg	v(G)tot
Project	1.82	11259

Table 4- Whole Project

Average Operation, Maximum Operation, Weighted Method Complexities

- Average Operation Complexity (OCavg) is the Average Cyclomatic Complexity of all non-abstract methods in each class. Inherited methods are not counted.

- Maximum Operation Complexity (OCmax) is the Maximum Cyclomatic Complexity of all non-abstract methods in each class. Inherited methods are not counted.

- Weighted Method Complexity (WMC) is the total cyclomatic complexity of the methods of each class.

	OCavg	OCmax	WMC
Total	2638.13509	5037.066	11228
Average	1.6007984	3.065693	13.20941

Table 5 – Classes

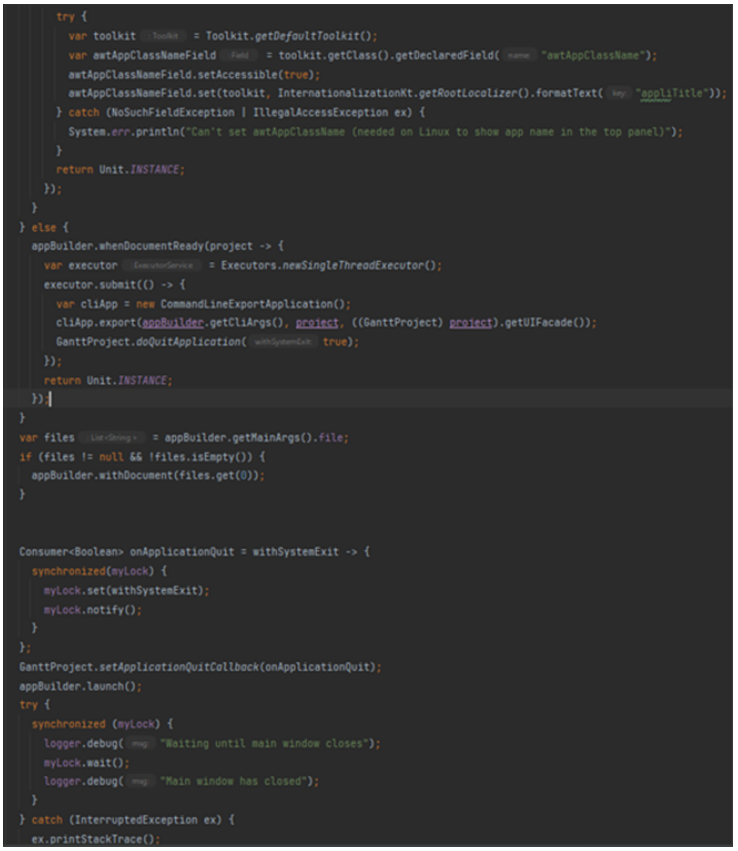
As this table shows, the Average of the Average **Cyclomatic Complexity** of the methods of each class don't differ much to those shown above in **Table 1**. Even the Average of the Maximum **Cyclomatic Complexity** value is around 3 so even the methods with higher values have values that are considered in the acceptable range.

Relation between Code Smells and Complexity Metrics

Of course Complexity Metrics are related directly with code smells.

When a method has high values of **Cognitive Complexity** it means that it is a method that is hard to read and understand. That is a code smell because, maybe for the method's creator it is easy to read it, it's hard for others to understand it. And this increases the chances of others that want to work in that part of the code to commit defects and errors. This also makes the code harder to be expandable which is a very important part of a good coding practice.

Same problem with **Cyclomatic Complexity**. Hard testing code makes it harder to work on, and increases the chances of defects and errors. For example, in a code smell detected by me(Martim Costa 64901) we can see a **Huge function** with a lot of if/else statements and try/catch statements which means that there are a high number of possible independent paths through the function. To test that would take a lot of time and patience. Here is a print of part of that function.



I also want to mention that, although that's not always the case, **Cyclomatic Complexity** can be one of the factors driving up **Cognitive Complexity** and vice versa.

Conclusion

Analyzing and according to this data, I can conclude that this Project in general in terms of understanding and reading it is easy to do so. I can also say that about the difficulty to test this Project the values of **Cyclomatic Complexity** are below the value that are considered acceptable (usually 10 is the maximum value to be considered that).

Of course some improvements can be done, some methods have high values of **Cognitive Complexity** and **Cyclomatic Complexity**. Some ways to reduce **Cognitive Complexity** are :

-Avoiding multiple if-else or nested if-else statements.

-Move repeated Code/nested if else to a separate function.

-Reduce the number of parameters of the method. It is always good to have max 2–3 parameters. If it exceeds you can wrap all of them into a class and pass the object.

About **Cyclomatic Complexity**:

