



Engenharia de Software (MiEI) – 2022/2023
3º Ano - 1º Semestre

Relatório Fase 2

Gantt Project

Realizado por:

David Moreira nº 59984

Joana Maroco nº60052

João Lopes nº60055

José Romano nº59241

Sofia Monteiro nº60766

Índice

1	Introdução.....	6
2	Apresentação das Funcionalidades.....	7
2.1	Templates - Abandonada.....	7
2.2	Filtros - Implementada	8
2.3	Alarmes - Implementada.....	8
3	Análise do Código.....	10
4	Code Smells	11
4.1	Comentários	11
4.1.1	Falta de comentários -Autor João Lopes, Review José Romano	11
4.1.2	Código antigo em comentário -Autor Joana Maroco, Review David Moreira	11
4.1.3	Comentário a lembrar um raciocínio – Autor David Moreira, Review João Lopes	12
4.1.4	TODO comentado -Autor José Romano, Review Joana Maroco	12
4.1.5	Excesso de comentários -Autor David Moreira, Review Joana Maroco	13
4.2	Classes.....	14
4.2.1	Classe que devia ser um enumerado -Autor Sofia Monteiro, Review Joana Maroco	14
4.2.2	Classe com múltiplos métodos, definida dentro de uma interface - Autor José Romano, Review David Moreira	14
4.2.3	Classe vazia -Autor Sofia Monteiro, Review José Romano	15
4.2.4	Classe exaustivamente extensa – Autor David Moreira, Review Sofia Monteiro.....	15
4.3	Métodos.....	16
4.3.1	Método demasiado grande e complexo -Autor Joana Maroco, Review João Lopes	16
4.3.2	Utilização de múltiplos if, em vez de um único switch -Autor Sofia Monteiro, Review João Lopes.....	17
4.3.3	Duplicação de código- Autor José Romano, Review Sofia Monteiro	17

4.3.4	Métodos não utilizados (Dead Code) – Autor João Lopes, Review David Moreira	18
4.4	Variáveis/Constantes	19
4.4.1	Valores que deviam ser constantes – Autor João Lopes, Review Sofia Monteiro.....	19
4.4.2	Constantes não utilizadas -Autor Joana Maroco, Review José Romano	19
5	GoF Design Patterns	21
5.1	Memento Pattern-Autor Joana Maroco, Review Sofia Monteiro	21
5.2	The Adapter Pattern -Autor Joana Maroco, Review José Romano	22
5.3	Factory Pattern -Autor Joana Maroco, David Moreira.....	22
5.4	Abstract Factory Pattern -Autor David Moreira, Review João Lopes	23
5.5	Factory Pattern- Autor José Romano, Review Sofia Monteiro.....	24
5.6	Singleton Pattern (Creational Pattern) -Autor Sofia Monteiro, Review João Lopes.....	24
5.7	Observer Pattern(Behavioral Pattern) -Autor Sofia Monteiro, Review Joana Maroco.....	25
5.8	Observer Pattern(Behavioral Pattern) – Autor David Moreira, Review Sofia Monteiro	26
5.9	State Pattern(Behavioural Pattern) -Autor Sofia Monteiro, Review José Romano.....	26
5.10	Command Design Pattern -Autor David Moreira, Review Joana Maroco 27	
5.11	Command Pattern -Autor José Romano, Review David Moreira.....	28
5.12	Command Pattern (Behavioural Pattern) -Autor João Lopes, Review David Moreira	28
5.13	Iterator Pattern – Autor José Romano, Review João Lopes	29
5.14	Facade Pattern (Structural Pattern) – Autor João Lopes, Review José Romano.....	30
5.15	Composite Pattern (Structural Pattern) -Autor João Lopes, Review Joana Maroco.....	30
6	Use Cases	32
6.1	Create Task - João Lopes, Review Sofia Monteiro	32
6.1.1	Descrição	32
6.1.2	Use Case	32
6.2	Delete Resource – Sofia Monteiro, Review José Romano	33

6.2.1	Descrição	33
6.2.2	Use Case	33
6.3	Create Resource – David Moreira, Review Joana Maroco	34
6.3.1	Descrição	34
6.3.2	Use Case	34
6.4	Merge Task - Joana Maroco, Review João Lopes	34
6.4.1	Descrição	34
6.4.2	Use Case	35
6.5	Change Task Ending Date – José Romano, Review David Moreira	35
6.5.1	Descrição	35
6.5.2	Use Case	36
6.6	Alerta	37
	37
6.7	Filtro	38
7	Metrics	39
7.1	Lines of Code – Joana Maroco	39
7.1.1	Explicação das métricas recolhidas	39
7.1.2	Potenciais locais “problemáticos”	41
7.1.3	Relação mantida com os <i>Code Smells</i> identificados.....	41
7.2	Complexity – David Moreira	42
7.2.1	Explicação das métricas recolhidas	42
7.2.2	Potenciais locais “problemáticos”	43
7.2.3	Relação mantida com os <i>Code Smells</i> identificados.....	43
7.3	Chidamber-Kemerer – João Lopes	43
7.3.1	Explicação das métricas recolhidas	43
7.3.2	Potenciais locais “problemáticos”	44
7.3.3	Relação mantida com os <i>Code Smells</i> identificados.....	44
7.4	MOOD – José Romano	45
7.4.1	Explicação das métricas recolhidas	45
7.4.2	Potenciais locais “problemáticos”	45
7.4.3	Relação mantida com os <i>Code Smells</i> identificados.....	46
7.5	Martin Packaging	46

7.5.1	Explicação das métricas recolhidas	46
7.5.2	Potenciais locais “problemáticos”	47
7.5.3	Relação mantida com os Code Smells identificados.....	47
8	Dificuldades	48
9	Conclusão.....	50

1 Introdução

A segunda fase deste projeto envolvia estender o código anteriormente analisado, através da implementação de duas novas funcionalidades. Ambas as funcionalidades foram apresentadas e estudadas por vários membros do grupo, de acordo com a abordagem de cada um ao projeto e à sua organização. Após isto, foram, então, discutidas durante uma das aulas práticas e aprovadas pelo professor responsável pelo turno em questão. Essas duas ideias acabaram por ser a base para a análise do código do GanttProject, tal como para o desenvolvimento de todo este projeto.

Ao longo deste documento, iremos descrever os vários passos que tomamos até chegarmos ao estado final do programa, desde as ideias iniciais, até às muitas dificuldades encontradas pelo caminho.

2 Apresentação das Funcionalidades

2.1 Templates - Abandonada

User Story: “Como Scrum Master eu quero ter a possibilidade de personalizar as informações disponíveis nos recursos, para facilitar a organização do projeto, independentemente do contexto.”

Os Templates ou Atributos, como lhe chamámos, foi a primeira ideia que tivemos para este projeto. Ela surgiu após explorarmos um pouco as funcionalidades disponíveis no GanttProject, especialmente em relação à criação de um HumaResource, ou seja, de um recurso.

O objetivo era poder adicionar informação extra aos recursos, de forma a adaptar-se ao tipo de projeto a ser planeado. Um exemplo que usámos, durante a discussão na aula prática, foi o próprio projeto que estávamos a desenvolver.

Como alunos, cada um de nós tem um número associado (ex: David Moreira nº 59984). Este valor, no nosso caso, seria essencial estar associado a cada um dos membros do grupo. De certa forma, é mais importante que alguns dos valores por definição: email, número de telemóvel, ect...

Chegámos, inclusive, a pensar em múltiplas interfaces gráficas para adicionar e remover Templates, de forma muito semelhante à criação de Recursos ou de Tarefas, como é possível ver nas duas figuras ao lado.

No entanto, após uma análise mais profunda do código, apercebemo-nos que já era possível fazer algo deste género, com as “Colunas Personalizadas”. Pelo que nos vimos obrigados a abandonar esta ideia.

Atributo	Valor
Numero	60766
Universidade	FCT
Curso	MIEI
Nome	Sofia Monteiro
Email	dcl.monteiro...
Telemovel	999999999
Luto	100000.0

■ Fixo - pelo utilizador
■ Alterável
● Exemplo
● Fixo - pelo aplicador

Fig. 1 -Interface 1.

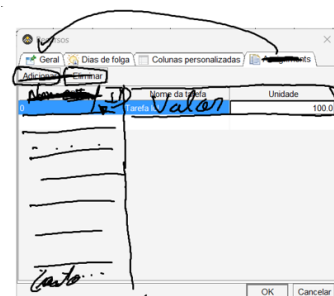


Fig. 2 -Interface 2.

2.2 Filtros - Implementada

User Story: "Como organizador de um grande projeto, quero encontrar e alterar informações sobre os meus recursos mais rapidamente, para que seja mais fácil gerir recursos específicos"

A ideia dos filtros, apesar de ser considerada inicialmente, foi apenas implementada devido ao impedimento na ideia anterior (Templates). Sendo assim, o tempo de planeamento e análise para esta função foi consideravelmente reduzido.

Como o nome indica, os Filtros consiste na filtragem de Tasks ou HumanResources (decidimos optar pela segunda opção) de acordo com uma variável que considerámos importante. Considerámos esta uma ideia essencial, especialmente em projetos de grande escala, e até julgamos um pouco mais simples de implementar, na altura. Isto foi uma das principais razões para termos escolhido os filtros, visto que tínhamos consideravelmente menos tempo à nossa disposição.

Chegámos apenas a desenhar uma simples interface, quando pensámos nesta ideia, que, na altura, estava mais virada para a filtragem de Tasks por intervalos de tempo.

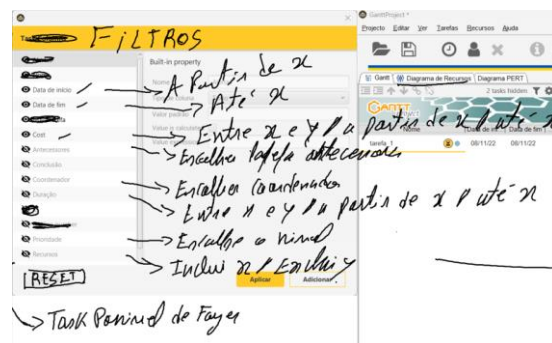


Fig. 3 -Interface.

2.3 Alarmes - Implementada

User Story: "Como utilizador, eu quero receber alertas relativamente à data de conclusão do ganttproject mais próximo, para que seja possível controlar de forma mais eficiente a relação entre as tarefas em falta e o tempo restante."

A ideia dos alarmes, ao contrário das anteriores, surgiu para auxiliar o projeto ao longo do seu desenvolvimento, e não apenas no início (quando se cria o projeto). O seu objetivo é possibilitar a criação e personalização de alarmes em forma de *pop up* ou simplesmente um *icon* na tarefa. É bastante semelhante a uma simples aplicação de eventos, num calendário, que permite que o utilizador seja notificado um certo número de dias (que pode ser alterado por ele mesmo), antes de uma certa data (data do fim da

tarefa).

Decidimos que a configuração por predefinição faria com que o alerta se ativasse no dia final da tarefa, uma vez que, por norma, é o mais útil, não apenas para quem está a fazer a tarefa, mas para quem está a organizá-la.

Inicialmente, tivemos também duas abordagens em relação à implementação desta função. Uma delas implicava tratá-la como os HumanResources e as Tasks (Fig. 4), permitindo, assim, ter múltiplos alarmes para uma mesma Task. Chegamos, inclusive, a ter uma ideia da interface, mas abandonamos rapidamente esta ideia, uma vez que estava muito para além do possível no tempo disponível. A outra forma era simplesmente associar um Alarme a uma Task, tornando este um dos seus “atributos” (Fig. 5).

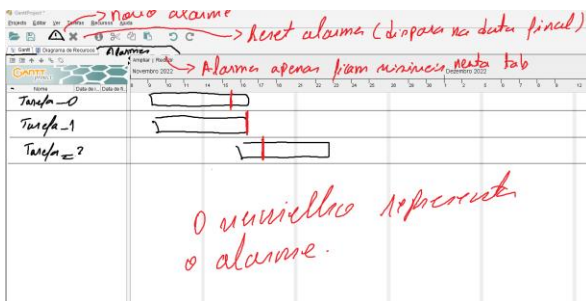


Fig. 4 -Comportamento Igual às Tasks/resources

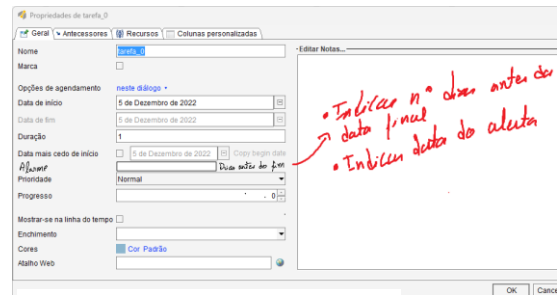


Fig. 5 -Alteração Interface das Tasks

3 Análise do Código

O principal foco do trabalho, nas primeiras semanas, deu-se na análise do código e no planeamento para implementar as funções escolhidas, uma vez que este é a principal componente da própria cadeira.

O objetivo inicial foi encontrar as zonas necessárias a alterar para tornar o código antigo compatível com o que íamos adicionar. Esta tarefa não foi difícil, uma vez que, apesar de ter pouca documentação, as *packages* e classes estão relativamente bem organizadas no sistema de ficheiros. Encontrar as classes que representam HumanResources/Tasks e saber onde adicionar as novas classes a implementar correu sem qualquer dificuldade.

A forma como foram implementadas, por outro lado, causou muitas dúvidas. Com classes dentro de outras classes, classes dentro de interfaces, instâncias de classes abstratas, instâncias a viajar de classe para classe, método para método... Foi perdido muito tempo a tentar fazer sentido do que realmente estava a acontecer.

Após a mudança de *branch* para a “BRANCH_2_8_9”, fomos obrigados a reiniciar a análise, apesar de existirem bastantes parecenças, entre os dois códigos.

Ao longo da análise, fomos trocando notas uns com os outros, quer seja em forma escrita (Fig. 7), por fala, ou desenhos (Fig. 6). Também há que apontar que, grande parte da demora neste processo deveu-se à pesquisa intensiva sobre a classe *java.swing*.

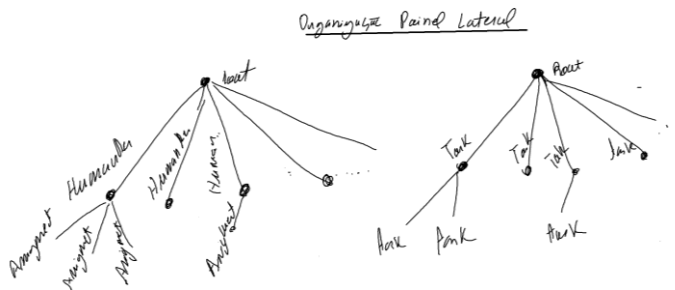


Fig. 6 -Análise do Funcionamento do Painel Lateral

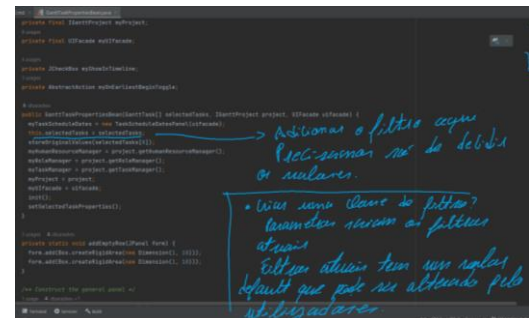


Fig. 7 -Localização de Código a Alterar

4 Code Smells

4.1 Comentários

4.1.1 Falta de comentários -Autor João Lopes, Review José Romano

Ao longo do código, foi notada uma grande falta de comentários, quer seja a descrever uma classe, uma interface ou até mesmo simples métodos. Algumas classes chegam mesmo a ter unicamente o @author como comentário. Qualquer novo desenvolvedor do projeto terá bastante dificuldade em compreender o que um certo pedaço de código faz, ou deve fazer.

Exemplo: Classe abstrata GPCalendarBase

Localização:

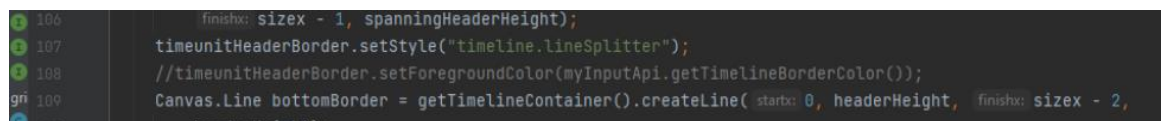
ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.calendar.GP
CalendarBas
e

4.1.2 Código antigo em comentário -Autor Joana Maroco, Review David Moreira

Referenciando, em primeiro lugar, um problema encontrado relativamente aos comentários elaborados no código. Em múltiplas classes, é possível encontrar em comentário vestígios de código antigo (ou possível código novo) por implementar. Por vezes, aparentam ser apenas cópias do código acima.

Pelas linhas 107 e 108 de
ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.chart.scenTi
melineSceneBuilder podemos encontrar, nomeadamente:

Exemplo:



```
106         finish: sizeX - 1, spanningHeaderHeight);  
107         timelineHeaderBorder.setStyle("timeline.lineSplitter");  
108         //timelineHeaderBorder.setForegroundColor(myInputApi.getTimelineBorderColor());  
109         Canvas.Line bottomBorder = getTimelineContainer().createLine( startX: 0, headerHeight, finish: sizeX - 2,
```

Localização:

ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.chart.
scene.TimelineSceneBuilder

Linhas: 107 e 108, respetivamente.

4.1.3 Comentário a relembrar um raciocínio – Autor David Moreira, Review João Lopes

Durante a observação e análise do código do open source, reparei na existência de comentários que servem como lembretes de raciocínio, completamente inconsistente com o resto dos comentários. No caso do exemplo a baixo é possível verificar a extensa explicação do funcionamento do if e das suas consequências, enquanto o resto da classe tem uma grande falta de comentários.

Exemplo:

```
85 if (dependeeVector.getHProjection().reaches(dependantVector.getHProjection().getPoint())) {
86     // when dependee.end <= dependant.start && dependency.type is
87     // any
88     // or dependee.end <= dependant.end && dependency.type==FF
89     // or dependee.start >= dependant.end && dependency.type==SF
90     Point first = new Point(dependeeVector.getPoint().x, dependeeVector.getPoint().y);
91 }
```

Localização:

ganttpproject.biz.ganttpproject.src.main.java.biz.ganttpproject.core.chart.scene.gantt.
DependencySceneBuilder.java

Linha: 85-89

4.1.4 TODO comentado -Autor José Romano, Review Joana Maroco

A utilização de comentários para informar os outros programadores que um método se encontra incompleto ou com problemas é também um ‘Code Smell’. A retirada deste pedaço de código ou o término deste permitirá que os outros programadores olhem para o código de maneira menos confusa, sendo também mais acessível para estes darem o seu contributo.

Exemplo:

```
303 @ dbarashev private static Duration convertLag(TaskDependency dep) {
304     // TODO(dbarashev): Get rid of days
305     return Duration.getInstance(dep.getDifference(), TimeUnit.DAYS);
}
```

Localização:

ganttpproject/biz.ganttpproject.impex.msproject2/src/main/java/biz/ganttpproject/imp
ex /msproject2/ProjectFileExporter.java

Linhas: 303-306

4.1.5 Excesso de comentários -Autor David Moreira, Review Joana Maroco

Apesar de na maior parte das classes não serem quase apresentados comentários, o que por si só é um code smell, existem classes, como a deste exemplo, na qual existem demasiados comentários; o que acaba também por ser um code smell. Demasiados comentários numa pequena porção de código não permitem que um programador consiga entender o código de forma tao eficiente.

Exemplo:

```
35  |  /**
36      * Class used to implement performant, high-quality and intelligent image
37      * scaling and manipulation algorithms in native Java 2D.
38      * <p/>
39      * This class utilizes the Java2D "best practices" for image manipulation,
40      * ensuring that all operations (even most user-provided {@link BufferedImage}
41      * s) are hardware accelerated if provided by the platform and host-VM.
42      * <p/>
43      * <h3>Image Quality</h3>
44      * This class implements a few different methods for scaling an image, providing
45      * either the best-looking result, the fastest result or a balanced result
46      * between the two depending on the scaling hint provided (see {@link Method}).
47      * <p/>
48      * This class also implements an optimized version of the incremental scaling
49      * algorithm presented by Chris Campbell in his <a href="http://today.java
50      * .net/pub/a/today/2007/04/03/perils-of-image-getscaledinstance.html">Perils of
51      * Image.getScaledInstance()</a> article in order to give the best-looking image
52      * resize results (e.g. generating thumbnails that aren't blurry or jagged).
53      * <p>
54      * The results generated by imgscalr using this method, as compared to a single
55      * {@link RenderingHints#VALUE_INTERPOLATION_BICUBIC} scale operation look much
56      * better, especially when using the {@link Method#ULTRA_QUALITY} method.
57      * <p/>
58      * Only when scaling using the {@link Method#AUTOMATIC} method will this class
```

Localização:

ganttproject/ganttproject/src/main/java/org/imgscalr/Scalr.java

Linhas: 35-196

4.2 Classes

4.2.1 Classe que devia ser um enumerado -Autor Sofia Monteiro, Review Joana Maroco

No código, existe pelo menos uma classe que contém apenas valores constantes de um mesmo Objeto. A classe ShapeConstants consiste apenas em 21 constantes de tipo ShapePaint e uma lista com todas estas constantes, sem qualquer variável ou método associados. Este tipo de classe devia, em vez disso, ser considerado um enumerado.

Exemplo:

```
26 public static final ShapePaint TRANSPARENT = new ShapePaint( width: 4, height: 4, new int[] { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 });
27
3 usages
89 public static ShapePaint[] PATTERN_LIST = { TRANSPARENT, DEFAULT, CROSS, VERT, HORZ, GRID, ROUND, NW_TRIANGLE,
90 NE_TRIANGLE, SW_TRIANGLE, SE_TRIANGLE, DIAMOND, DOTS, DOT, SLASH, BACKSLASH, THICK_VERT, THICK_HORZ, THICK_GRID,
91 THICK_SLASH, THICK_BACKSLASH };
92 }
93
```

Localização:

ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.chart.render.
ShapeConstants.java

Linhas: 26-27 e 89-91, respetivamente.

4.2.2 Classe com múltiplos métodos, definida dentro de uma interface -Autor José Romano, Review David Moreira

Na Interface ColumnList podemos encontrar definidas mais uma interface e uma classe com múltiplos métodos. Isto torna o código bastante confuso e difícil de ler. Sendo assim, seria preferível separar esta classe e interface extra.

Exemplo:

```
25 4 implementations dbarashev +2
    public interface ColumnList {
        4 implementations dbarashev

60 dbarashev +2
    class ColumnStub implements ColumnList.Column {
        7 usages
```

Localização:

ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.table.ColumnList.java

Linhas: 25 e 60, respetivamente.

4.2.3 Classe vazia -Autor Sofia Monteiro, Review José Romano

Foi criada, também, uma classe totalmente vazia, que não chega a ser utilizada em lado nenhum. Esta classe pode ser considerada 'dead code', caso simplesmente não seja utilizada, ou então 'speculative generality' caso o seu autor tenha intenção de a usar, no início, mas acabou por não ser necessária.