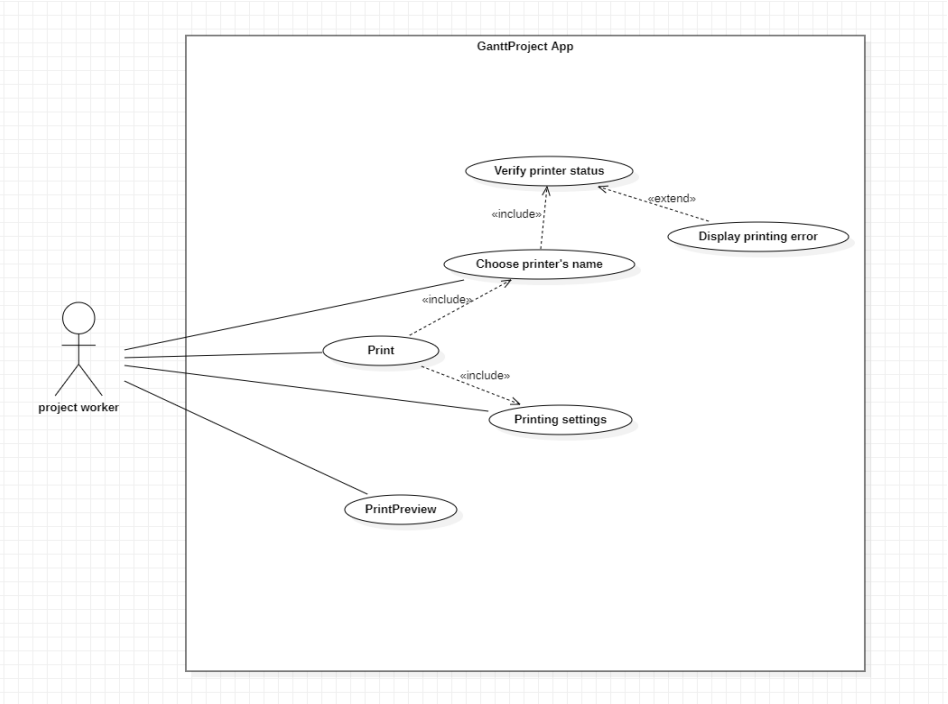
- Use small methods. Try reusing code wherever possible and create smaller methods which accomplish specific tasks.

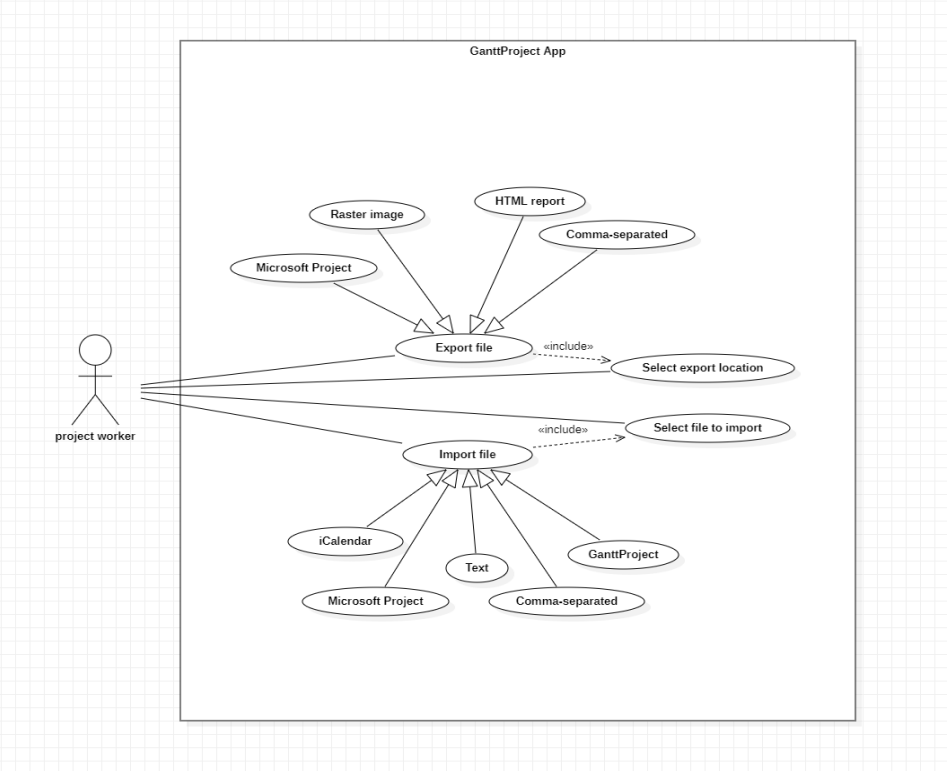
- Reduce if/else statements.