Exemplo:

```
1 package biz.ganttproject.impex.msproject2;
2
3 public class WebStartIDClass {
4
5 }
6
```

Localização:

ganttproject.biz.ganttproject.impex.msproject2.src.main.java.biz.ganttproject.impex.msproject2.WebStartIDClass.java

Linhas: 1-5

4.2.4 Classe exaustivamente extensa – Autor David Moreira, Review Sofia Monteiro

Ao longo da análise do código, também encontrei várias classes anormalmente extensas. Um exemplo destas é a classe Canvas, com mais de 708 linhas de código. A maior parte destas é devido ao facto de estar a declarar várias classes, como a Rhombus, Arrow, e a Line. Classes estas que podiam ser declaradas individualmente, diminuindo, assim, o peso sobre a classe Canvas.

Exemplo:

```
44 usages 2 inheritors dbarashev
148 public static class Polygon extends Shape {
5 usages
```



```

230 public static class Rectangle extends Polygon {
246 public static class Rhombus extends Polygon {
257 public static class Arrow extends Shape {

```

Localização:

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

Linhas: 148, 230, 246, 257, respetivamente.

4.3 Métodos

4.3.1 Método demasiado grande e complexo -Autor Joana Maroco, Review João Lopes

Na classe DateParser existe um método com um total de 122 linhas, o que o torna confuso e difícil de ler. Este método pode ser facilmente decomposto em múltiplos métodos auxiliares de nomes fáceis de ler.

Exemplo:

```

35 private static Calendar getCalendar(String isodate)
36     throws InvalidDateException {
37     // YYYY-MM-DDThh:mm:ss.sTZD
38     StringTokenizer st = new StringTokenizer(isodate, delim: "-T:..+Z", returnDelims: true);
39

```

Localização:

ganttproject.biz.ganttproject.core.src.main.java.org.w3c.util.DateParser.java

Linhas: 35-157

4.3.2 Utilização de múltiplos if, em vez de um único switch -Autor Sofia Monteiro, Review João Lopes

Na classe GPTimeUnitStack, o método encode(TimeUnit timeUnit) utiliza vários if, uns atrás dos outros. Estes if podiam ser facilmente substituídos por um único switch, cujo default envia uma exceção.

Exemplo:

```
124 public String encode(TimeUnit timeUnit) {  
125     if (timeUnit == HOUR) {  
126         return "h";  
127     }  
128     if (timeUnit == DAY) {  
129         return "d";  
130     }  
131     if (timeUnit == WEEK) {  
132         return "w";  
133     }  
134     throw new IllegalArgumentException();  
135 }
```

Localização:

ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.time.impl.GP
TimeUnitStack.java

Linhas: 124-135

4.3.3 Duplicação de código- Autor José Romano, Review Sofia Monteiro

A duplicação de código de maneira a preencher todos os dias de um calendário retrata uma má prática de código (code smell) muito comum, a repetição de código. Com o objetivo de eliminar a repetição de código, é possível recorrer a um ciclo {for each} de maneira a percorrer todos os valores pertencentes ao Enum {Day}, chamando desta forma a função {calendar.setWorkingDay(...)} uma vez apenas.

Exemplo:

```

2 usages  dbarashev
110 @ private void exportWeekends(ProjectCalendar calendar) {
111     ProjectCalendarHours workingDayHours = calendar.getCalendarHours(Day.MONDAY);
112     calendar.setWorkingDay(Day.MONDAY, isWorkingDay(Calendar.MONDAY));
113     calendar.setWorkingDay(Day.TUESDAY, isWorkingDay(Calendar.TUESDAY));
114     calendar.setWorkingDay(Day.WEDNESDAY, isWorkingDay(Calendar.WEDNESDAY));
115     calendar.setWorkingDay(Day.THURSDAY, isWorkingDay(Calendar.THURSDAY));
116     calendar.setWorkingDay(Day.FRIDAY, isWorkingDay(Calendar.FRIDAY));
117     calendar.setWorkingDay(Day.SATURDAY, isWorkingDay(Calendar.SATURDAY));
118     if (calendar.isWorkingDay(Day.SATURDAY)) {
119         copyHours(workingDayHours, calendar.addCalendarHours(Day.SATURDAY));
120     }
121     calendar.setWorkingDay(Day.SUNDAY, isWorkingDay(Calendar.SUNDAY));
122     if (calendar.isWorkingDay(Day.SUNDAY)) {
123         copyHours(workingDayHours, calendar.addCalendarHours(Day.SUNDAY));
124     }
125 }

```

Localização:

ganttproject/biz.ganttproject.impex.msproject2/src/main/java/biz/ganttproject/i
mpex/msproject2/ProjectFileExporter.java

Linhas: 110-124

4.3.4 Métodos não utilizados (Dead Code) – Autor João Lopes, Review David Moreira

Um outro Code Smell que detetamos foi o facto de existirem métodos não utilizados, ou seja, código morto. Estes podem confundir o programador e estendem o código desnecessariamente. Uma solução possível é retirá-los e apenas adicioná-los quando forem realmente necessários.

Exemplo:

```

594 public void setMaxLength(int maxLength) {
595     myMaxLength = maxLength;
596 }
597
598 public int getMaxLength() {
599     return myMaxLength;
600 }

```

Localização:

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

Linhas: 394-400

4.4 Variáveis/Constantes

4.4.1 Valores que deviam ser constantes – Autor João Lopes, Review Sofia Monteiro

Especialmente na classe WeekendCalendarImpl, foi notado o uso repetitivo de certos valores, cujo significado é igual. Neste caso, o 7, que representa o número de dias da semana. Em vez de criar uma constante, apenas é repetido o número ao longo de toda a classe, cada vez que é necessário. Isto dificulta muitas vezes a compreensão do código e obrigada o desenvolvedor a encontrar todas as vezes que o número foi mencionado, caso tenha intenção de o alterar.

Exemplo:

```
64         private final DayType[] myTypes = new DayType[7];
65
363         setPublicHolidays(calendar.getPublicHolidays());
364         for (int i = 1; i <= 7; i++) {
365             setWeekDayType(i, calendar.getWeekDayType(i));
366         }
```

Localização:

ganttproject.biz.ganttproject.core.src.main.java.biz.ganttproject.core.calendar.WeekendCalendarImpl

Linha: 64 e 364, respetivamente.

4.4.2 Constantes não utilizadas -Autor Joana Maroco, Review José Romano

Foram também encontradas constantes que nunca são utilizadas no código. Estas, naturalmente, também representam um Code Smell, uma vez que podem confundir um novo desenvolvedor. Sendo assim, a melhor solução é retirá-las, até serem realmente necessárias.

Exemplo:

```
28 public class CustomColumnsException extends Exception {  
29     public static final int ALREADY_EXIST = 0;  
30  
31     public static final int DO_NOT_EXIST = 1;  
32 }
```

Localização:

ganttproject.ganttproject.src.main.java.biz.ganttproject.customproperty.CustomColumnsException.java

Linhas: 29-31

5 GoF Design Patterns

5.1 Memento Pattern-Autor Joana Maroco, Review Sofia Monteiro

Esta interface e as respetivas implementações preservam o estado antigo, corrente e futuro do documento a editar. Isto é o foco do Memento Pattern, uma vez que ele permite que em qualquer momento seja possível ao utilizador (des)faça quaisquer operações realizadas no texto/documento a tratar.

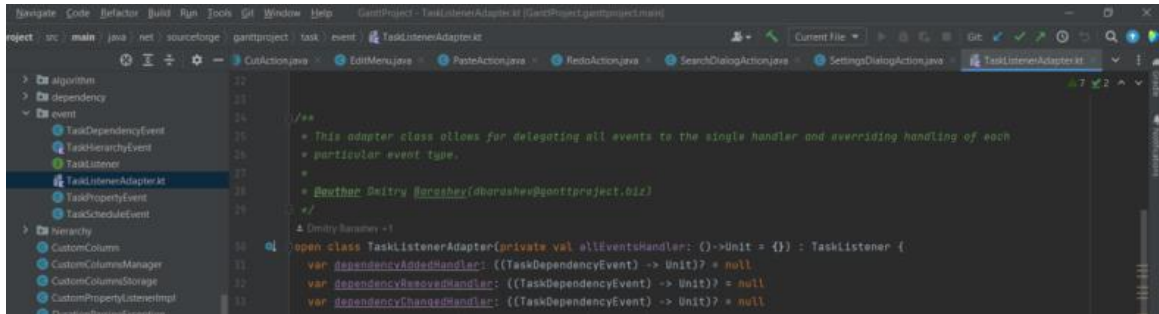
Pelas linhas 33-49 de `ganttproject/src/main/java/net/sourceforge/ganttproject/document/DocumentManager.java` temos, por exemplo:

```
public interface DocumentManager {  
    Document newUntitledDocument() throws IOException;  
    Document newDocument(String path) throws IOException;  
    Document newAutosaveDocument() throws IOException;  
    Document getLastAutosaveDocument(Document priorTo) throws IOException;  
    Document getDocument(String path);  
    ProxyDocument getProxyDocument(Document physicalDocument);  
    void changeWorkingDirectory(File parentFile);  
    String getWorkingDirectory();  
    GPOptionGroup getOptionGroup();  
}
```

5.2 The Adapter Pattern -Autor Joana Maroco, Review José Romano

Este Design Pattern foca-se em facilitar a comunicação entre dois sistemas/objetos, através de uma interface compatível para ambos. E é isto que observamos neste exemplo. A TaskListenerAdapter é uma interface de apoio ao manipulador de eventos consoante cada tipo de evento.

Pelas linhas 24 a 33 de ganttproject/src/main/java/net/sourceforge/ganttproject/task/event/TaskListenerAdapter.kt, temos, por exemplo:



5.3 Factory Pattern -Autor Joana Maroco, David Moreira

Como se pode ver neste exemplo, é definida a interface ActionStateChangedListener para criar todos os produtos distintos deste tipo, mas deixa a criação real do produto para a classe concreta do Produto.