- **Reviewed by Bernardo Carvalho 60012**

Use case Diagrams

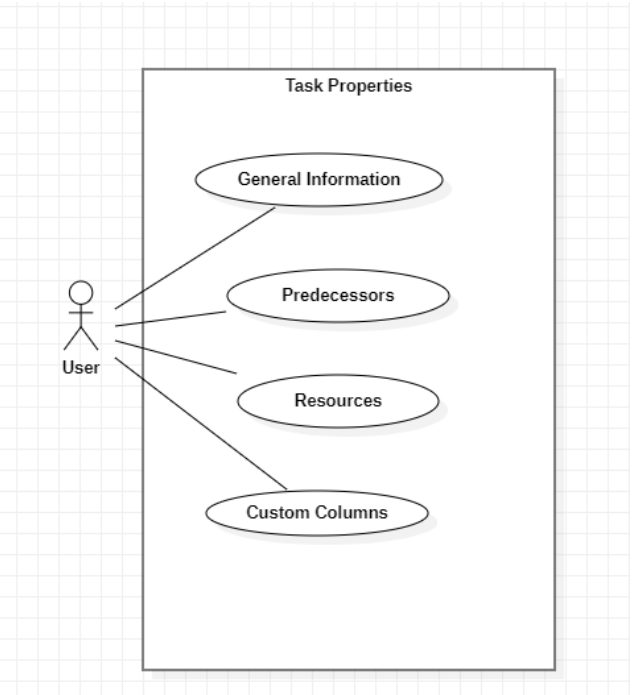
Pedro Arruda 60663

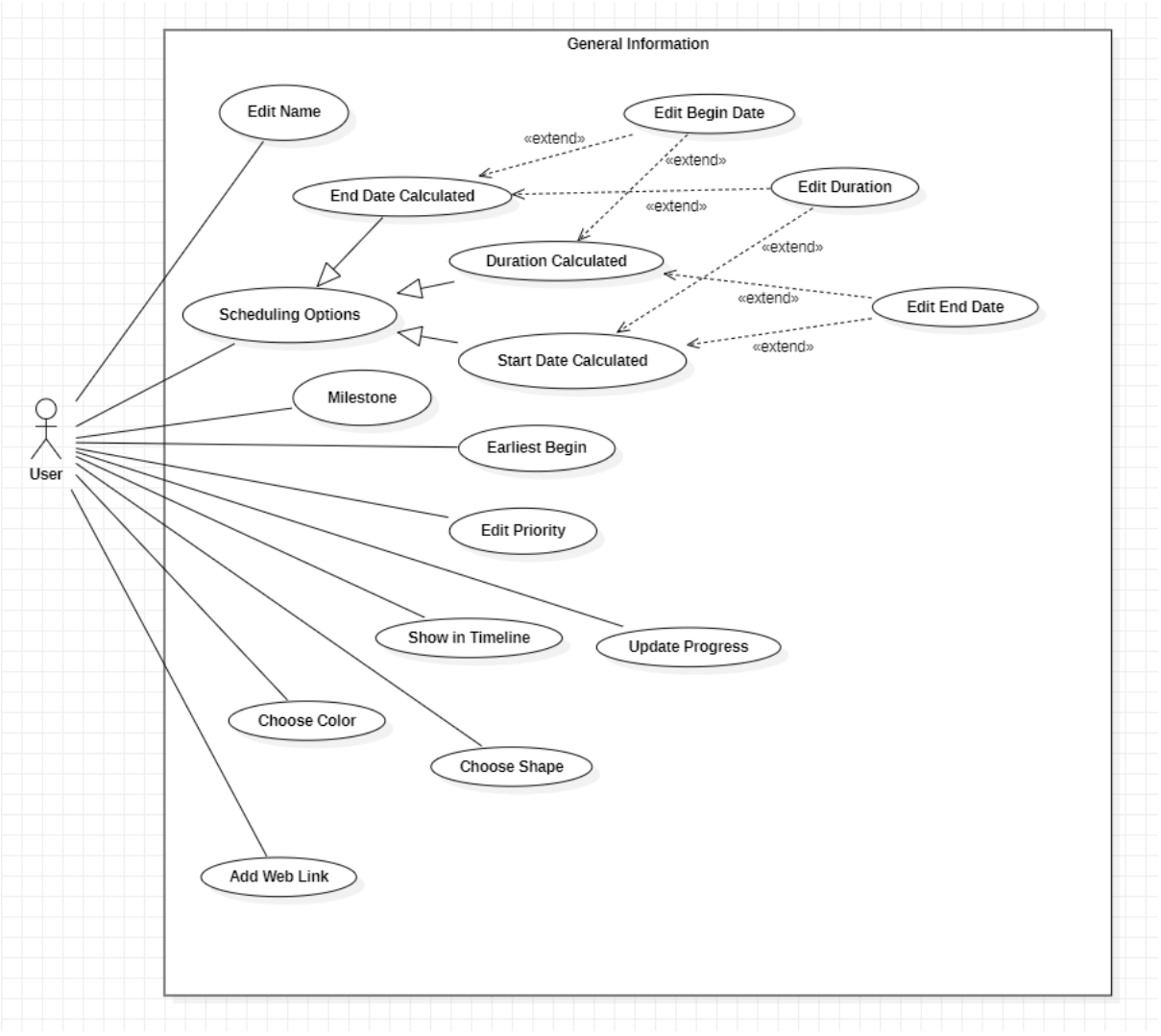


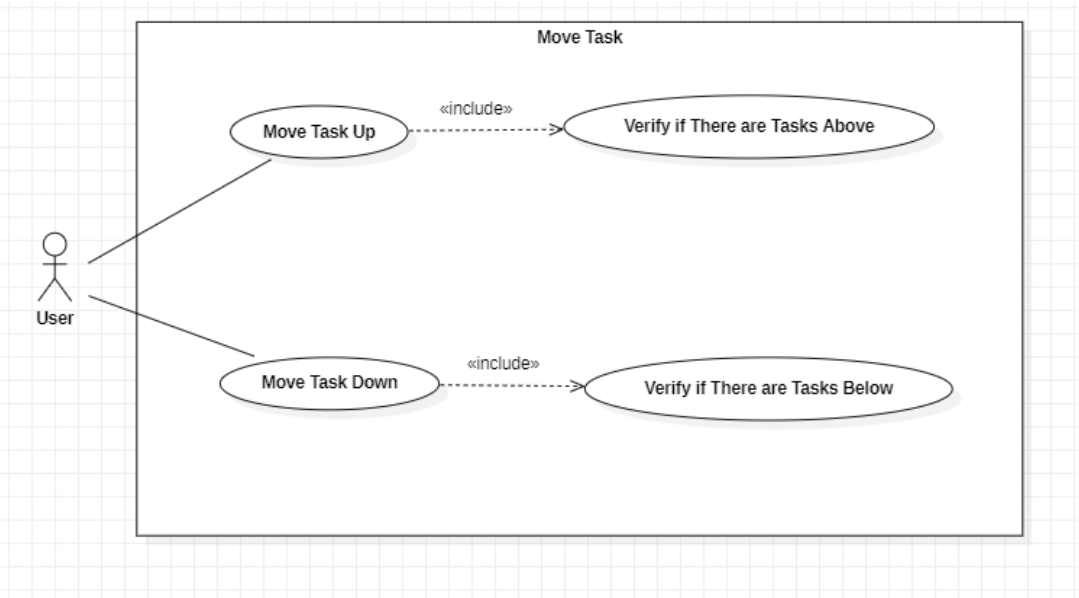


- **Reviewed by Martim Costa 64901**