Apresento as linhas 80-97 de ganttproject/src/main/java/net/sourceforge/ganttproject/action/ArtefactAction.java (do produto) sendo a localização da interface: ganttproject/src/main/java/net/sourceforge/ganttproject/action/ActionStateChangedListener.java

```
ganttproject > action > ArtefactAction > actionStateChanged
README x ArtefactAction.java x ActionStateChangedListener.java x
71 protected String getLocalizedDescription() {
72     if (myProvider == null) {
73         return super.getLocalizedDescription();
74     }
75     GPAction activeAction = (GPAction) myProvider.getActiveAction();
76     return activeAction.getLocalizedDescription();
77 };
78
79 7 usages dbarashev +1
80 @Override
81 public void actionStateChanged() {
82     // State of a delegate action has been changed, so update out state as well
83     GPAction activeAction = (GPAction) myProvider.getActiveAction();
84     if (activeAction == null) {
85         setEnabled(false);
86     } else {
87         setEnabled(activeAction.isEnabled());
88     }
89 }
```

5.4 Abstract Factory Pattern -Autor David Moreira, Review João Lopes

No exemplo apresentado, é definida a interface `CalendarFactory` que controla todos os tipos de unidade que aparecem no calendário relativamente ao tempo e duração. No entanto, para cada um dos Produtos que o implementam, ainda são implementadas classes específicas, isto é, conseguimos observar uma "fábrica de fábricas", como descrito nas teóricas.

```
86 public abstract class CalendarFactory {
87     7 implementations dbarashev
88     public static interface LocaleApi {
89         7 implementations dbarashev
90         Locale getLocale();
91         7 implementations dbarashev
92         DateFormat getShortDateFormat();
93     }
94
95     5 usages
96     private static LocaleApi ourLocaleApi;
97
98     dbarashev
99     public static Calendar newCalendar() { return (Calendar) Calendar.getInstance(ourLocaleApi.getLocale()).clone(); }
100
101     dbarashev
102     protected static void setLocaleApi(LocaleApi localeApi) { ourLocaleApi = localeApi; }
103
104     dbarashev
105     public static GanttCalendar createGanttCalendar(Date date) { return new GanttCalendar(date, ourLocaleApi); }
106
107     dbarashev
108     public static GanttCalendar createGanttCalendar(int year, int month, int date) {
109         return new GanttCalendar(year, month, date, ourLocaleApi);
110     }
111
112     dbarashev
113     public static GanttCalendar createGanttCalendar() { return new GanttCalendar(ourLocaleApi); }
114 }
```

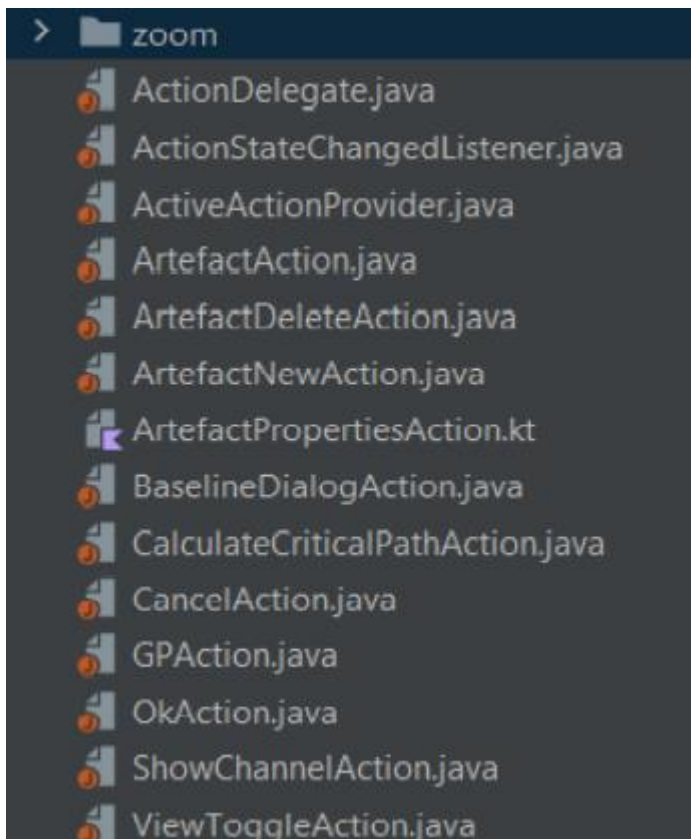

Localização:

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

5.5 Factory Pattern- Autor José Romano, Review Sofia Monteiro

O Factory Pattern define-se como a criação de uma interface para a criação de objetos numa superclasse para assim permitir às subclasses alterarem o tipo de objetos que vão criar.

Neste caso, temos a interface ActionListener em que depois esta é implementada por vários objetos de tipos diferentes.



Localização: ganttproject/src/main/java/net/sourceforge/ganttproject/action

5.6 Singleton Pattern (Creational Pattern) -Autor Sofia Monteiro, Review João Lopes

Um padrão do estilo singleton tem como função principal central de forma mais restrita as variáveis globais: garante que existe apenas uma instância de

classe, ou seja, existe apenas a classe singleton (classe GanttLookAndFeels). Tem também como característica ter um objeto compartilhado por diferentes partes do programa.

```
package net.sourceforge.ganttproject.gui;

import ...

/**
 * @author Michael Haeusler (michael at akatose.de) This singleton class stores
 * info about the installed LookAndFeels.
 */
public class GanttLookAndFeels {

    5 usages
    protected Map<String, GanttLookAndFeelInfo> infoByClass;

    3 usages
    protected Map<String, GanttLookAndFeelInfo> infoByName;
```

Localização:

ganttproject/src/main/java/net/sourceforge/ganttproject/gui/GanttLookAndFeels.java

5.7 Observer Pattern(Behavioral Pattern) -Autor Sofia Monteiro, Review Joana Maroco

Este padrão serve essencialmente para em vez de fazermos a chamada de um método diversas vezes para várias classes, por exemplo update(), fazemos apenas uma vez no observer/listener e este comunica com os restantes caso seja necessário.

Neste caso, temos a classe GPCalendarListener.java que é exatamente um listener de algum update que tenha de haver no calendário e, se houver, envia isso para as classes correspondentes.

```

package biz.ganttproject.core.calendar;

/**
 * Calendar listeners are notified when calendar is changed, namely,
 * when weekends days change or holidays list change.
 *
 * @author dbarashev (Dmitry Barashev)
 */
public interface GPCalendarListener {
    void onCalendarChange();
}

```

Localização:

ganttproject/biz.ganttproject.core/src/main/java/biz/ganttproject/core/calendar/GPCalendarListener.java

5.8 Observer Pattern(Behavioral Pattern) – Autor David Moreira, Review Sofia Monteiro

Usamos este tipo de padrão quando se pretende atualizar algum parâmetro que irá alterar por consequência muitos outros parâmetros. Para tal usamos Interfaces Listeners que automaticamente atualizam todos os outros parâmetros a atualizar quando recebe o update do parâmetro original. Neste exemplo podemos observar o ScrollingListener que recebe o update quando se dá scroll a algum número de dias.

5.9 State Pattern(Behavioural Pattern) -Autor Sofia Monteiro, Review José Romano

O padrão State tem como objetivo alterar o comportamento de um objeto quando o seu estado se altera. É usado de maneira a simplificar o código, limpando o número de excessivo de condições. No método parseDuration, pertencente à classe GPTimeUnitStack, verifica-se a obediência a este padrão, dado que a condição Character.isDigit(nextChar) muda de comportamento, dependendo do estado (state) em que se encontra.

```

@Override
public TimeDuration parseDuration(String lengthAsString) throws ParseException {
    int state = 0;
    StringBuffer valueBuffer = new StringBuffer();
    Integer currentValue = null;
    TimeDuration currentLength = null;
    lengthAsString += " ";
    for (int i = 0; i < lengthAsString.length(); i++) {
        char nextChar = lengthAsString.charAt(i);
        if (Character.isDigit(nextChar)) {
            switch (state) {

```

Localização:

ganttproject/biz.ganttproject.core/src/main/java/biz/ganttproject/core/time/impl/GP
TimeUnitStack.java

Linhas: 161-254

5.10 Command Design Pattern -Autor David Moreira, Review Joana Maroco

Como se pode observar neste exemplo, existem classes que representam ações específicas (no caso, "Copy", "Cut", "Edit", "Paste", "Redo", "RefreshView", "Search Dialog", "SettingsDialog" e "Undo"). Todas estas classes guardam linhas de código para serem executadas mais tarde as vezes que forem necessárias e vão ser enviadas pelo Invoker para o Receiver para este corre o comando ou a ação.

```

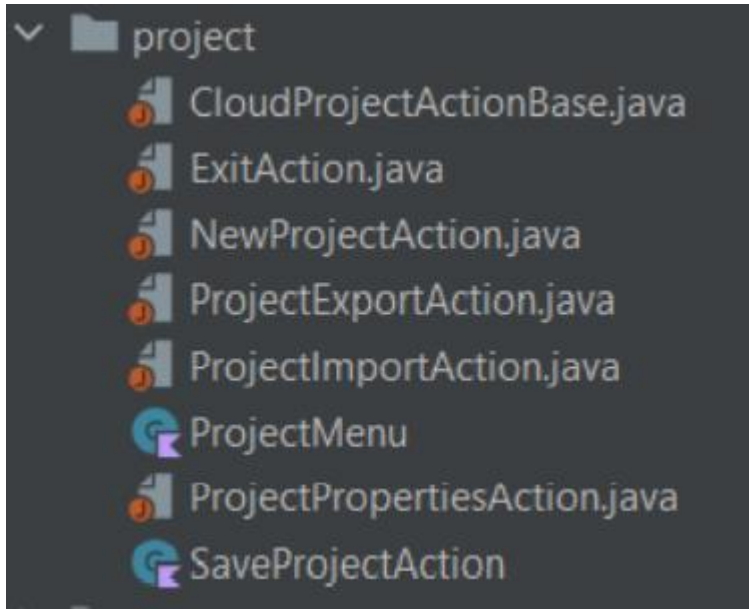
30 public class CopyAction extends GPAction {
    3 usages
31 private final GPViewManager myViewmanager;
32
    2 usages dbarashev
33 public CopyAction(GPViewManager viewManager) {
34     super( name: "copy");
35     myViewmanager = viewManager;
36 }
37
    dbarashev +1
38 @Override
39 public void actionPerformed(ActionEvent e) {
40     if (calledFromAppleScreenMenu(e)) {
41         return;
42     }
43     myViewmanager.getSelectedArtefacts().startCopyClipboardTransaction();
44 }

```

Localização: ganttproject/src/main/java/net/sourceforge/ganttproject/action/edit

5.11 Command Pattern -Autor José Romano, Review David Moreira

Observando o exemplo, podemos verificar que há classes que expressam ações concretas, tais como "ExitAction", "NewProjectAction", "ProjectExportAction", entre outras.



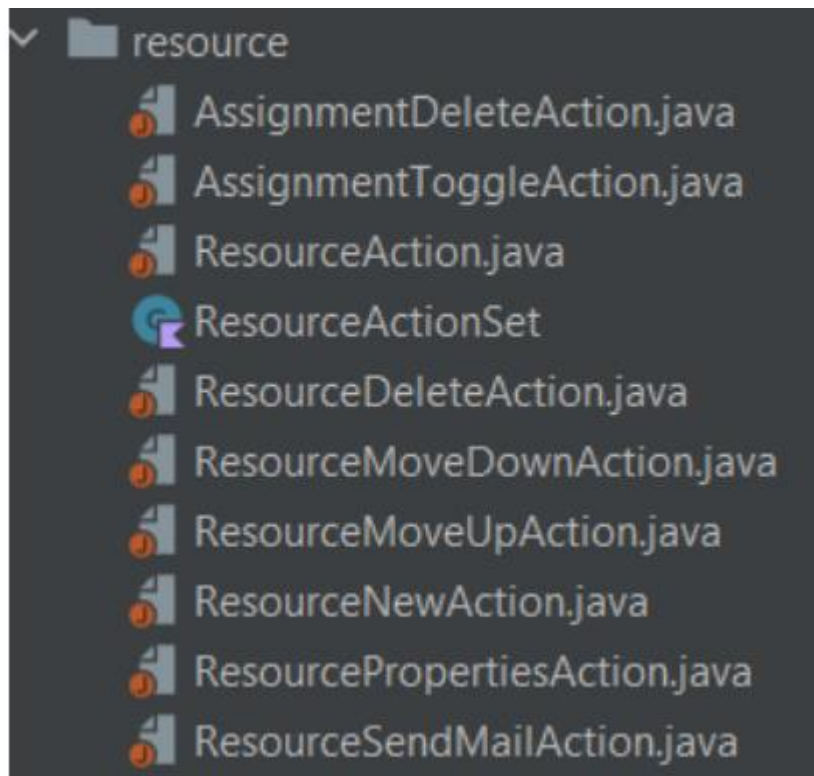
Localização:

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

5.12 Command Pattern (Behavioural Pattern) -Autor João Lopes, Review David Moreira

O command Pattern é essencialmente existirem objetos que se caracterizem por comandos, mas que são definidos como classes. Neste caso temos, por exemplo, "AssignmentDeleteAction", "AssignmentToggleAction", "ResourceAction", "ResourceDeleteAction" entre outras classes.

Exemplos:



Localização:

ganttproject/src/main/java/net/sourceforge/ganttproject/action/resource

5.13 Iterator Pattern – Autor José Romano, Review João Lopes

É usado um iterador para correr todos os documentos existentes na nossa aplicação.

```
/** @return an Iterator over the entries of the list of Documents MRU */  
@dbarashev  
public Iterator<String> iterator() { return documents.iterator(); }
```

Localização:

ganttproject/src/main/java/net/sourceforge/ganttproject/document/DocumentsMRU.java

Linhas: 95-97

5.14 Facade Pattern (Structural Pattern) – Autor João Lopes, Review José Romano

Este padrão simplifica a interação com um *framework* complexo de diversas Tasks/Tarefas. No caso, é uma interface para aceder a todas elas de forma hierárquica.

Exemplo:

```
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);  
}
```

Localização:

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

Linhas apresentadas: 32-44

5.15 Composite Pattern (Structural Pattern) -Autor João Lopes, Review Joana Maroco

Usado para implementação de estruturas do tipo tree (classe TreeTableViewSkin). Tem como características a partilha de uma interface (TableViewSkinBase), algo que permite a criação de folhas (leaf) mais simples e retira a preocupação do cliente necessitar de ter em conta as classes dos objetos com os quais trabalha.

Exemplo:

```
* @see TableView
* @see TreeTableView
* TableViewSkin
* @see TreeTableViewSkin
```

```
/**
 * Keeps track of how many leaf columns are currently visible in this table.
 */
2 usages  ⤴ Dmitry Barashev
private void updateVisibleColumnCount() {
    visibleColCount = getVisibleLeafColumns().size();

    updatePlaceholderRegionVisibility();
    requestRebuildCells();
}
```

Localização:

ganttproject/ganttproject/src/main/java/biz/ganttproject/lib/fx/treetable/TableViewSkinBase.java

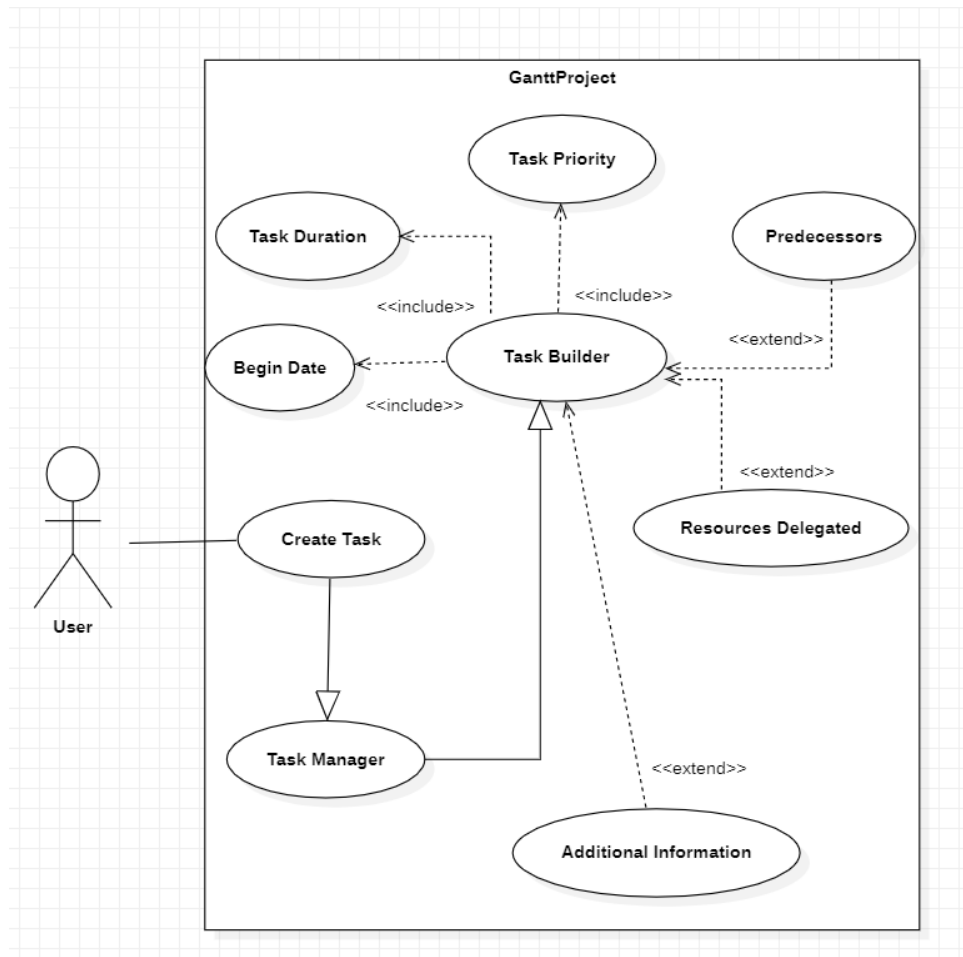
6 Use Cases

6.1 Create Task - João Lopes, Review Sofia Monteiro

6.1.1 Descrição

O Use Case abaixo representa o processo para a criação de uma Task/Tarefa, incluindo os atributos necessários para o fazer, que o próprio programa obtém.

6.1.2 Use Case

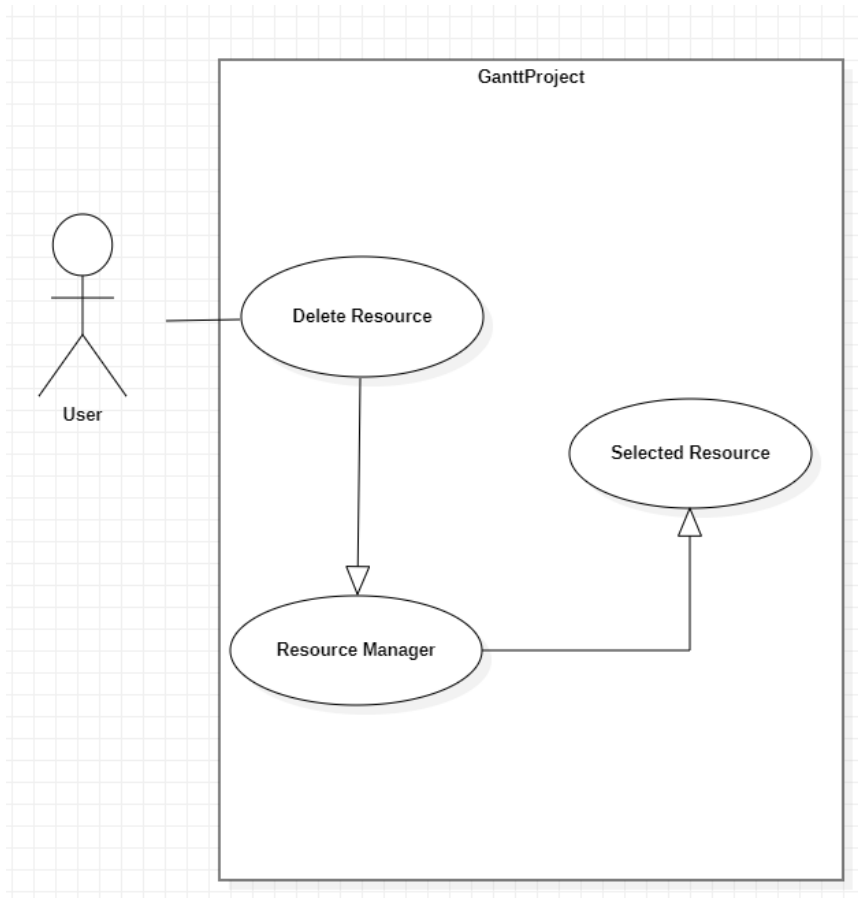


6.2 Delete Resource – Sofia Monteiro, Review José Romano

6.2.1 Descrição

O Use Case em causa mostra como são eliminadas Tasks do programa, em que o Ator Principal (único ator) representa um utilizador do programa.

6.2.2 Use Case

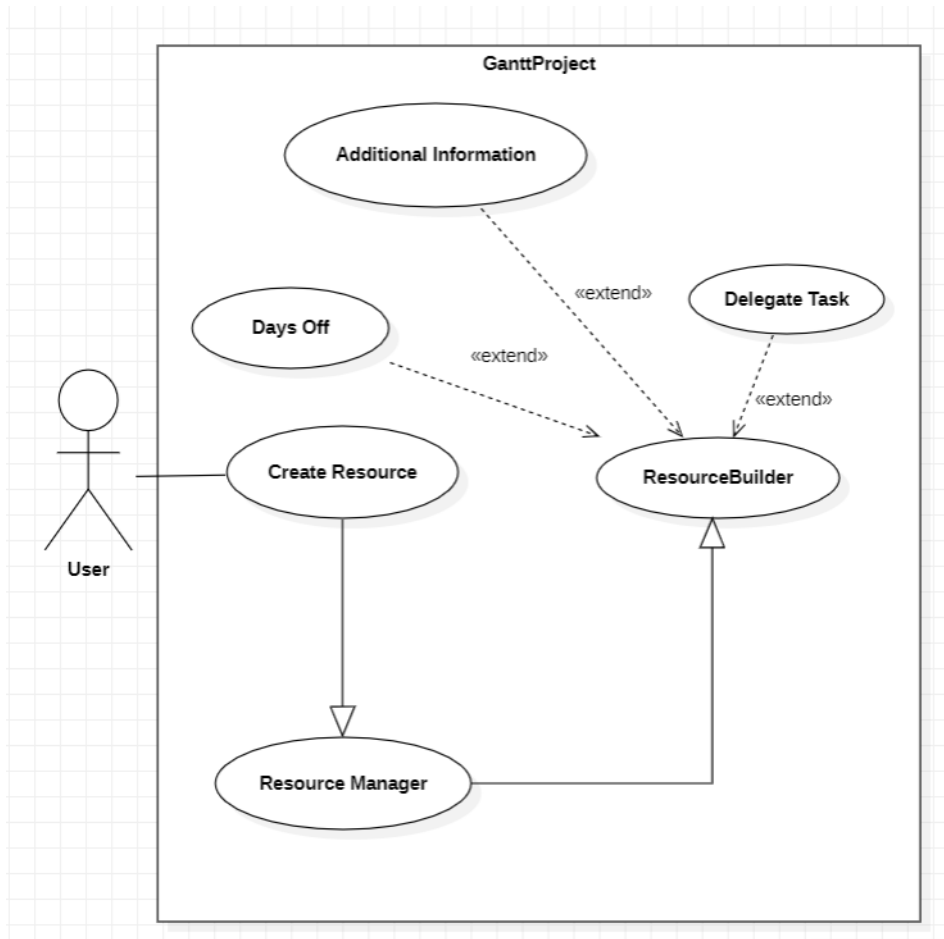


6.3 Create Resource – David Moreira, Review Joana Maroco

6.3.1 Descrição

Este Use Case mostra como um Recurso é criado, em que o seu autor é um "user" do programa.