Ana Gadelha 59943

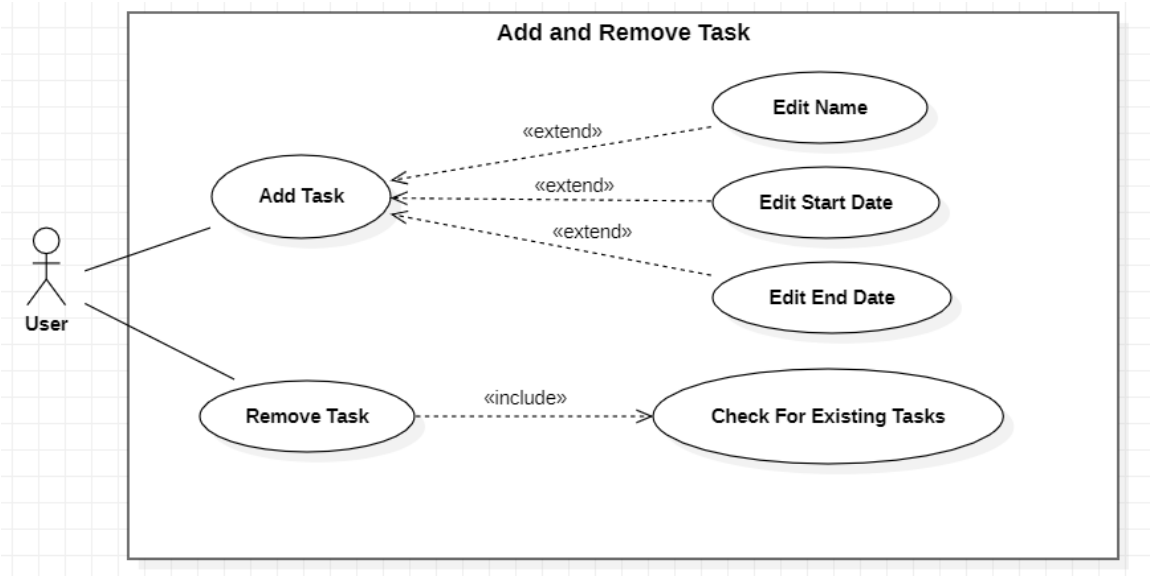


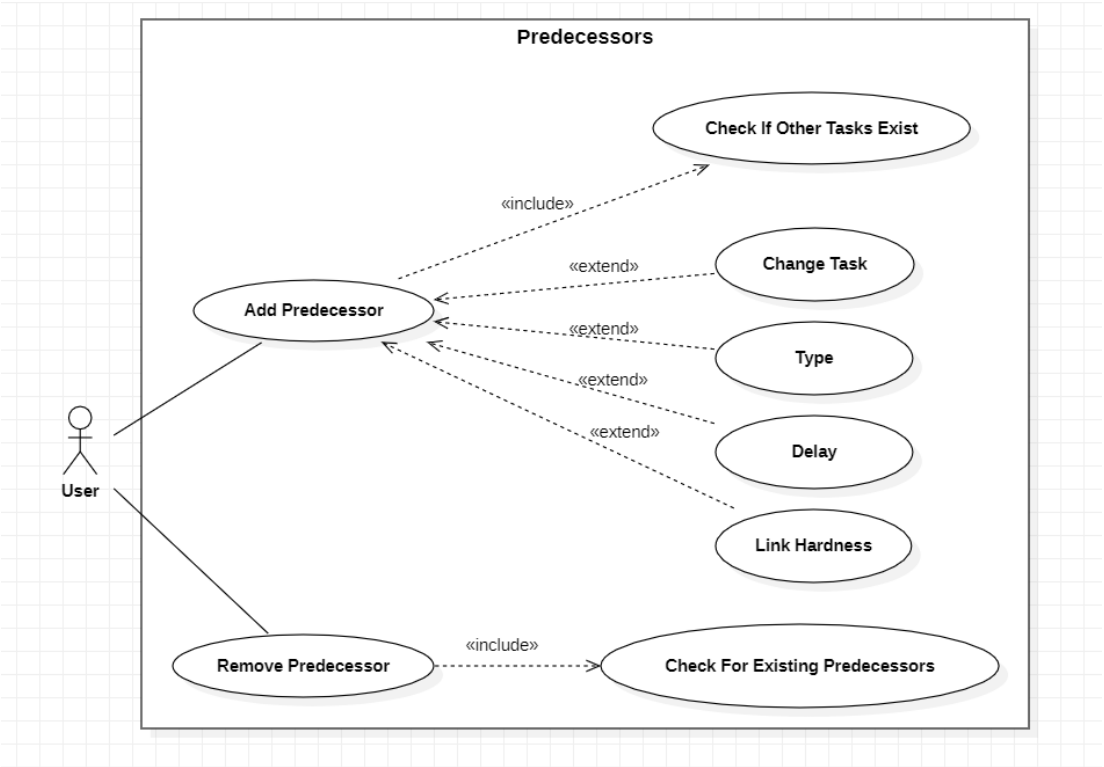


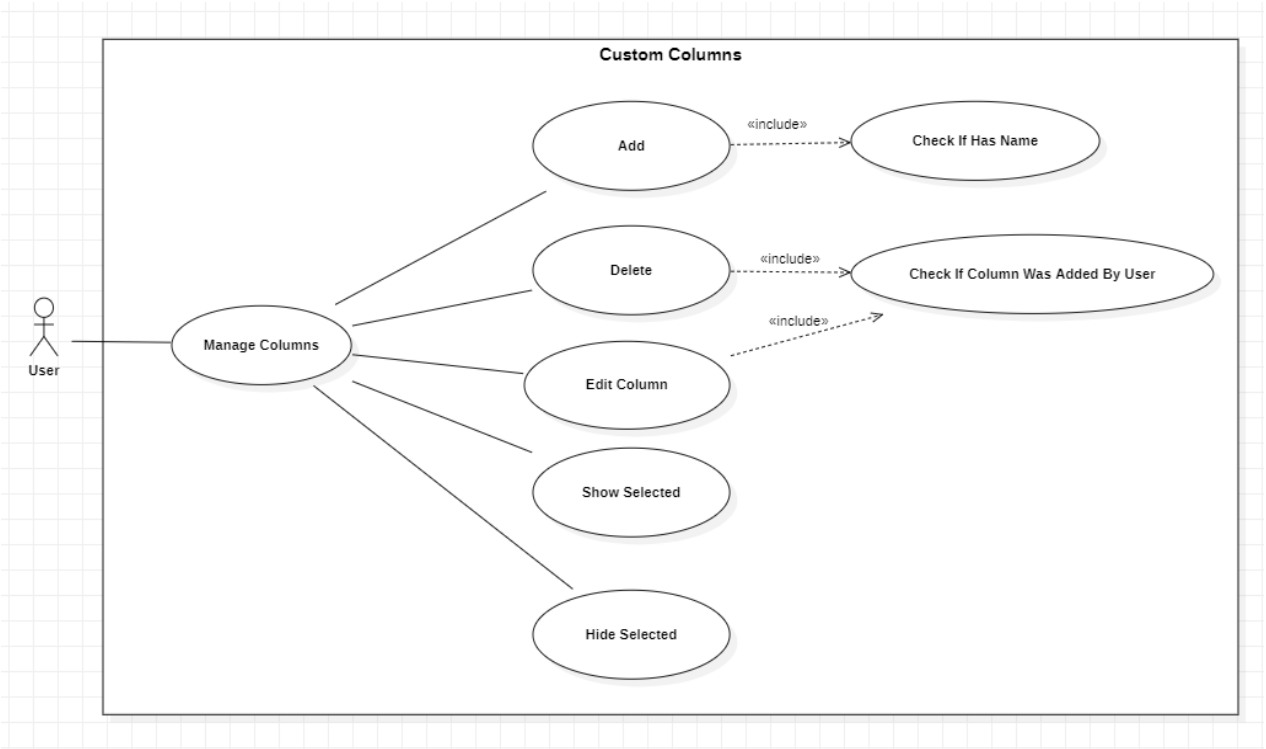


- **Reviewed by Rodrigo Mesquita 59943**

Rodrigo Mesquita 57372

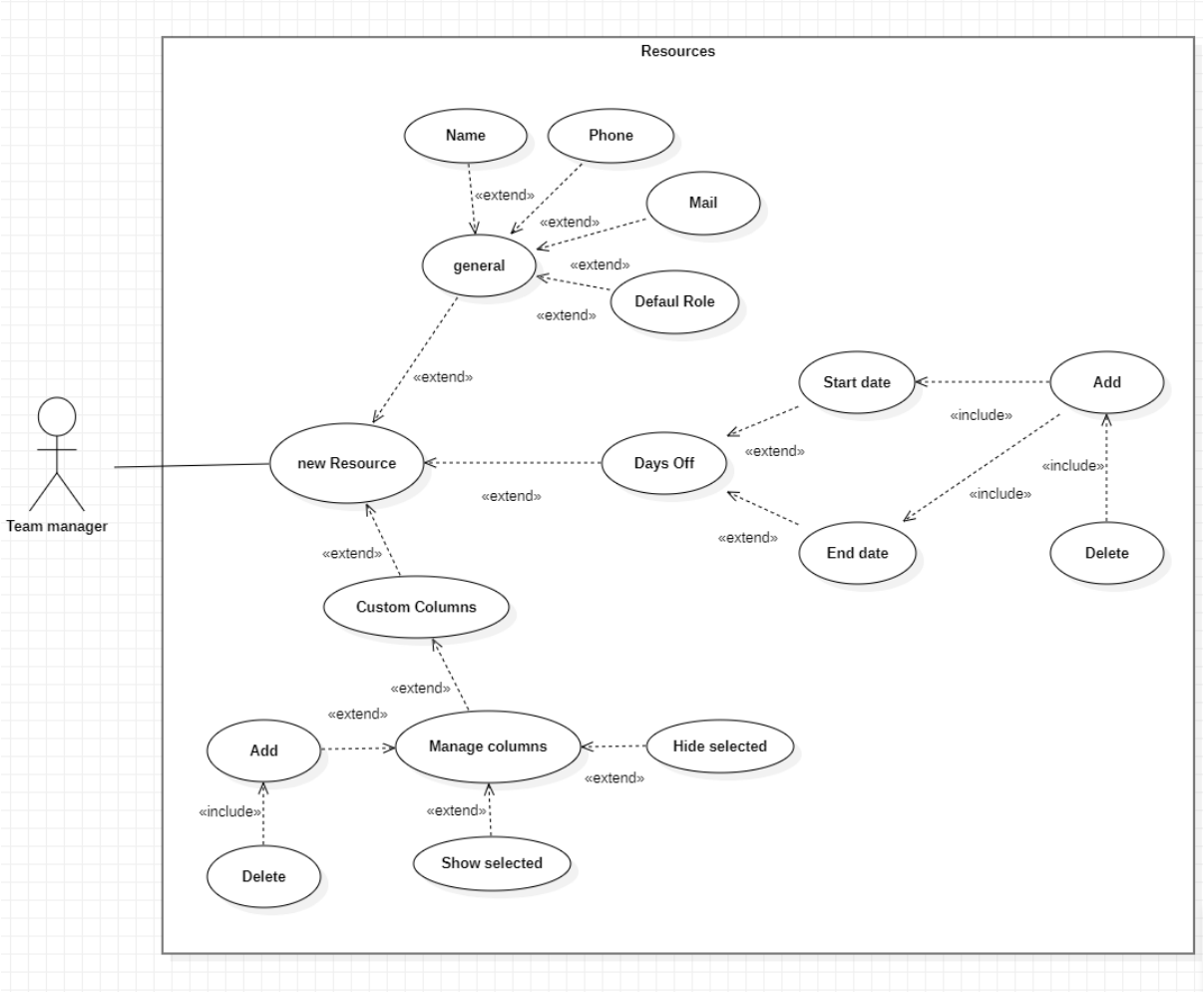