6.3.2 Use Case

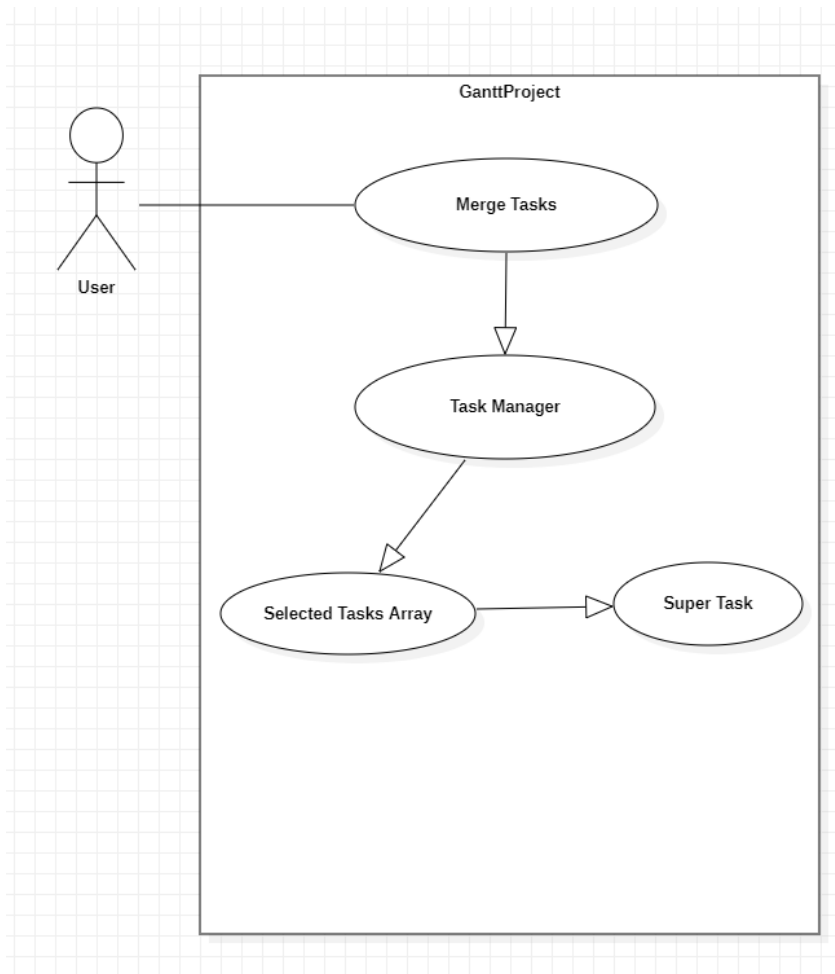


6.4 Merge Task - Joana Maroco, Review João Lopes

6.4.1 Descrição

Este Use Case representa uma função de Merge Task, ou seja, ele une uma série de Tasks a uma única tornando-as dependentes desta. O Ator Principal representa um utilizador comum do programa.

6.4.2 Use Case

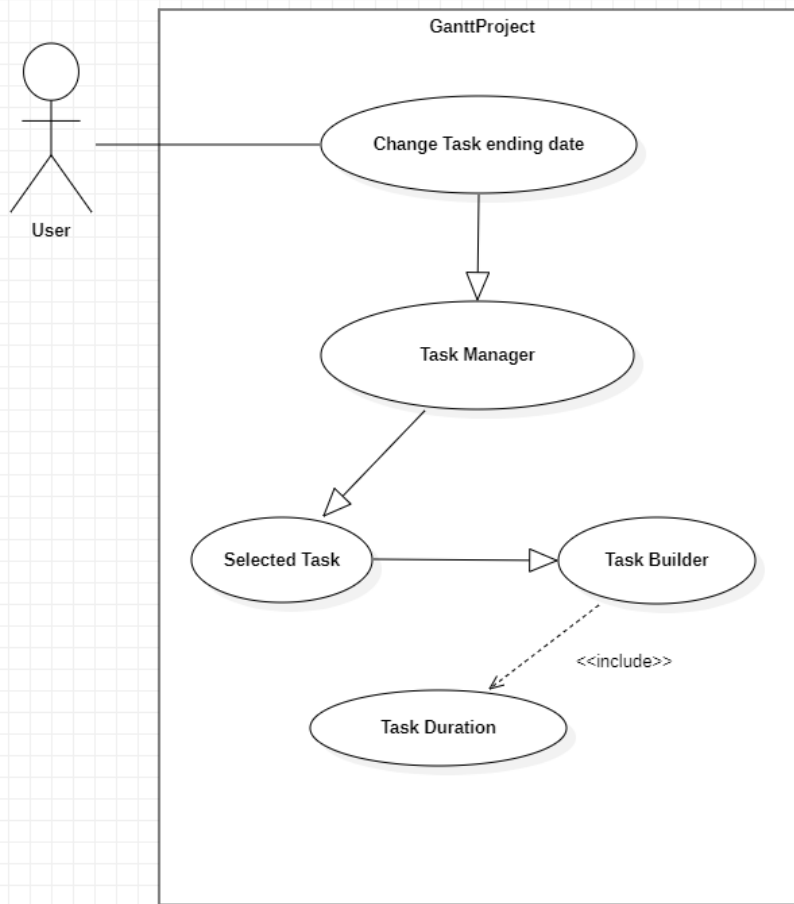


6.5 Change Task Ending Date – José Romano, Review David Moreira

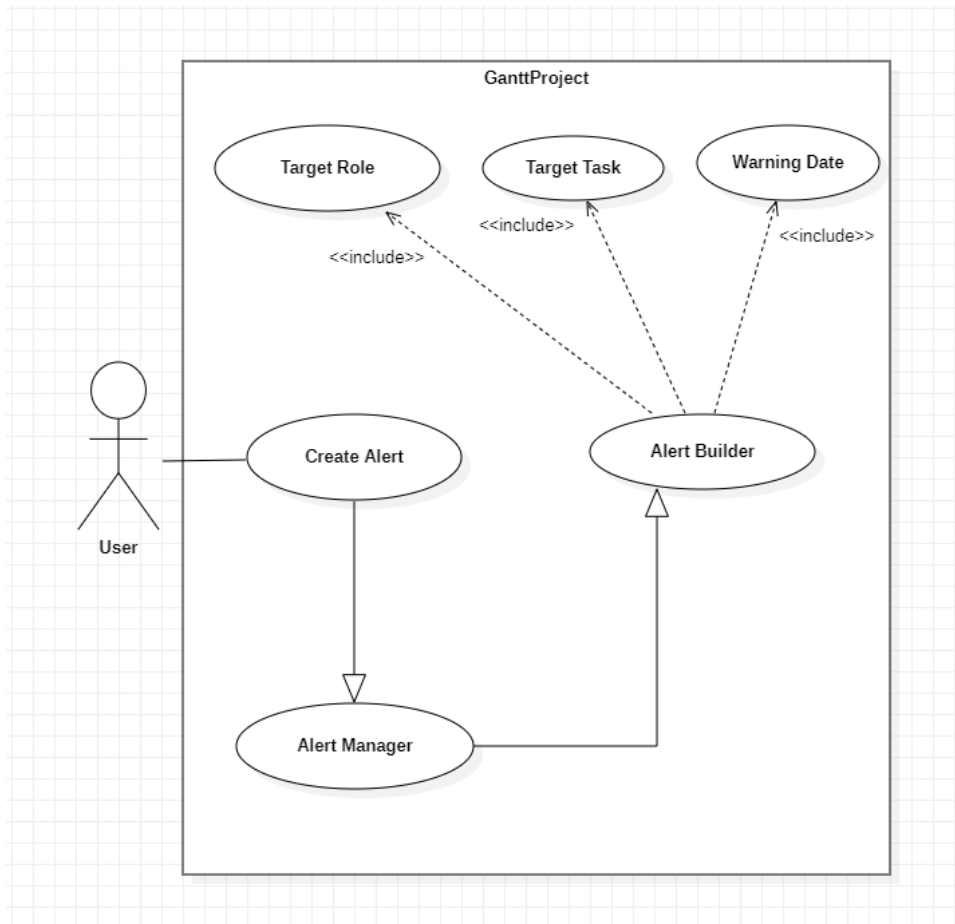
6.5.1 Descrição

O Use Case representa a mudança da data final de uma tarefa em específico. O autor principal é qualquer usuário do programa, apesar de ser mais apontado para o responsável pela tarefa em causa.

6.5.2 Use Case

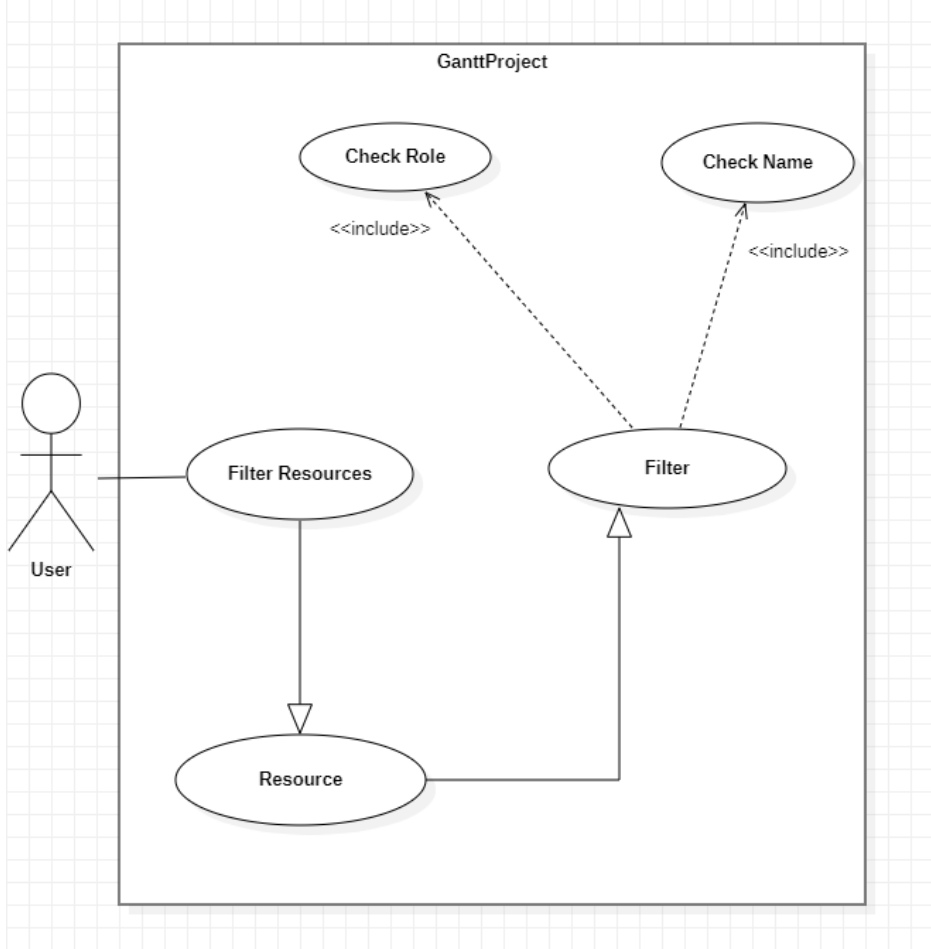


6.6 Alerta



Este Use Case representa a criação de um Alerta, uma das novas funções que decidimos implementar. O Ator Principal representa um utilizador comum do programa.

6.7 Filtro



Este Use Case representa a filtragem de um Recurso, uma das novas funções que decidimos implementar. O Ator Principal representa um utilizador comum do programa.

7 Metrics

7.1 Lines of Code – Joana Maroco

7.1.1 Explicação das métricas recolhidas

O conjunto de métricas oferecido pelo *plugin* que foi escolhido denomina-se de “*Lines of code metrics*”. Este conjunto foi desenvolvido seguindo a métrica que se denomina pelo mesmo nome – *Source Lines of Code* (SLOC), ou *Lines of Code* (LOC).

Ora, por linha de código entende-se qualquer linha de texto num ficheiro de código que não seja um comentário nem uma linha, mas também linhas de cabeçalho, caso existam número nas declarações ou em fragmentos de declarações dessa linha. Assim, este conjunto de métricas SLOC é usado para medir o tamanho de um programa contando, principalmente, com o número de linhas no ficheiro do código.

Para cada caso, observamos que foram medidos valores de métricas diferentes. O nome explica por si, no fundo, a medida usada. Temos:

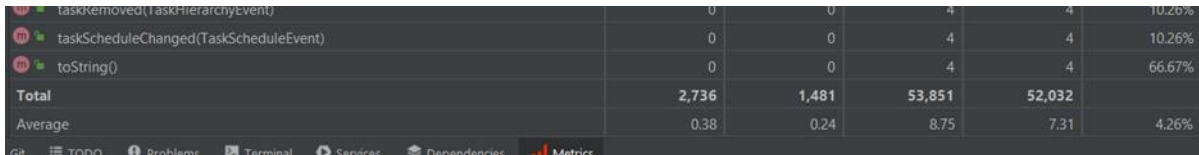
- Para Métodos:
 - Número de linhas em comentário (CLOC)
 - Número de linhas de *Javadoc* (JLOC)
 - Número de linhas de código (LOC)
 - Número de linhas que não são comentário (NCLOC)
 - Percentagem de linhas de código relativas (RLOC)
- Para Classes:
 - Número de linhas em comentário (CLOC)
 - Número de linhas de *Javadoc* (JLOC)
 - Número de linhas de código (LOC)
- Para Interfaces:
 - Número de linhas em comentário (CLOC)
 - Número de linhas de *Javadoc* (JLOC)
 - Número de linhas de código (LOC)
 - Número de linhas que não são comentário (NCLOC)
- Para Pacotes:
 - Número de linhas em comentário (CLOC)
 - Número de linhas em comentário (recursivo) (CLOC(rec))
 - Número de linhas de *Javadoc* (JLOC)
 - Número de linhas de *Javadoc* (recursivo) (JLOC(rec))
 - Número de linhas de código (LOC)

- Número de linhas de código (recursivo) (LOC(rec))
- Número de linhas de código-produto (LOCp)
- Número de linhas de código-produto (recursivo) (LOCp(rec))
- Número de linhas de código-teste (LOCt)
- Número de linhas de código-teste (recursivo) (LOCt(rec))
- Número de linhas que não são comentário (NCLOC)
- Número de linhas que não são comentário (produto) (NCLOCp)
- Número de linhas que não são comentário (produto, recursivo) (NCLOCp(rec))
- Número de linhas que não são comentário (teste) (NCLOCt)
- Número de linhas que não são comentário (teste, recursivo) (NCLOCt(rec))
- Para Módulos:
 - Número de linhas de *Javadoc* (JLOC)
 - Número de linhas escritas em *Groovy* (L(Groovy))
 - Número de linhas escritas em *HTML* (L(HTML))
 - Número de linhas escritas em *Java* (L(J))
 - Número de linhas escritas em *Kotlin* (L(KT))
 - Número de linhas escritas em *XML* (L(XML))
 - Número de linhas de código (LOC)
 - Número de linhas de código-produto (LOCp)
 - Número de linhas de código-teste (LOCt)
 - Número de linhas que não são comentário (NCLOC)
 - Número de linhas que não são comentário (produto) (NCLOCp)
 - Número de linhas que não são comentário (teste) (NCLOCt)
- Para Tipos de Ficheiro:
 - Número de linhas de código (LOC)
 - Número de linhas que não são comentário (NCLOC)
- Para todo o Projeto:
 - Número de linhas em comentário (CLOC)
 - Número de linhas de *Javadoc* (JLOC)
 - Número de linhas escritas em *Groovy* (L(Groovy))
 - Número de linhas escritas em *HTML* (L(HTML))
 - Número de linhas escritas em *Java* (L(J))
 - Número de linhas escritas em *Kotlin* (L(KT))
 - Número de linhas escritas em *XML* (L(XML))
 - Número de linhas de código (LOC)
 - Número de linhas de código-produto (LOCp)
 - Número de linhas de código-teste (LOCt)
 - Número de linhas que não são comentário (NCLOC)

- Número de linhas que não são comentário (produto) (NCLOCp)
- Número de linhas que não são comentário (teste) (NCLOCt)

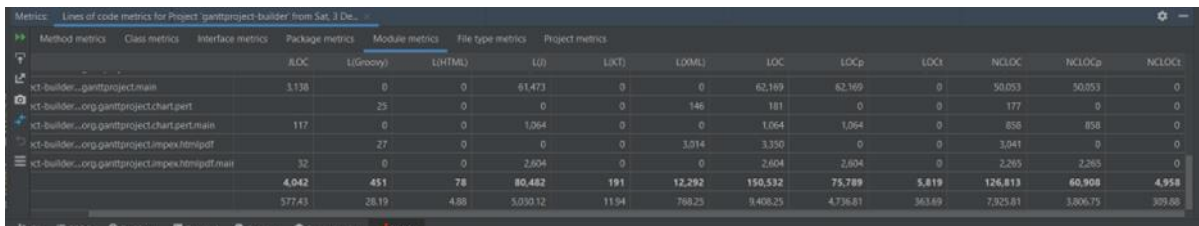
7.1.2 Potenciais locais “problemáticos”

- Para Métodos, observamos que **apenas** 4.26% (RLOC) do código são linhas consideravelmente relevantes para o código:



Method	LOC	RLOC	NCLOC	NCLOCp	NCLOCt
taskRemoved(TaskHierarchyEvent)	0	0	4	4	10.26%
taskScheduleChanged(TaskScheduleEvent)	0	0	4	4	10.26%
toString()	0	0	4	4	66.67%
Total	2,736	1,481	53,851	52,032	
Average	0.38	0.24	8.75	7.31	4.26%

- Para Módulos, apenas aproximadamente **metade** das linhas de código (LOC) fazem parte de código-produto (LOCp), ou seja, apenas metade é útil:



Module	LOC	LOCp	LOCt	NCLOC	NCLOCp	NCLOCt
ict-builder...gantproject.main	3,138	0	0	61,473	0	0
ict-builder...gantproject.chart.pert	29	0	0	146	181	0
ict-builder...gantproject.chart.pert.main	117	0	0	1,064	1,064	0
ict-builder...gantproject.inspect.html.pdf	27	0	0	3,014	3,350	0
ict-builder...gantproject.inspect.html.pdf.main	32	0	0	2,604	2,604	0
Total	4,042	451	78	80,482	191	12,292
Average	377.43	28.19	4.88	5,030.12	11.94	769.23

7.1.3 Relação mantida com os *Code Smells* identificados

Os problemas apresentados no ponto anterior, em conjunto com análise dos restantes resultados propostos pelo decorrer da métrica, conferem alguns dos pontos que foram apresentados pelo nosso grupo na fase anterior como *code smells*, tais como:

- “Código antigo em comentário”
- “Comentário a lembrar um raciocínio”
- “TODO comentado”
- “Excesso de comentários”
- “Classe com múltiplos métodos, definida dentro de uma interface”
- “Classe vazia”
- “Classe exaustivamente extensa”
- “Método demasiado grande e complexo”
- “Duplicação de código”

- “Métodos não utilizados (Dead Code)”
- “Valores que deviam ser constantes”

7.2 Complexity – David Moreira

7.2.1 Explicação das métricas recolhidas

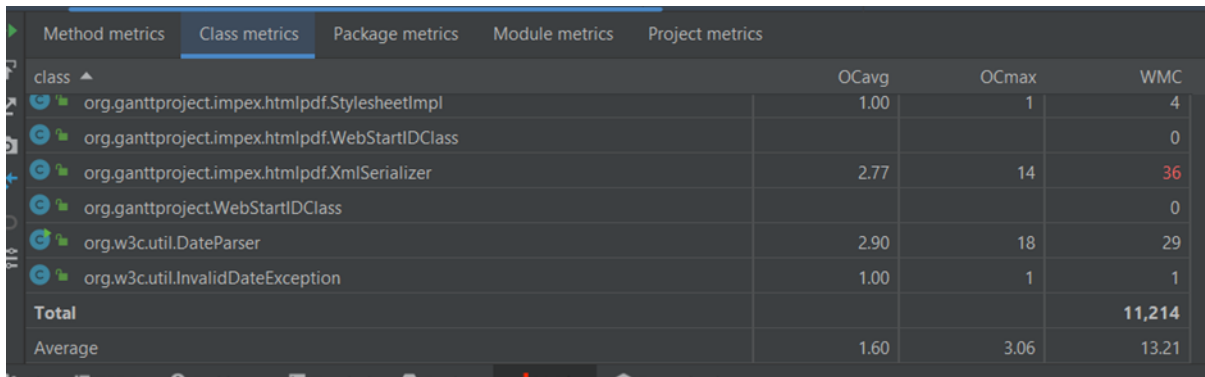
O conjunto de métricas oferecido pelo *plugin* que foi escolhido denomina-se de “*Complexity metrics*”, sendo que este conjunto foi desenvolvido seguindo a métrica que se denomina pelo mesmo nome.

Ora, o conjunto em referência é um conjunto de métricas de software usado para prever informações críticas sobre confiabilidade e manutenção de sistemas de software e contém todos os principais fatores responsáveis pela complexidade. Temos, como valores medidos:

- Para Métodos:
 - Complexidade cognitiva (CogC)
 - Complexidade ciclomática essencial (ev(G))
 - Complexidade de *design* (iv(G))
 - Complexidade ciclomática (v(G))
- Para Classes:
 - Complexidade média da operação (OCavg)
 - Complexidade máxima da operação (OCmax)
 - Complexidade do método ponderado (WMC)
- Para Pacotes:
 - Complexidade ciclomática média (v(G)avg)
 - Complexidade ciclomática total (v(G)tot)
- Para Módulos:
 - Complexidade ciclomática média (v(G)avg)
 - Complexidade ciclomática total (v(G)tot)
- Para todo o Projeto:
 - Complexidade ciclomática média (v(G)avg)
 - Complexidade ciclomática total (v(G)tot)

7.2.2 Potenciais locais “problemáticos”

- No geral, a média da complexidade nos diferentes valores do programa é mantida. No entanto, para Classes, o valor da complexidade do método ponderado (WMC) é muito mais elevado, o que indica que as classes, no seu geral, são muito complexas:



Method metrics	Class metrics	Package metrics	Module metrics	Project metrics
class		OCAvg	OCmax	WMC
org.ganttproject.impex.htmlpdf.StylesheetImpl		1.00	1	4
org.ganttproject.impex.htmlpdf.WebStartIDClass				0
org.ganttproject.impex.htmlpdf.XmlSerializer		2.77	14	36
org.ganttproject.WebStartIDClass				0
org.w3c.util.DateParser		2.90	18	29
org.w3c.util.InvalidDateException		1.00	1	1
Total				11,214
Average		1.60	3.06	13.21

7.2.3 Relação mantida com os *Code Smells* identificados

Os problemas apresentados no ponto anterior, em conjunto com análise dos restantes resultados propostos pelo decorrer da métrica, conferem alguns dos pontos que foram apresentados pelo nosso grupo na fase anterior como *code smells*, tais como:

- “Classe com múltiplos métodos, definida dentro de uma interface”
- “Classe vazia”
- “Classe exhaustivamente extensa”
- “Método demasiado grande e complexo”
- “Métodos não utilizados (Dead Code)”
- “Valores que deviam ser constantes”

7.3 Chidamber-Kemerer – João Lopes

7.3.1 Explicação das métricas recolhidas

O conjunto de métricas oferecido pelo plugin que foi escolhido denomina-se de “Chidamber-Kemerer metrics”.

Ora, este conjunto de métricas é usado, por norma, para medir algumas características de sistemas orientados a objetos, como classes, passagem de mensagens, herança e encapsulamento. Por outro lado, a falta de manutenção de software exige que sejam feitas no sistema existente.