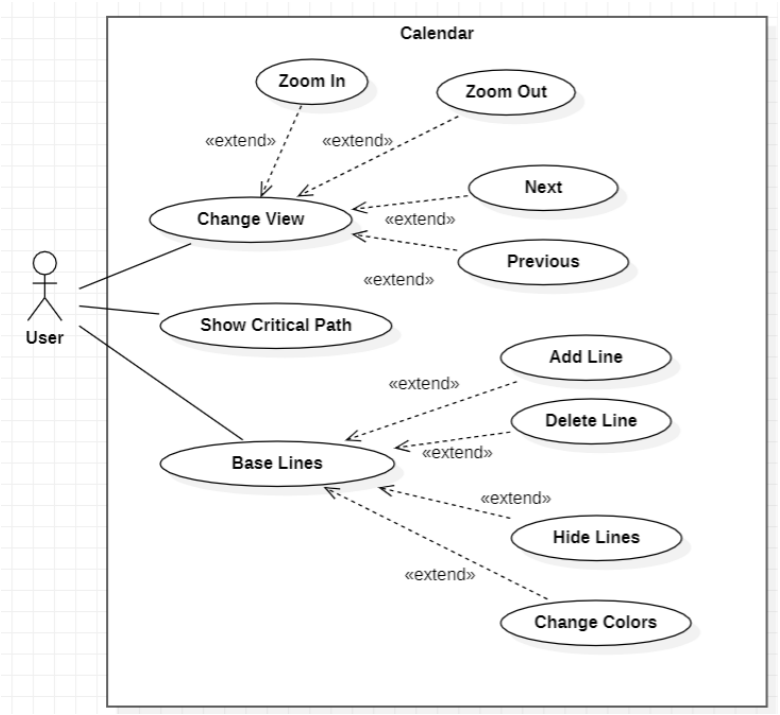
- **Reviewed by: Bernardo Carvalho 60012**

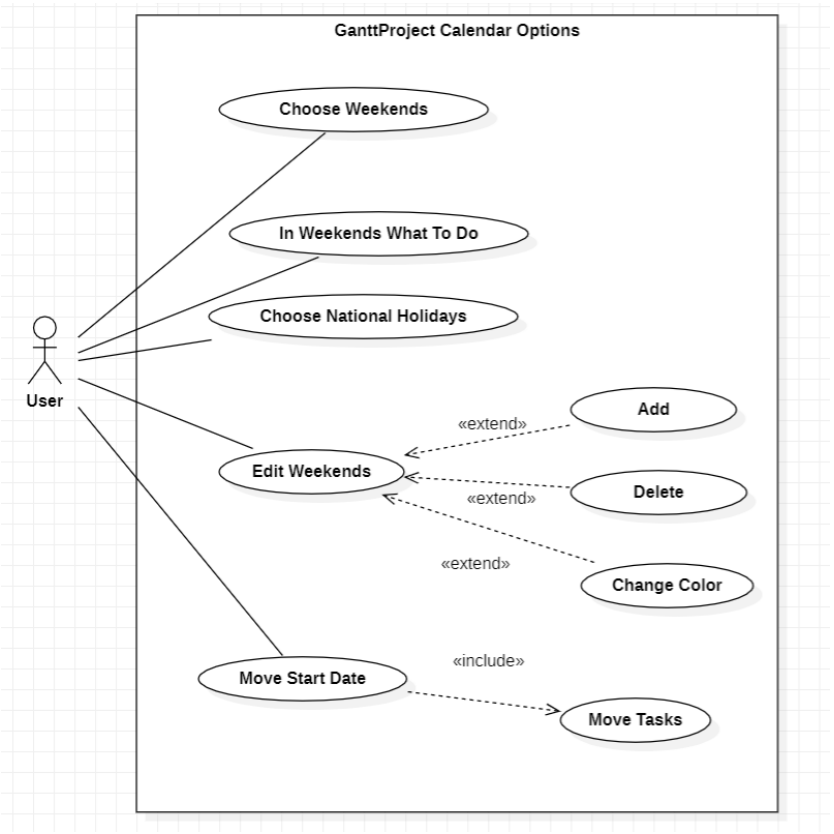
Bernardo Carvalho 60012



- **Reviewed by Pedro Arruda 60663**

Martim Costa 64901





- **Reviewed by Ana Gadelha 59943**

Implementation User Stories

User story 1:

As a user, I want to be able to check important statistics about the progress of the work, so that I can conclude whether the work style should be improved or maintained.