Assim, para classes, este conjunto mede as seguintes métricas:

- Acoplamento entre objetos (CBO)
- Profundidade da árvore de herança (DIT)
- Falta de coesão de métodos (LCOM)
- Número de filhos (NOC)
- Resposta para a classe (RFC)
- Complexidade do método ponderado (WMC)

7.3.2 Potenciais locais “problemáticos”

Em média, temos um valor de 21.00 para a medida de resposta para a classe. Um valor elevado de RFC indica um maior número de falhas, cujas classes são mais complexas e mais difíceis de entender. A testagem e debugging será mais complicada:

Metrics: Chidamber-Kemerer metrics for Project 'ganttproject-builder' from S...							
Class metrics							
class	CBO	DIT	LCOM	NOC	RFC	WMC	
org.ganttproject.IMPex.htmlpdf.StyleSheetImpl	3	1	2	2	4	4	
org.ganttproject.IMPex.htmlpdf.WebStartIDClass	0	1	0	0	0	0	
org.ganttproject.IMPex.htmlpdf.XmlSerializer	29	2	3	1	65	36	
org.ganttproject.WebStartIDClass	0	1	0	0	0	0	
org.w3cutil.DateParser	10	1	1	0	36	29	
org.w3cutil.InvalidDateException	7	3	0	0	2	1	
Total						11,214	
Average	10.97	1.96	2.44	0.56	21.00	13.21	

7.3.3 Relação mantida com os Code Smells identificados

Os problemas apresentados no ponto anterior, em conjunto com análise dos restantes resultados propostos pelo decorrer da métrica, conferem alguns dos pontos que foram apresentados pelo nosso grupo na fase anterior como code smells, tais como:

- “Classe que devia ser um enumerado”
- “Classe com múltiplos métodos, definida dentro de uma interface”
- “Classe exaustivamente extensa”
- “Método demasiado grande e complexo”

- “Utilização de múltiplos ifs em vez que um único switch”
- “Duplicação de código”
- “Valores que deviam ser constantes”
- “Constantes não utilizadas”

7.4 MOOD – José Romano

7.4.1 Explicação das métricas recolhidas

O conjunto de métricas oferecido pelo plugin que foi escolhido denomina-se de “MOOD metrics”. Este conjunto foi desenvolvido seguindo a métrica definida por Fernando Brito e Abreu.

Ora, este conjunto foi concebido para fornecer um resumo da qualidade geral de um projeto orientado a objetos. Assim, são medidas as seguintes métricas (%) para todo o projeto:

- Fator de ocultação do atributo (AHF)
- Fator de herança de atributo (AIF)
- Fator de acoplamento (CF)
- Fator de ocultação do método (MHF)
- Fator de herança de método (MIF)
- Fator de polimorfismo (PF)

7.4.2 Potenciais locais “problemáticos”

Podemos analisar que, para os valores comuns, temos os valores de AIF, MHF e PF elevados. Isto levanta algumas questões perante , como referido, a herança dos atributos, o encapsulamento dos métodos e todo o polimorfismo do projeto.

Factor	Minimum	Maximum	Minimum Tolerance	Maximum Tolerance
MHF	12.7 %	21.8%	9.5 %	36.9%
AHF	75.2 %	100 %	67.7%	100%
MIF	66.4 %	78.5 %	60.9%	84.4%
AIF	52.7 %	66.3 %	37.4%	75.7%
COF	0 %	11.2 %	0%	24.3%
POF	2.7 %	9.6 %	1.7%	15.1%

Metrics: MOOD metrics for Project 'ganttproject-builder' from Sun, 4 Dec 202... x

Project metrics

project	AHF	AIF	CF ▲	MHF	MIF	PF
project	88.11%	76.06%	2.10%	45.40%	51.73%	29.56%

7.4.3 Relação mantida com os Code Smells identificados

Os problemas apresentados no ponto anterior, em conjunto com análise dos restantes resultados propostos pelo decorrer da métrica, conferem alguns dos pontos que foram apresentados pelo nosso grupo na fase anterior como code smells, tais como:

- “Classe que devia ser um enumerado”
- “Classe com múltiplos métodos, definida dentro de uma interface”
- “Método demasiado grande e complexo”
- “Utilização de múltiplos ifs em vez de um único switch”
- “Duplicação de código”
- “Métodos não utilizados (Dead Code)”
- “Valores que deviam ser constantes”
- “Constantes não utilizadas”

7.5 Martin Packaging

7.5.1 Explicação das métricas recolhidas

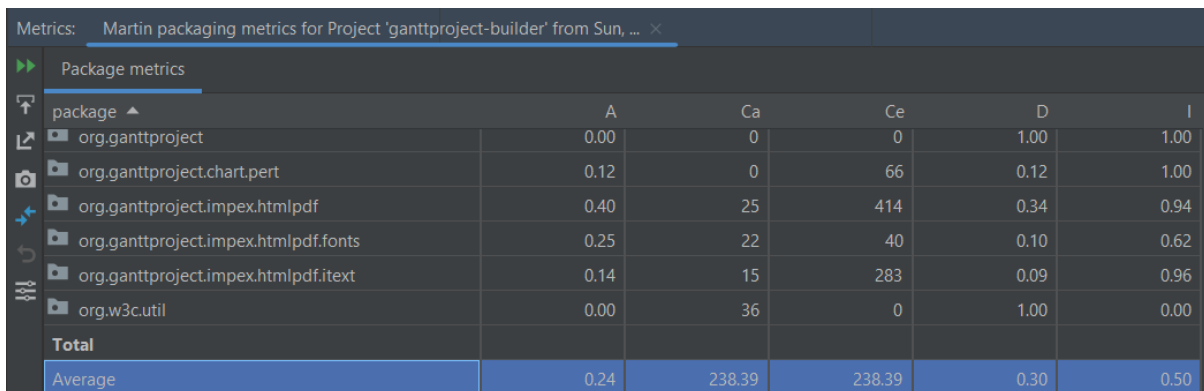
O conjunto de métricas oferecido pelo *plugin* que foi escolhido denomina-se de “*Martin packaging metrics*”. Este conjunto foi desenvolvido seguindo o grupo de métricas proposto por Robert “Uncle Bob” Martin em 1994. O foco é estudar a relação entre pacotes do projeto.

Temos, para cada pacote, os seguintes valores medidos:

- Abstração (A)
- Acoplamentos aferentes (Ca)
- Acoplamentos eferentes (Ce)
- Distância da sequência principal (D)
- Instabilidade (I)

7.5.2 Potenciais locais “problemáticos”

Podemos analisar que, para os valores comuns, temos o valor de *Ce* elevado (>20), o que sugere instabilidade no pacote; e o valor de *I* é intermédio ($0.3 < I < 0.7$), indicando estabilidade intermédia no pacote, o que é longe do ideal.



The screenshot shows a table titled 'Metrics: Martin packaging metrics for Project 'ganttproject-builder' from Sun, ...'. The table has columns for package names and metrics A, Ca, Ce, D, and I. The packages listed are org.ganttproject, org.ganttproject.chart.pert, org.ganttproject.impex.htmlpdf, org.ganttproject.impex.htmlpdf.fonts, org.ganttproject.impex.htmlpdf.itext, and org.w3c.util. A 'Total' row and an 'Average' row are also present.

package	A	Ca	Ce	D	I
org.ganttproject	0.00	0	0	1.00	1.00
org.ganttproject.chart.pert	0.12	0	66	0.12	1.00
org.ganttproject.impex.htmlpdf	0.40	25	414	0.34	0.94
org.ganttproject.impex.htmlpdf.fonts	0.25	22	40	0.10	0.62
org.ganttproject.impex.htmlpdf.itext	0.14	15	283	0.09	0.96
org.w3c.util	0.00	36	0	1.00	0.00
Total					
Average	0.24	238.39	238.39	0.30	0.50

7.5.3 Relação mantida com os Code Smells identificados

Os problemas apresentados no ponto anterior, em conjunto com análise dos restantes resultados propostos pelo decorrer da métrica, conferem alguns dos pontos que foram apresentados pelo nosso grupo na fase anterior como *code smells*, tais como:

- “Classe que devia ser um enumerado”
- “Classe com muitos métodos, definida dentro de uma interface”
- “Classe vazia”

8 Dificuldades

Ao longo do desenvolvimento do projeto, passamos por múltiplas dificuldades, algumas das quais necessitavam de rescrever grande parte do código do GanttProject, ou que envolviam não conseguir sequer iniciar o programa em algumas máquinas, sem, antes, perder horas a tentar perceber o problema. Todos estes acontecimentos foram as principais causas dos atrasos e impossibilidades no desenvolvimento.

O principal problema foi a alteração da *branch* utilizada. Após esta mudança, vimo-nos obrigados a converter completamente a nossa abordagem ao programa. A principal diferença era a forma como as Tasks e os HumanResources são apresentadas. Antes, o código que organizava a parte gráfica era feito totalmente em *Kotlin*, enquanto que, na nova *branch*, é feito através de uma biblioteca do java (java.swing).

Esta mudança de *branch* foi considerada o maior problema, uma vez que fomos obrigados a voltar a analisar o código inteiro, de forma a localizarmos a zona que precisamos de alterar. O código novo é ainda mais confuso e caótico do que o anterior, sem qualquer documentação para apoiar, o que o torna ainda mais difícil de analisar e requer mais tempo. Isto especialmente tendo em conta que queremos que o nosso código seja fácil de compreender e que não seja necessário alterar radicalmente, caso um outro desenvolvedor queira adicionar mais funções por cima.

A biblioteca do java utilizada, o java.swing, também tem muita pouca documentação disponível, obrigando-nos a passar horas a pesquisar online, o que atrasou bastante o projeto, especialmente porque muitas das classes do ganttproject são extensões de classes do java.swing, o que faz com que a maior parte dos seus métodos não estejam disponíveis para visualização ou edição. Isto acontece principalmente na construção do painel lateral, onde são visíveis as *Tasks/HumanResources*.

Apesar de implementarmos funcionalidades relativamente simples, o código original não nos permite fazê-lo, sem antes sermos obrigados a rescrever classes inteiras, o que ocupa muito para além do tempo disponível e necessita de bem mais recursos.

Um exemplo de um destes problemas surgiu durante a implementação de filtros nos *HumanResources*. Para fazer isto, cada vez que é adicionado um *HumanResource*, somos obrigados a aplicar o filtro e, caso este novo item



Fig. 8 – Erro encontrado

não pertença à solução filtrada, é lançada uma exceção (Fig.8).

Esta exceção, segundo a nossa análise do código, é causada pelo facto de um *HumanResource* ser selecionado, quando criado. Isto faz com que o programa procure o caminho para a seleção do recurso e receba um vazio. No entanto, algo assim é impossível de alterar, uma vez que ele recebe esta informação diretamente de uma classe do java.swing.

9 Conclusão

Concluindo, nesta fase do projeto, o grupo começou por se juntar e decidir as ideias futuramente discutidas e aprovadas pelo professor das aulas práticas. De seguida, passou-se ao planeamento e análise do código, à criação dos Use Cases, à avaliação das Metrics e à implementação das funções em causa.

Todos estes processos foram bastante demorados, especialmente a análise e implementação do código, devido a muitas das dificuldades que foram surgindo, tais como a mudança de *branch*, a falta de comentários, a dificuldade em perceber o código e o facto de, inicialmente, o programa não correr em algumas das máquinas.

Todo o grupo demonstrou bastante esforço e empenho, especialmente na face de todos os problemas. E, independentemente desses constrangimentos, o desenvolvimento do projeto decorreu de forma organizada e sempre através de comunicação, pelo que, apesar de o resultado não ser o desejado, sentimos que os objetivos foram cumpridos.