User story 2:

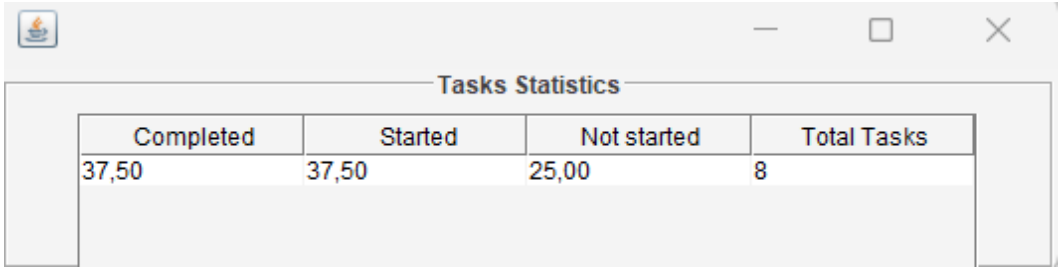
As a team manager, I want to know who has finished their tasks and who hasn't, so that I can give new tasks to those who have none.

Implementations

Implementation 1 - Statistic Data

This new functionality is used to show the user the percentage of tasks that are completed, on doing and the tasks that haven't started yet. This way it is easier for whoever is controlling the work to see if he should maintain the style of work or improve.
 It opens a new window that shows in percentage the type of tasks by the completion progress. It can easily be found in this new menu, called “Statistics” on the top of the app.

An example of data displayed:



Adding the menu

This new menu was created and added in the GanttProject.java

```
342     }
343     mHuman.add(myResourceActions.getResourceSendMailAction());
344     bar.add(mHuman);
345
346   // this will add a "button" to open the stats menu
347     StatsMenu statsMenu =new StatsMenu(getProject());
348     bar.add(statsMenu.createMenu());
349
350     HelpMenu helpMenu = new HelpMenu(getProject(), getUIFacade(), getProjectUIFacade());
351     bar.add(helpMenu.createMenu());
352
```

New class StatsMenu:
Located at **net.sourceforge.ganttproject.action.stats;**

- Imports

```
package net.sourceforge.ganttproject.action.Stats;

import javax.swing.Action;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

import net.sourceforge.ganttproject.IGanttProject;
import net.sourceforge.ganttproject.action.Stats.Stats;
```

- Creation of Variables and Constructor

```
//Menus stats
1 usage  Martim +1 *
public class StatsMenu {
    2 usages
    private Stats stats;

    Martim +1 *
    public StatsMenu(IGanttProject project) {

        stats = new Stats(project);
    }
```

- New method createMenu
This is the method that will create the new menu.

```
// this will create the menu to stats
Martim +1 *
public JMenu createMenu() {
    JMenuBar menu = new JMenuBar();
    JMenu estati = new JMenu("Statistics");
    JMenuItem table = new JMenuItem("Table");
    table.setToolTipText("Statistics Table");
    menu.add(estati);
    estati.add(table);
    Martim *
    table.addActionListener(new ActionListener() {
        Martim
        public void actionPerformed(ActionEvent ev) { stats.makeTable(); }

});
    return estati;}
```

Create the new menu

Create an option that you can select in that menu

Create a TipText, that popup when you leave the mouse above the option "Table"

Add the menu and the option to the menu

ActionListener, when clicked call the method makeTable from Stats

New class Stats
Located at **net.sourceforge.ganttproject.action.stats;**

- Imports

```java
package net.sourceforge.ganttproject.action.stats;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JFrame;
import javax.swing.border.TitledBorder;
import javax.swing.JScrollPane;
import javax.swing.JTable;

import net.sourceforge.ganttproject.task.TaskManager;
import net.sourceforge.ganttproject.IGanttProject;

import java.text.DecimalFormat;
```

- Creation of Variables and Constructor

```java
public class Stats {
// used to round doubles
    3 usages
    private static final DecimalFormat df = new DecimalFormat("0.00");
    4 usages
    private final TaskManager myTaskManager;

    Martim
    public Stats(IGanttProject project) {
        myTaskManager = project.getTaskManager();

    }
```

- New method makeTable

```java
//This method create the table
    Martim +1
    public  void makeTable(){
        JFrame frame = new JFrame();
        JPanel panel = new JPanel();


        panel.setBorder(BorderFactory.createTitledBorder(
                BorderFactory.createEtchedBorder(), "Tasks Statistics", TitledBorder.CENTER, TitledBorder.TOP));
//create table
        String[][] rec = getPercentage();
        String[] header = { "Completed", "Started", "Not started", "Total Tasks" };
        JTable table = new JTable(rec, header);
        panel.add(new JScrollPane(table));
        frame.add(panel);
        frame.setSize(550, 400);
        frame.setVisible(true);
    }
```

Create the table with the title Tasks Statistics, with 4 columns

- New method getPercentage

```java
    private String[][] getPercentage() {

        double completed = 0.0;
        double onDoing = 0.0;
        double notStarted = 0.0;

        String[][] result = {{"0", "0", "0","0"}};

        double calculatedCompleted;
        double calculatedOnDoing;
        double calculatedNotStarted;

        int numberOfTasks = myTaskManager.getTaskCount();
        if (numberOfTasks != 0) {

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

                if (myTaskManager.getTasks()[i].getCompletionPercentage() == 100) {
                    completed++;
                } else if (myTaskManager.getTasks()[i].getCompletionPercentage() > 0) {
                    onDoing++;
                } else {
                    notStarted++;
                }
            }
            calculatedCompleted = (completed / numberOfTasks) * 100;
            calculatedOnDoing = (onDoing / numberOfTasks) * 100;
            calculatedNotStarted = (notStarted / numberOfTasks) * 100;

            String str = df.format(calculatedCompleted) + "";
            String str1 = df.format(calculatedOnDoing) + "";
            String str2 = df.format(calculatedNotStarted) + "";
            String str3 = numberOfTasks + "";

            result[0][0] = str;
            result[0][1] = str1;
            result[0][2] = str2;
            result[0][3] = str3;
        }

        return result;
```

Variables

See if the tasks are completed, on doing or not started and count them

Turns into percentage

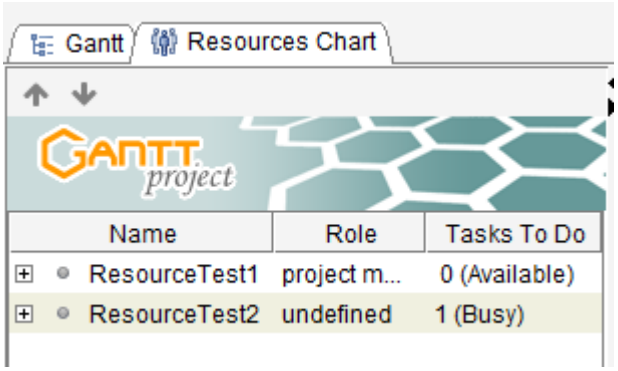
Transform into strings and round to two decimals.
Then put the values in the first row of the matrix

Implementation 2 - Resource Availability

Like the user story for this implementation says, this new functionality is used to easily find out which resources have tasks to do (busy) or are available to receive new tasks.

It calculates the number of assignments that a resource hasn't completed (completion percentage not 100) and updates alone as soon as any percentage is updated.

It can easily be found as a new column in the Resources Chart section, as demonstrated in the image below.



Adding the column

This new column was created in the ResourceDefaultColumn class and set to visible and not editable by default. This visibility can be changed in the app the same way as with any other column.

```
dbarashev +1
public enum ResourceDefaultColumn {
  NAME(new ColumnList.ColumnStub("0", null, true, 0, 150), String.class,  nameKey: "tableColResourceName",  editable: true),
  ROLE(new ColumnList.ColumnStub("1", null, true, 1, 85), String.class,  nameKey: "tableColResourceRole",  editable: true),
  TODO(new ColumnList.ColumnStub("6", null, true, 2, 100), String.class,  nameKey: "todo",  editable: false),
  EMAIL(new ColumnList.ColumnStub("2", null, false, -1, 75), String.class,  nameKey: "tableColResourceEMail",  editable: true),
  PHONE(new ColumnList.ColumnStub("3", null, false, -1, 50), String.class,  nameKey: "tableColResourcePhone",  editable: true),
  ROLE_IN_TASK(new ColumnList.ColumnStub("4", null, false, -1, 75), String.class,  nameKey: "tableColResourceRoleForTask",  editable: true),
  STANDARD_RATE(new ColumnList.ColumnStub("5", null, false, -1, 75), Double.class,  nameKey: "tableColResourceRate",  editable: true);
```

Setting the column value

To do this, new methods and variables were added to the class HumanResource.

```
1 usage  Pedro
public String getNotCompletedTasks(){

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

      if (getAssignments()[i].getTask().getCompletionPercentage() == 100){

        t++;
      }
    }

  int available = (myAssignments.size() - t);

  if (available == 0)
    return " 0 (Available)";
  else
    return String.valueOf(available) + " (Busy)";
}

 Pedro
public void setNotCompletedTasks(){

  notCompletedTasks = getNotCompletedTasks();
}
```

The method getNotCompletedTasks runs a cycle through all the resource's assignments (tasks) and registers the number of tasks that have been completed (completion percentage is equal to 100). After this the number of tasks that haven't been completed is calculated by subtracting this first value from the total number of assignments.

Finally, the built string to be shown is returned.

The method setNotCompletedTasks sets the value of the variable notCompletedTasks.

With this done, we can jump into the ResourceNode class to actually make this value show up in the destined column.

```
1 usage  Pedro
public void setToDo(String number){ resource.setNotCompletedTasks(); }

1 usage  Pedro
public String getToDo(){ return resource.getNotCompletedTasks(); }

dbarashev +1
@Override
public Object getStandardField(ResourceDefaultColumn def) {
  switch (def) {
  case NAME: return getName();
  case ROLE: return getDefaultRole();
  case TODO: return getToDo();
  case EMAIL: return getEMail();
  case PHONE: return getPhone();
  case STANDARD_RATE: return getResource().getStandardPayRate();
  default: return "";
  }
}
```

```
dbarashev +2
@Override
public void setStandardField(ResourceDefaultColumn def, Object value) {
  switch (def) {
  case NAME:
    setName(value.toString());
    return;
  case EMAIL:
    setEMail(value.toString());
    return;
  case PHONE:
    setPhone(value.toString());
    return;
  case ROLE:
    setDefaultRole((Role) value);
    return;
  case TODO:
    setToDo(value.toString());
    return;
  case STANDARD_RATE:
    assert value instanceof Double : "Rate accepts numeric values";
    getResource().setStandardPayRate(BigDecimal.valueOf((Double)value));
  }
}
```

By calling the two newly created methods we can then set the value of the new column accordingly.

Video demonstration

https://youtu.be/z3tKM2WHcVQ